Forms Builder

Version 25.2

Help Guide

July 2025

Anthology Inc.

5201 Congress Avenue
Boca Raton, FL 33487
Main: +1.561.923.2500
Support: +1.800.483.9106
www.anthology.com

© 2025 Anthology Inc. All rights reserved.

ANTHOLOGY, the Anthology logo, and CampusNexus are exclusive trademarks of Anthology Inc. Microsoft and Microsoft Dynamics are trademarks of Microsoft Corporation. Other third-party trademarks or service marks are property of their respective owners. Information is subject to change.

CONFIDENTIALITY NOTICE:

The information contained in this document is confidential. It is the property of Anthology Inc. and shall not be used, disclosed, or reproduced without the express written consent of Anthology Inc.

Revision History

Date	Description
July 2025	Release of document for Forms Builder Version 25.2. Refer to What's New.

Contents

Get Started **26**

What's New 27

Forms Builder Version 25.2 27

Forms Builder Version 25.1.1 27

Forms Builder Version 25.1 27

Forms Builder Version 25.0 27

About Forms Builder 28

Database Providers and Authentication 28

Workspaces 29

Form Designer 30

Sequence Designer 30

Export/Import 31

Internationalization 31

Custom Content 31

Settings 31

Basic Steps 32

Known Limitations 34

Grid Component - Date Format Issues 34

DateTime Values in PDF 35

Breaking Change: Variables no longer allowed as Model Values 35

formInstance.UserInfo not populated when using Azure AD 35

Creating Forms 35

Payment Country in Credit Card Payment Component 35

Using Multiselect for Single Property Collections 35

Entities in "Show All" List Not Fully Defined 35

Rendering Sequences 36

Accessing the Sequence List 36
Managing/Modifying Workflow Definitions 36
Student Address Changes 36
Miscellaneous 36
Limitations for Mobile Devices 36
Required Skills 37
Prerequisite Knowledge 37
Advanced Forms Builder and Workflow Development 37
Accessibility 37
Installation 39
API Keys 40
Using Earlier Product Versions 40
Apply a New SSL Certificate to STS 42
Best Practices for a Successful Go-Live 43
Logging 43
Workflows 44
Form Data 46
E-Sign Sequences 46
Application Initialization 47
Persisted Workflow Instances 48
CampusNexus CRM Integrations 51
Prerequisites 51
Integrate Forms Builder with CampusNexus CRM 11.1 or Later 51
Integrate Workflow Composer with CampusNexus CRM 11.1 or Later 52
Run an OData Query in the Web Client 52
View Lookup Query Results 53
Set Up the Database Environment 54
CampusNexus CRM Environment 54
Verify the Setup 55
Anthology Student Environment 55

Verify the Setup 56
CampusNexus CRM and Anthology Student Environment 56
Update Forms Builder URLs (HTTPS or HTTP) 57
Upgrade Considerations 59
Designer **62**
Custom Content 63
Image Files 64
Video Files 65
CSS Files (Style Sheets) 65
JavaScript 68
Navigation Bar Logos 69
Sequence Logos 70
Remove Local Custom Content 71
Export/Import 72
Prerequisites 72
Export Sequences 72
Import Sequences 74
Form Designer 77
AutoSave for Forms 83
Binding 84
Binding in Forms Builder 84
Model Property and Arguments 84
Specify Model Value and Create Argument 85
Default Arguments 85
Casing of Arguments 87
Components 88
Adobe Sign 89
Properties 89
Working with the Adobe Sign Component 92
Calendar/Scheduler 94

Enhancements in Forms Builder 3.7 94
Enhancements in Forms Builder 3.5 94
Properties 97
Minimal Calendar/Scheduler 114
Calendar/Scheduler Initialized by Model Data 118
Calendar/Scheduler Initialized by OData Query 122
Calendar with EntityState Property 124
CAPTCHA 135
Prerequisites 135
Properties 136
Checkbox 139
Properties 139
Credit Card Payment 143
Properties 143
Payment Processing with PayPal 152
Payment Processing with ACI 163
Payment Processing with IATS 165
Payment Processing with TouchNet 166
Date Picker 168
Properties 168
Troubleshoot a Date Picker 173
Date Time Picker 175
Properties 175
DocuSign 181
Properties 181
Working with the DocuSign Component 184
Localization of DocuSign Sequences 188
Allow Sequential Signing 188
Drop-down List 189
Properties 189

Drop-down List with Value List ... 200
Drop-down List with Workflow Initialized List ... 204
Drop-down List with Workflow Initialized List via ExecuteQuery ... 208
Drop-down List with Workflow Initialized List and Template ... 214
Drop-down List for Extended Properties ... 217
Expand/Collapse Panel ... 223
Properties ... 223
File Upload ... 227
Enhancements in Forms Builder 3.5 ... 227
Properties ... 227
Grid ... 231
Grid Property Settings ... 233
Grid Columns Properties ... 238
Grid Initialized via OData Query ... 258
Grid Bound to an Entity ... 261
Grid Bound to Custom Model Data (non-Entity) ... 263
Grid Bound to Results of ExecuteODataQuery ... 268
Grid Populated by ExecuteQuery ... 272
CRM Grid for One-to-Many Relationships ... 282
Google Map ... 289
Prerequisites ... 289
Properties ... 290
HTML ... 293
Properties ... 293
Access Model Values Using JavaScript ... 295
DatePicker Widget ... 296
Set Default Values for Form Fields ... 298
Bind HTML to an Argument ... 300
Hyperlink ... 302
Properties ... 302

IFrame 304
Properties 304
JSON Debug Info 307
Properties 307
Label 310
Properties 310
Modify the CSS for the Label 311
Locale 313
Properties 313
Locale Assignment Using Workflow 315
Masked Text Box 316
Properties 316
Multiselect 321
Properties 321
Custom Multiselect with Value List 330
Custom Multiselect with Workflow Initialized List 334
Numeric Text Box 338
Properties 338
Popup 344
Properties 344
Progress Bar 348
Properties 348
Use Case: Compare Numeric Data 351
Radio Button 353
Properties 353
Specify a Default Selection 358
Create a Validation Item 358
Repeater 360
Properties 360
Example: Create a Repeater 364

Binding Values in a Repeater Layout 367
Repeater Workflow Arguments 368
SignNow 369
Properties 369
Working with the SignNow Component 371
Single-select Search 373
Properties 373
Adjust the Search Grid Columns Property 378
TabStrip 380
Properties 380
Creating a TabStrip 382
Text Box 383
Properties 383
Textarea 391
Properties 391
Time Picker 396
Properties 396
Tooltip 402
Properties 402
Typeahead 407
Properties 407
View Summary 412
Properties 412
Control Property Settings 416
Binding Properties 416
Notation for Array Variables 416
AngularJS Expression Sandbox Security 417
Database Tables for Property Settings 417
Update of Properties 417
Editable Properties 417

Copy and Paste Controls 421
Limitations 422
Custom Styles 423
Create a Style for a Label Control 424
Replace the Logo 425
Video: Combine Form Sections on a Form 425
Exclude a Component from a PDF 426
Date Formats 427
Example 1: Admissions Deposit - Received Date Field 427
Example 2: Date Picker Component 428
Date & Time Values and Offsets 429
Delete Forms 430
Fields 431
Select the Database Provider 431
Find Fields in an Entity 432
Search for Fields 433
Show All Fields 433
Form Sections 434
Style Form Sections 434
Apply a Style to a Form Section 434
Apply a Style to a Form Section Title 435
Hide Empty Space for a Form Section with E-Sign Components 436
Visible Property 436
Layout Enhancements 437
Flexible Column Layout 437
Merged Form Sections 437
Reusable Form Sections 438
Create and Save a Form Section 438
Edit a Form Section 440
Add a Form Section to a Form 440

Edit a Form Section in a Form 441
Delete a Form Section 442
Delete a Form Section from a Form 442
Multiselect for Single Property Collections 443
Ethnicities List 443
Programs List 445
Save Forms and Form Properties 448
Unsaved Changes Dialog 448
Form Property Settings 449
School Defined Fields 452
SerializableDynamicObject 457
Dynamic Objects 457
Serializable Objects 457
Dictionary Objects 457
Validation on Form Save 459
Boolean Properties 459
Model Property 459
Validation Errors for School Defined Fields 460
Validation Error for Id Property on File Upload 461
HTML Syntax Checking 461
Indeterminate Flags 461
Internationalization 463
Definitions 463
Internationalization and Localization in Forms Builder 463
Culture Scripts for Kendo Components 464
Localization of DocuSign Forms 465
Steps to Localize Sequences 466
Sequence Designer 473
Analytics Tracker 477
Enable Form Analytics 477

Map Forms Builder Fields to Zuko ... 478
Delete Sequences ... 479
Delete Persisted Workflow Instances ... 480
Delete Sequence Instances ... 480
Delete All Instances ... 481
Footer and Header ... 482
Footer ... 483
Header ... 484
Nav Button Position ... 484
Open the Workflow for a Sequence ... 486
Role ... 487
Role Configuration ... 487
Save Sequences ... 489
Unsaved Changes Dialog ... 489
Save Sequence As ... 490
Validation Checks ... 490
Sequence Identifier ... 492
Assign a Sequence Identifier to a Sequence ... 493
Create a Unique Sequence Identifier ... 493
Themes ... 495
Apply a Theme to a Sequence ... 497
Welcome and Confirmation Forms ... 498
Create a Custom Welcome Form ... 498
Create a Custom Confirmation Form ... 499
Settings ... 501
Entity and Property Visibility Settings ... 507
Import File Validation ... 509
Workflows ... 511
Activities for Forms Builder ... 512
CreateAdobeSignRequest ... 513

Properties 514
CreateDocuSignRequest 516
Properties 516
CreateSignNowRequest 519
Properties 519
GetAdobeSignConfig 522
Properties 522
GetAdobeSignRecipientStatus 523
Properties 524
GetAdobeSignSignedDocument 525
Properties 526
GetAttachments 527
Properties 528
GetDocuSignConfig 530
Properties 531
GetDocuSignRecipientStatus 533
Multi Route Workflow Example 533
Properties 540
GetDocuSignRecipientFormData 541
542
Properties 542
GetSignedDocument 544
Properties 544
GetSignNowConfig 547
Properties 547
GetSignNowRecipientStatus 549
Properties 549
GetSignNowSignedDocument 551
Properties 553
LookupUser 554

Properties 555
ObjectCollectionCopier<> 557
Properties 558
Using SaveEntity vs. SaveEntityCollection with ObjectCollectionCopier<> 559
Example: RepeaterSerializableDynamicObject with SaveEntity 559
Example: RepeaterSerializableDynamicObject with SaveEntityCollection 562
ObjectCopier<> 566
Properties 567
Example: RepeaterSerializableDynamicObject 568
Example: SerializableDynamicObject 571
ObjectToEntityCollectionCopier<> 573
Properties 574
Example: RepeaterSerializableDynamicObject with SaveEntityCollection 575
PrintUrlToPdf 579
Workflow Example 579
Usage Notes 582
Sequences with File Upload Component 582
Multi Page Forms 582
PrintUrlToPdf Times Out 582
Incompatible Components 583
Auxiliary Service 583
Properties 584
Page Breaks in PDF Files 587
TranslateText 588
Properties 589
VerifyCardPayment 591
Properties 594
WaitForFormBookmark 595
Properties 597
Collect IP Address 598

Create, Get, and Save Entity Activities ... 601
CreateEntity<> ... 601
GetEntity<> ... 601
SaveEntity<> ... 603
Best Practice to Prevent DbUpdateConcurrency Exceptions ... 604
Custom Validations ... 606
Single Validation ... 606
Placement of the Custom Validation ... 608
Multiple Validations ... 609
Multiple Validations Items When Processing a Grid ... 614
Grid Using Entity Collection Activities ... 621
Add, Edit, and Save Records in a Collection ... 621
Link a Portal Account to a Student Record ... 631
Multi Route Forms ... 638
Step 1: Create forms for the sequence ... 638
Step 2: Build the sequence ... 640
Step 3: Define transitions in Workflow Composer ... 640
Step 4: Render and test the sequence ... 647
Passing Values to an End State Form ... 649
Example ... 649
State Machine Workflows ... 651
State ... 651
Transition ... 655
Shared Trigger Transition ... 657
Update a Form After Creation of a Sequence ... 659
Adding an Entity to a Workflow ... 659
Workflows for CampusNexus CRM ... 662
CampusNexus CRM Events and Objects ... 662
Workflow Activities for CampusNexus CRM ... 662
Renderer ... **664**

Azure AD Authentication 665
Prevent Student Users from Accessing Form Designer 665
Workflow Examples 668
Login and Account Creation via Portal 676
Renderer Web.Config Changes 677
Azure AD Claims 678
Embed a Form on a Website 680
Procedure 680
Forms Builder Form as External Link in Portal 682
Configure an IFrame Link in the Portal Configuration Tool 682
Modify the Workflow to Resolve the AuthGuid (for anonymous sequences) 684
Link Sequences to Portal Document Center 687
Update Documentation Links in Portal 687
Associate Document Statuses with Documents 687
Multiple Renderer URLs 689
Campus Example 689
Multiple Renderer URLs for Multiple Student STS Instances 690
Add Custom Style Sheets and Images 691
Associate Sequences With Uploaded CSS Files 692
Select Style Sheets Using Workflow Activities 693
Preview and Update a Form Sequence 697
Redirects for Rendered Sequences 698
Anonymous Sequences 698
Authenticated Sequences 698
Default Navigation Paths within Sequences 699
Renderer Authentication 701
Renderer Connection Strings 703
Renderer Media Variables 705
Renderer URL Query Parameter 706
Syntax 706

Pass a URL Query Parameter to a Workflow ... 706
Example ... 706
Pass "addonQueryParams" via the URL ... 708
Example ... 708
Sequence List ... 710
Vanity URL ... 711
Use Cases ... **713**
Credit Card Payment Form ... 714
Prerequisites ... 714
Create the Form Sequence ... 714
Modify the Workflow ... 727
Test the Rendered Sequence ... 734
E-Signature Forms ... 738
Electronic Signatures vs. Digital Signatures ... 738
Electronic Signatures ... 738
Digital Signatures ... 738
Adobe Sign Forms ... 740
Adobe Sign Settings ... 741
To Integrate With Adobe Sign ... 741
Adobe Sign Workflow Sample - Single Signer ... 743
Prerequisites ... 743
Create the Workflow ... 743
Adobe Sign Workflow Sample - Multiple Signers ... 756
Prerequisites ... 756
Test the Multiple Signer Feature ... 757
Chrome Browser Settings for Multiple Signers ... 758
Create the Workflow ... 759
DocuSign Forms ... 778
DocuSign Settings ... 779
Setting up Forms Builder for DocuSign ... 779

DocuSignWorkflow Sample - Single Signer 785
Prerequisites 785
Enhancements in Forms Builder 3.6 785
Create the Workflow 786
DocuSignWorkflow Sample - Multiple Signers 804
Prerequisites 804
Values in the Type Property 807
Test the Multiple Signer Feature 808
Set Up DocuSign Account Preferences 808
Create the Workflow 808
Move from Test to Production 829
Log in to DocuSign 830
Manage Tab 830
Permissions 831
SignNow Forms 832
SignNow Settings 833
Set Up Forms Builder for SignNow 833
Troubleshoot SignNow Settings 835
SignNow Workflow Sample - Single Signer 837
Prerequisites 837
Create the Workflow 837
SignNow Workflow Sample - Multiple Signers 853
Prerequisites 853
Test the Multiple Signer Feature 854
Chrome Browser Settings for Multiple Signers 854
Create the Workflow 855
FERPA Form 877
Build the Form 877
Create a Sequence 882
Edit the Workflow 883

Validate the Data in the Web Client 890
Submit the Release of Information Form 891
Confirm the Updates in the Web Client 892
Check the Renderer Log 893
Request for Information Form 894
Build the Form 894
Create a Query in the Web Client 903
Add the Query to the Form 905
Create a Sequence 906
Edit the Workflow 907
Submit the RFI Form 916
Validate the Data in the Web Client 917
Check the Renderer Log 918
Troubleshooting 919
Audits 920
Forms - Add, Edit, Delete 920
Sequences - Add, Edit, Delete 921
Settings - Edit 921
Imports 922
Azure Blob Storage 923
Basics 928
Developer Tools 929
Console 929
DOM Explorer (IE) / Elements (Chrome) 929
Fiddler 930
Field Length Changes Impacting Forms Builder 930
Steps to Modify the Field Length 930
List of Field Length that Affects Forms Builder 930
HTTP Status Codes 933
4xx Client Errors 933

5xx Server Errors 933
Installation Errors Related to CRM Contracts.dll 934
Test Web Services for Designer and Renderer 935
Log Files 937
Enhanced Logging in Forms Builder 3.4 and Later 937
Best Practices for Logging 938
Location of Log Files 939
Forms Builder Logs 939
Event Logs 939
LogLine/LogObject Activities 939
Common Errors and Solutions 940
Logging in Azure 944
Best Practices for Logging 945
LogLine/LogObject Activities 945
Troubleshoot E-Sign Forms 946
Write the PDF to Disk 946
Error Code "TAB_OUT_OF_BOUNDS" - DocuSign 946
E-Sign Document is Blank 947
PrintUrlToPdf Times Out 948
Hyperlinks Display with Target URL in Parentheses 948
Webhook Responses for Multiple Signer Sequences 948
Troubleshoot Fields and Components 950
Missing Keys 950
Validation Error on Text Boxes 950
Invalid Property Names in Grids 951
Troubleshoot Rendered Sequences 952
Workflow Error on Rendered Forms 952
Server Error - Workflow Aborted 952
Forms are Skipped 953
E-Sign Document is Blank 953

Disappearing Grid Rows on Edit 953
Slow Loading of Authenticated Sequences 954
Visually Examine Data in Renderer 955
Debug - Show Generated JSON Model 956
DbUpdateConcurrency Exception 958
Access Denied Error 958
Student Address Changes 959
Troubleshoot Workflows 960
Exception Handling 960
Workflow Definition Is Not Displayed 960
Workflow Error Indication on Rendered Forms 960
Common Workflow Errors 961
Validation Messages 965
Assign Ids 965
SQL Query to Determine the UserName for a Persisted Workflow 966
Troubleshooting Tips & Tricks 968
Angular JS 969
Angular JS Resources 969
Validation Regex Property in Forms Builder 970
Understanding OData 971
Data Model 971
Command Model 972
Query Model 973
OData Queries 973
Rest APIs - Swagger 976
Log File Locations & Names 979
On Premise Installations 979
Forms Builder 979
Workflow Saved Events 979
Workflow Saving Events 980

Web Client 980
Workflow Composer 981
Azure Cloud Subscriptions 982
Azure Blob Storage 982
Azure Storage Explorer 982
Tips 986
Best Practices for Logging 986
LogObject 987
LogLine 988
CampusLink web.config File 988
Reading Log Files 988
Service Module Host 990
When is Service Module Host Used? 993
V1 Contracts 993
V2 Contracts 994
Forms Builder 995
Task Scheduler Occurrence Event 995
Forms Builder Access Errors 997
Web Client URL 997
CMCDataServices URL 998
Activity Errors 999
Configuration Issues 1003
Packages 1003
Connection Strings 1004
Email Configuration 1005
JSON Debug 1006
Workflow Execution 1008
Task Scheduler Occurrence Event 1008
Workflow Validation of Business Process in Web Client 1010
Resources **1014**

Analytics Tracker - Quick Reference 1015
Request a Partner Referral 1015
Map Fields 1015
CampusNexus CRM Entities 1021
Contact 1022
Lead 1023
Anthology Student Entities 1024
Academic Year 1025
Account Status 1028
Address Type 1029
Admissions Deposit 1030
Applicant Areas Of Study 1032
Applicants 1033
Area Of Study 1035
Area Of Study Requirement Rule 1037
Area Of Study Type 1038
Document 1039
Document Transcript Request 1041
ISIR Verification 1042
Pending Applicant 1043
Pending Applicant Area of Study 1048
Pending Applicant Ethnicity 1049
Pending Applicant Previous Education 1050
Pending Prospect Inquiry 1051
Pending Prospect Inquiry Ethnicity 1054
Prospect Inquiry 1055
Prospect Inquiry Lead Source 1056
Prospect Inquiry Program of Interest 1057
Student 1058
Student Address Changes 1061

Student Advisor 1062
Student Agency Branch 1063
Student Area of Study 1064
Student Athletic Detail 1065
Student Course 1066
Student Credit Card 1067
Student Disability Detail 1069
Student Enrollment Period 1070
Student Ethnicity 1072
Student Extra Curricular Activity 1073
Student Ledger Card Transaction 1074
Student Previous Education 1076
Student Relationship Address 1078
Student Service Type 1080
Student Transfer Credit 1081
Student Veteran Detail 1083
Student_Staff Picture 1084
Cheat Sheets 1085
Exposed Events 1086
Form Events 1086
Component Events 1086
Examples of External Event Listeners 1094
GitHub Repositories 1095
Forms Builder Sequence Templates 1096
Workflow Samples 1096
Integration Samples 1096
General GitHub Resources 1096
OData Queries 1097
Build Queries Using Views for Anthology Student 1098
Create a View and Export a Query 1098

Populate the Lookup Query in Form Designer 1100
Build Queries Using the Data Model 1102
View the Metadata 1102
Example: Student Entity Metadata (Excerpt) 1103
Execute a Query 1104
Modify a Query 1104
Change the Sort Order 1105
Remove the "select" Option 1105
Use the "$expand" Option for Navigation Properties 1105
Change the "$filter" Option 1105
Build a Cascading Query Using AngularJS 1106
OData Syntax Reference 1106
Populate the Lookup Query in Form Designer 1106
Run Queries in Web Client for CampusNexus CRM 1109
Build Queries for Occupation Insight 1110
Rendered Form 1110
Form Layout 1110
Drop-down List - States 1111
Drop-down List - Occupations 1113
Grid - SalariesByState 1115
Grid - OccupationStateProjections 1117

Get Started

Forms Builder is a web-based application for the creation, design, publication, and management of form workflows. It enables organizations to customize complex processes without the need for expensive programming, custom web design, or service costs.

Forms Builder enables users to:

- Create forms for every constituent at the institution: new applicants, existing students, faculty, and staff
- Build forms that can be used multiple times across any campus

Target users include IT staff who respond to business requests for adjustments in applicant processes, or business users who work directly with their administrative staff to build and deploy custom processes.

Related Help Systems

https://help.anthology.com/Content/Home.htm

What's New

Forms Builder Version 25.2

Forms Builder has been updated to resolve client-reported issues and security vulnerabilities.

Forms Builder 25.2.0 Release Notes

Forms Builder Version 25.1.1

Forms Builder introduces an Import File Validation setting to ensure the system verifies whether an XML file has been externally modified during imports. By default, this setting enables you to validate file integrity.

Forms Builder 25.1.1 Release Notes

Forms Builder Version 25.1

Forms Builder has been updated to resolve the following issues:

- The http calls which are connected to the external applications have been modified.
- The SNAT PORT issues have been fixed.

Due to security reasons, we are restricting to modify the XML file after it is exported from any environment. While importing any XML file, we are validating the XML file with signature. If signature mismatches, system will not allow to import the XML to any environment. Now, before upgrading to the Forms Builder 25.1, customers must complete the export and import sequences in their environments. So after the upgrade, without valid signature element in XML, the XML cannot be imported.

Forms Builder 25.1 Release Notes

Forms Builder Version 25.0

Forms Builder has been tested for compatibility with Student Database regarding field length changes. A list of field length changes that affect Forms Builder is mentioned below:.

- Field Length Changes Impacting Forms Builder

If you modify the exported Forms Builder file and then attempt to import it again, the file will not be uploaded. Before upgrading to Forms Builder version next release, all forms must be upgraded to Forms Builder version 25.0.

- Export/Import

Forms Builder 25.0 Release Notes

About Forms Builder

The next generation of Forms Builder uses OData (Open Data Protocol) to access and expose data from various data sources such as the Anthology Student database, the CampusNexus CRM database, or both. Forms Builder has access to the entire data model for these applications. The adapters built for Forms Builder 2.x to expose data properties and implement business logic rules are no longer applicable and do not exist in Forms Builder 3.x or later.

Forms Builder 3.x and later provides greater flexibility to the user and enables integration with Workflow ("Form Flow"). Each new sequence that is created and saved automatically saves a corresponding workflow definition that can be further customized or edited in Workflow Composer.

Important
OData calls must be made from **within** Forms Builder or Workflow.
OData calls via ApiProxy outside of Forms Builder or Workflow are **not** supported.

The following components are used with Forms Builder 3.x and later:

- Web client for CampusNexus CRM and/or web client for Anthology Student
- Workflow Composer
- CMCFormsBuilderDesigner_V3
- CMCFormsRenderer_V3

Designer is installed on **port 9002** by default. When you access your Forms Builder URL with this port number, the Home page is displayed.

Renderer is installed on **port 9003** by default. When you access your Forms Builder URL with this port number at http://<server>.<domain>:9003/#/Sequencelist, the Sequence List is displayed.

The URLs and port numbers are customizable during installation.

Database Providers and Authentication

CampusNexus CRM, Anthology Student, or both products can be database providers for Forms Builder. The web.-config files of Form Designer and Renderer contain "enabled" attributes for the products. Depending on the options selected in Installation Manager, the "enabled" attributes are set to "true" for each product.

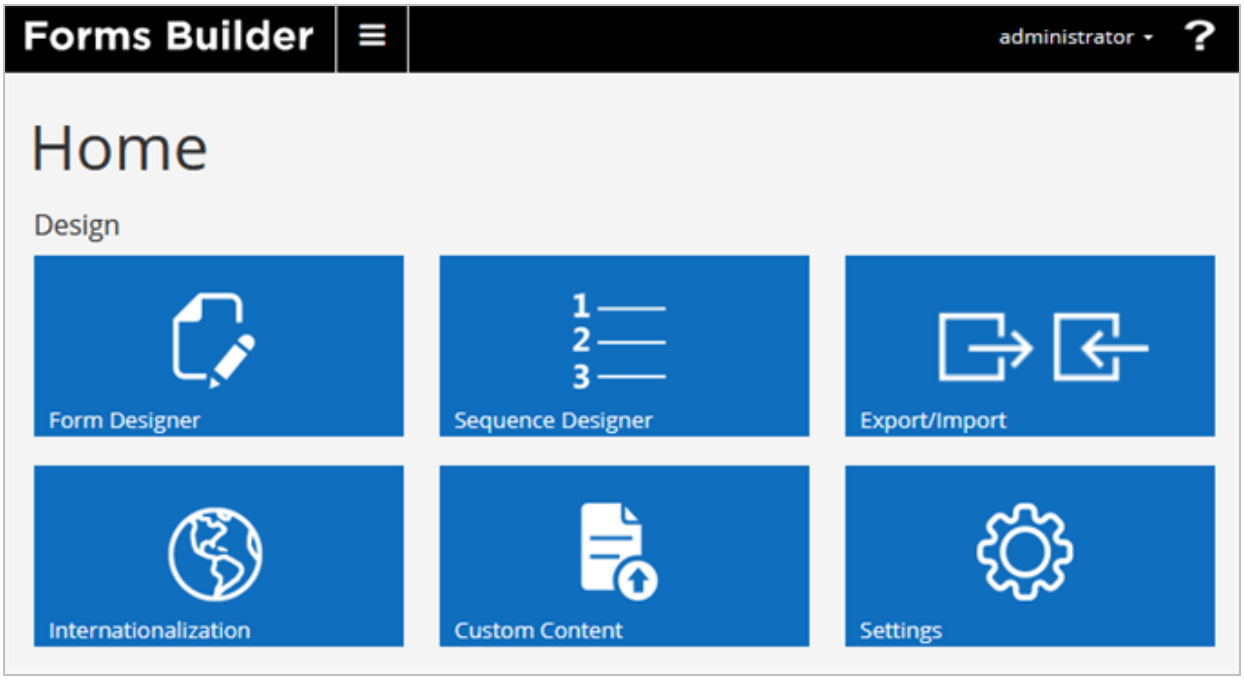
Authentication of Forms Builder users is based on the security tokens issued by the Student STS for Anthology Student and the Contact STS for CampusNexus CRM. The <trustedIssuers> section of the web.config file for Forms Builder Renderer has empty thumbprints for the Student STS and/or Contact STS.

- When Anthology Student is enabled during installation, the Student STS thumbprint is populated.
- When CampusNexus CRM is enabled, the Contact STS thumbprint is populated.
- When both products are enabled, both thumbprints are populated. Single sign-on allows Forms Builder users to access both databases.

The user can select the database service provider in Form Designer (refer to Select Provider).

Workspaces

After logging in to the Forms Builder application, the **Home** page is displayed.



Use the Forms Builder header elements as follows:

- Select the product name to return to the Forms Builder home page from any other page.
- Select the **hamburger button** to access a menu of the navigation options (tiles) from any other page.
- Select the drop-down arrow next to the name of the logged in user to **Sign Out** or to view the **About Forms Builder** window.
- The **About Forms Builder** window contains the following information:
    - Forms Builder Version
    - Connections for database access
    - Web Client URLs for Anthology Student, CampusNexus CRM, and Occupation Insight (as applicable)
    - Install Date
- Select the question mark icon to access this help system.

The tiles on the Forms Builder home page link to the following workspaces:

- Form Designer
- Sequence Designer
- Export/Import
- Internationalization
- Custom Content
- Settings

Form Designer

The process of building forms starts with Form Designer. You access the URL and port assigned to CMCFormsBuilderDesigner_V3 and select the Form Designer tile on the home page. The first time you open Form Designer, the metadata service of the installed web client (Anthology Student and/or CampusNexus CRM) is invoked and the entities of the CampusNexus data model are loaded into Forms Builder.

The building blocks in Form Designer include:

- Forms (flyout menu)
- Tabs for Fields, Components, and Form Sections
- Layout pane
- Property Settings pane

You can build forms using any of the available CampusNexus entities. You select an entity, select an associated field, and drag the field into the form layout pane. For more information, refer to Fields.

You can add custom controls to any form. The custom controls are on the Components tab and include check boxes, hyperlinks, text boxes, and so on.

You arrange fields and/or components in the Layout pane based on your selected column layout for the form. You use the Control Property Settings pane to assign properties to the items in the Layout pane. Properties include options such as label, tooltip, class, and so on. The values for properties can be data types such as Boolean, strings, numeric, arrays, and expressions.

The **Model** property is especially powerful and flexible. Any field or component can be bound using the Model property. A bound control is a control whose source of data can be any value assigned to a workflow argument. The designer of the workflow determines the source of the value. It could be a constant or a calculated value from the workflow. It also can be a value from another control or from external JavaScript code using the global variable vmModelsRef. The Model property is an Angular JS container that allows you to pass data to a workflow regardless of whether the control is a based on a CampusNexus entity/field or custom component. You can use the Model property to create arguments to hold bidirectional data. Data can be passed both to and from a workflow, to and from another control, or to and from external JavaScript.

Sequence Designer

Once you have created and saved individual forms, you combine the forms into a sequence using Sequence Designer. You specify properties for sequences such as authentication, title, name, theme, end state form, and URL.

For every sequence Forms Builder automatically creates a workflow definition based on the initial layout of the sequence. The name of the workflow definition matches the sequence name. Forms Builder passes data to the workflow through arguments in the Model property of fields and components. Using Workflow Composer, you can customize the workflow definition and insert workflow activities that manipulate the data. When an activity starts executing, the values of all its arguments are evaluated. Workflow Composer enables you to apply customized logic to forms, write data to the database, and trigger specific events. For more information, refer to Open the Workflow for a Sequence and State Machine Workflows.

End users, for example students or leads, access the published forms and submit data that can then be stored in a database because of the modified workflow definition including applicable activities for saving/updating data.

The markup for the sequences is saved in the Anthology Student or CampusNexus CRM database. Unlike Forms Builder 2.x, Forms Builder 3.x and later does not require its own database. For more information, refer to Set Up the Database Environment.

Sequences can be anonymous or non-anonymous. Non-anonymous sequences require users to provide credentials and be authenticated (refer to Renderer Authentication).

Export/Import

The Export/Import workspace enables you to export and import sequences and workflows from one environment and to another. For more information, refer to Export/Import.

Internationalization

The Internationalization workspace enables you to generate .pot files and import .po files. The .pot files contain translatable text that is extracted from sequences. The .po files contain translations for specific languages. For more information, refer to Internationalization.

Custom Content

The Custom Content tile on the Home page enables users to upload files to the CustomContent table in the database. These files can then be used to customize forms. In previous releases, on premises customers could also use the file system reference method, and for Azure customers the database method was the only way of customizing forms.

As the file system reference method is no longer available from version 4.1, on premises and Azure customers must only use the database method as it simplifies Forms Builder upgrades.

For more information, refer to Custom Content.

Settings

The Settings workspace is used to configure attributes and resources for features such as themes, e-signature providers, reCAPTCHA, Credit Card Payment, and custom error messages. For more information, refer to Settings.

Basic Steps

The following table outlines the basic steps for getting started with Forms Builder. Each step points to reference information within this Help system. After you have performed the tasks and read the related information, you'll be able to build more complex forms, and you'll know where to find help.

Step	Task	Description	Help Topic
1	Check pre-requisites	Verify the environment setup and configuration.	Installation
2	Explore the UI	Familiarize yourself with the Forms Builder architecture and explore the workspaces.	About Forms Builder
3	Build a form	In Form Designer: a. Select **New Form**. b. Select the **Fields** tab. c. If your database provider is Anthology Student, select the **Student** entity. If your database provider is CampusNexus CRM, select the **Contact** entity. d. In the Layout pane, create a **2-column** section. e. Select the **First Name**field and drag it into the Layout pane. Accept the default properties. f. Select the **Last Name** fields and drag it into the Layout pane. Accept the default properties.	Fields Tab
4	Build a form (continued)	In Form Designer: a. Select the **Components** tab. b. Add an **HTML** component to the form. c. In the Property Settings pane, select the HTML field, and specify some text. d. Save the form.	Components Tab HTML Component
5	Build a sequence	In Sequence Designer: a. Select **New Sequence**. b. Drag the new form into the Layout pane and specify some sequence properties. c. Save the sequence.	Sequence Designer
6	View a sequence	Open the Sequence List in a different browser (or in an incog-nito session of the same browser) and view the rendered sequence. Keep the browser window open.	Sequence List

Step	Task	Description	Help Topic
7	Modify a form	Return to Form Designer and modify the property settings for a field or component and save the form again.	Control Property Settings
8	Review a sequence	Return to the browser window with the rendered form and reload/refresh the webpage. Verify that the modified properties are displayed as expected.	Troubleshooting
9	Open the workflow	Launch Workflow Composer and open the workflow.	Open the Workflow for a Sequence
10	Explore the workflow	In Workflow Composer, explore the states, transitions, and arguments that were automatically created for the sequence.	State Machine Workflows Activities for Forms Builder
11	Modify the workflow	Modify the Entry area of the first State in the workflow as follows: a. Add a **LookupUser** activity. b. Add a **GetEntity<>** activity. *Optional:* Modify the Action area of the Next transition in the workflow as follows: a. Add a **SaveEntity<>** activity. Save and publish workflow.	Create, Get, and Save Entity Activities LookupUser GetEntity<> (Workflow Help) SaveEntity<> (Workflow Help) Save and Publish Workflows (Workflow Help)
12	Test and verify	Return to the browser window with the rendered form and reload/refresh the webpage. Verify that the name of the student/lead is now pre-populated in the form fields. If you added the SaveEntity<> activity, verify that the data table is updated, and the data entered and saved on the form is displayed in the client for Anthology Student or CampusNexus CRM.	Troubleshooting
13	Clean up	Remove the test form and sequence.	Delete Forms Delete Sequences

For examples end-to-end procedures for building forms, sequences, and workflows, please refer to Use Cases.

Known Limitations

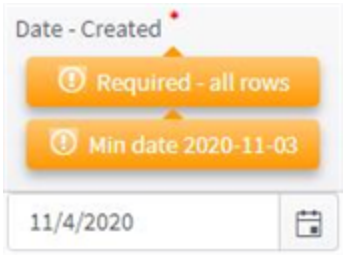
The following are known limitations in Forms Builder version 3.7 that will be addressed in a future version. Also listed below are known issues and workarounds that are applicable to the current version.

Grid Component - Date Format Issues

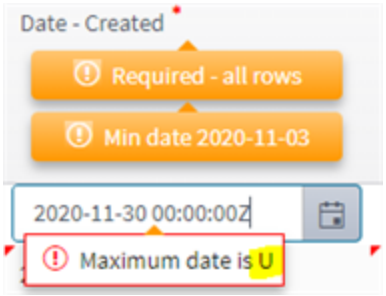
- When a date field is used in a Grid component and the Format value is not specified (i.e., default format {0:d} is used), validations for min/max values may not be triggered properly.

Prop Name	Title	Format	Type	Attributes	Required
City	City		string		false
State	Dropdown - State		Dropdown List		false
CreatedDateTime	Date - Created		date		true

  For example, a validation error may be appear even if the date entered falls within the specified min/max value range.



- When the date format type "u" is used (refer to Kendo Date Formatting), the validation is done properly; however, the wrong value will be displayed in the in-cell message. The message at the column level above will be correct. The first date validation check is always the Min date. It doesn't clear until the user goes to the next row.



DateTime Values in PDF

When the server hosting the Forms Builder application is located in a different time zone than the clients and clients enter DateTime values in sequences that are converted to PDF on the server, the DateTime values in the PDF do not match the values entered by the clients.

Currently, the only remedy is to set the server to same time zone as the clients.

Breaking Change: Variables no longer allowed as Model Values

All Model bindings defined or used in a workflow must be **arguments** (not variables). Any sequences that are bound to variables will no longer work.

formInstance.UserInfo not populated when using Azure AD

In Forms Builder 3.4 and later, if Azure AD is used for authentication, pre-built forms such as RFIs that use `formInstance.UserInfo`variables will no longer provide the capability to pre-populate form fields with user information. To address this issue, refer to Azure AD Claims.

Creating Forms

Payment Country in Credit Card Payment Component

The Credit Card Payment component currently does not pass Payment Country values other than "Unites States of America" to the PayPal site. If any other Payment Country values are specified, either the Country drop-down list will not be displayed, or the user will have to select the country in the payment form on the PayPal site. Payment Country is an optional property for the Credit Card Payment component.

Using Multiselect for Single Property Collections

There are several instances within the Anthology Student and CampusNexus CRM models whereby multiple selections of a single property are needed. Currently, the ability to achieve this capability via the dragging of the applicable properties from the model onto the Form Designer Layout pane is not supported. Instead, the custom Multiselect component from the Components tab in Form Designer must be used to achieve this capability. For more information, refer to Multiselect for Single Property Collections.

Entities in "Show All" List Not Fully Defined

The Lookup Query attribute in the metadata has been defaulted for all entities displayed in the default list.

For other entities in the model that are only displayed when “Show All” is selected, the Lookup Query attribute may not have any default values. For these entities the desired OData query will need to be manually entered. For information about creating OData queries, refer to OData Queries and https://www.odata.org/. The Control Type may also not be defined, in which case a custom component may need to be used.

Rendering Sequences

Accessing the Sequence List

A server error occurs when the Sequence List is opened in the same browser as Form Designer or Sequence Designer. To avoid this error, use different browsers, e.g., open the Sequence List in Google Chrome while Form Designer is open in Internet Explorer.

You can also use the incognito (or private browsing) mode in any browser. For example, you can open Form Designer in Chrome and then open the Sequence List in another instance of Chrome in incognito mode. Note that you must open another instance of the browser (not just another tab) in incognito (private browsing) mode.

For sequences that use components like CAPTCHA and DocuSign (which rely on third party applications), the browser security must be set so that certificates are accepted.

Managing/Modifying Workflow Definitions

Student Address Changes

The message *"You made a change to the address. Do you want to save the old address as another address type?"* is displayed when a student's address and related fields of the StudentEntity are modified. To avoid this message, include an Assign activity in the workflow with the following assignment: `studentEntity.StudentAddressAssociation = Cmc.Nexus.Common.Entities.StudentAddressAssociation.IgnoreInStudentAssociation`

The entity field `studentRelationshipAddressEntity.IsPermanentAddress` has been changed to `studentRelationshipAddressEntity.IsSeasonalAddress`. For more information, refer to Student.

Miscellaneous

Limitations for Mobile Devices

The following features are currently not supported on mobile devices:

- Calendar/Scheduler
- View Summary

The following components have limited functionality on a mobile device and must be used in horizontal orientation:

- Grid
- DocuSign

Required Skills

The Forms Builder application is intended to be used by staff members with the following knowledge and skills.

Prerequisite Knowledge

- Understanding of business processes
- Understanding of Anthology Student application and schema and/or CampusNexus CRM application and schema
- Awareness of .NET technologies and understanding of VB.NET
    - Creating variables, assigning data types, and a basic understanding of development languages
- Awareness of:
    - Windows Workflow Foundation
    - CSS themes
- SQL Knowledge
    - Ability to create SQL jobs, call stored procedures and write queries
- General development knowledge of variables, arguments, control logic, exception handling, debugging, etc.

Advanced Forms Builder and Workflow Development

Expertise in the following is recommended:

- AngularJS (expressions)
- OData
- REST (JSON)
- Bootstrap (themes)
- Workflow tracking and persistence
- TSQL skills to write stored procedures

Accessibility

Forms Builder (Designer and Renderer) underwent an accessibility compliance assessment in accordance with the provisions and guidelines set forth in Section 508 and WCAG 2.1 at the Level AA standard. The non-

conformant test findings were corrected, and color schemes and branding were aligned with other Anthology products.

Disclaimer: Customization of Forms Builder voids the Web Content Accessibility Guidelines (WCAG) conformance statement from Anthology.

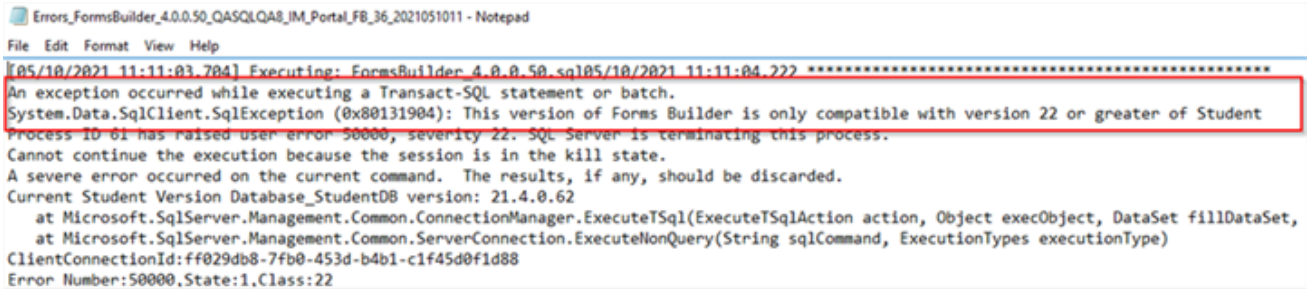
Installation

The topics in this section provide important information about the setup and configuration of Forms Builder in different environments.

> **Important**
>
> In any environment, after a fresh installation of Forms Builder, before you can publish the URLs of rendered form sequences for end users, you must log in to Form Designer and select the **Settings** tile.
>
> In the Settings workspace:
>
> - Provide your e-signature partner credentials
> - Provide your Google Map API key
> - Update the reCAPTCHA test key
> - Update the Payment test keys
>
> Updating the Error Message Text is recommended but not mandatory. For more information, refer to Settings.

Forms Builder 4.0 is compatible with **Anthology Student 22.0** or higher.

If Anthology Student 22.0 or higher is not found during the installation of Forms Builder 4.x or later, an exception occurs and is displayed in the log.



API Keys

To enhance the security of Anthology Inc. products, API keys were added to the products released in May 2018 and later. An API key is a secret token that is submitted with a web service request to identify the origin of the request. The key for the consumer of the service needs to match the key of provider of the service, otherwise access to the service is rejected. The API key is unique for each customer.

The API key is an AppSetting in the web.config files of applications built on the CampusNexus framework. It uses the following syntax:

```
<add key="apiKey" value=""/>
```

The API key is the same key that is used in the Package Manager screen of Installation Manager.



Installation Manager 1.18 and later automatically adds the key value to the web.config files during installation of the following product versions:

- CampusNexus CRM **12.0** and later
- Anthology Student **19.0** and later
- Contracts & Activities **19.0** and later
- Portal **19.0** and later
- Regulatory **10.1** and later
- Financial Aid Automation **6.2** and later
- Workflow Composer **2.6** and later

Using Earlier Product Versions

If you are using products with earlier versions in combination with any of the above listed versions, the API key must be manually added to the web.config file of the older version.

If there is no key defined in the web.config file, a default key that exists in the authentication provider will be used.

Depending on the product and version, you may need to overwrite the default key with your customer-specific key value.

— OR —

If the `appSettings` section does not contain the `<add key="apiKey" value=""/>`line, add the line and specify your key value.

The following is a snippet of a web.config file for Forms Builder Renderer 3.4:

```
  <appSettings>
    <add key="ConfigureCampusNexusWcfProxy" value="true" />
    <add key="ConfigureCVueNexusWcfProxy" value="true" />
    <!-- Following will be populated when Crm is enabled for Forms Builder -->
    <add key="CmcNexusCrmWebUrl" value="http://<server:port>/" />
    <add key="PaymentProvider" value="pilot-payflowpro.paypal.com" />
    <add key="AuxiliaryServiceBaseUrl" value="" />
    <!-- Following should be set to true only in Azure environments where the Auxiliary service
is installed and required. -->
    <add key="UseRemotePdfConversionService" value="false" />
    <!-- Following sets a time before conversion to PDF starts. Default 500, increase if blank
documents on a slow server. -->
    <add key="ViewCreatorDefaultStartConversionTimerInMilliseconds" value="" />
    <add key="ApiKey" value="<Your API key value>" />
  </appSettings>
```



If the API keys are not set up correctly, an "Access denied" error will be found in the Renderer log, for example, when a Forms Builder workflow calls an Anthology Student activity.

Apply a New SSL Certificate to STS

Perform the following steps to set up a new SSL certificate with Forms Builder websites. These steps are valid for any STS used by Forms Builder.

1. Ensure that a new certificate is installed in IIS.

   > ⚠ The SSL certificate must be a purchased public certificate which can be verified by common root certificate authorities such a GoDaddy or Verisign, among several. Self-signed certificates cannot be used except in a closed testing environment.

2. Determine the **Thumbprint** and **Subject Name** of the new certificate.

   You can run following command in Windows PowerShell with correct path to retrieve that information:

   ```
   Get-ChildItem -Path cert:\LocalMachine\My
   ```

3. Update the binding information for CMCPortalSTS to refer to the new certificate.

   In IIS, select **CMCPortalSTS > Edit Binding > Edit HTTPS** and choose the correct certificate.

4. Modify following two AppSettings in the web.config of CMCPortalSTS:

   - **SigningCertificateName**

     Copy the latest Subject Name from PowerShell in this location.

   - **CertThumbprint**

     Copy the latest Thumbprint from PowerShell in this location.

5. Modify following key in the web.config files of CMCFormsBuilderDesigner_V3 and CMCFormsRender_V3:

   - **Thumbprint under TrustedIssuers**

     Copy the latest Thumbprint from PowerShell in this location.

Best Practices for a Successful Go-Live

After installing or upgrading Forms Builder and developing form sequences in a test environment, it is important to observe some best practices to ensure a smooth transition to a production environment. The following recommendations are intended to help ensure a successful go-live.

Logging

Important

Log files may contain confidential information such as user names and passwords, account information, etc. It is your responsibility to protect sensitive user and system data.

To mitigate the risks of exposing sensitive data, observe the following best practices:

- Set the log level in production environments to the lowest, least detailed log level. Increase the log level in test environments only when needed. Reset the log level when testing is complete.

  The default logging provider used by Anthology Inc. products is NLog. NLog allows you to configure log targets, levels, rules, layouts, etc. through configuration. To configure logging, modify the NLog.config file in the application's executing directory. For Web applications, this file exists alongside the web.config file.

- When LogLine or LogObject workflow activities are used to capture entities that contain sensitive information, remove such activities as soon as testing is complete.

  We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

  If, instead you followed the recommendations, and the development machine NLog minLevel is set to "Info" and all logging is done at the "Information" level, and the production machine NLog minLevel is set to "Error" (default), then nothing needs to be done because the production machine will not log "Information" LogLine or LogObject activities. The additional benefit is that the logging is still available if a problem can only be found in a production machine and lowering the NLog minLevel to "Info" temporarily (and restarting the app pool) will allow troubleshooting.

NLog Levels

In a **test** environment, the NLog.config might have been set to **Info** for debugging purposes. The Info level captures detailed messages written by LogLine workflow activities.

In a **live** environment, the NLog.config should be set to **Error** level so that none of the LogLine information appears in the log files and performance is improved. By changing the NLog level you don't need to remove LogLine activities from the workflows. If troubleshooting is required in a live environment, you can set the NLog level temporarily to Info.

1. In the CMCFormsRenderer_V3 folder, locate the **NLog.config** file.
2. In the <rules> section of the NLog.config file, set **minLevel** to **Error**.

```
<rules>
   <logger
name = "*"
minLevel = "Error"
     writeTo = "file" />
</rules>
```

3. **Save** the NLog.config file.

> **In Forms Builder 3.5.1 and later**, the ability to set NLog levels in the Settings workspace of Form Designer is removed to prevent conflicts with Azure log configurations. Azure logs are stored in customer-specific tables. **If your Forms Builder deployment is in an Azure environment, contact Anthology Inc. obtain access to the Azure log tables or to request changes in the NLog settings.**

Several logger.debug statements and client-side logs have been modified to **Info** level to make them available to help debug issues in an Azure environment since in an Azure environment the log level is set to Info level for all products. The Info level is set for logs related to:

- Site Warmup
- LookupUser
- Account Controller
- PDF creation and e-sign documents
- Payment processing for PayPal, ACI, and IATS

Workflows

Use formInstance.ValidationMessages

Check the value of the ValidationMessages property on all workflow activities that have this property. The value should be set to formInstance.ValidationMessages to ensure that all form validation errors are captured in a live environment.

Don't Use Condition=True in Transitions

Set the value of the Condition field in transitions to Not formInstance.ValidationMessages.HasErrors or leave it blank. Do not use a Condition of True.

If the Condition of a transition evaluates to False, the transition will not occur. If the Condition is blank, the transition will occur. The value True can cause unexpected results.

Check the Placement of Custom Validations

Custom Validations using the CreateValidationItem activity should be placed in transitions after the WaitForFormBookmark activity and before the Condition.

Use Caution with Hard-Coded Values

When using hard-coded lookup values, any minor difference between test and live environment could cause errors with invalid or non-existent values. For example, if a DocumentType value is hard-coded as 43 instead of using a LookupReferenceItem activity on the DocumentType property, the value might not return the expected results in a different environment.

Place Save Activities in the Final Transition

Many of the save activities such as SaveDocument should be done in the final transition (not in the form/state itself). The save should occur after the WaitForFormBookmark activity. The transition's Condition for completion can then validate that no errors are returned by the activity before completing the sequence.

Initialize Values in the First State of a Workflow

Initialization of values based on an authenticated user should be done in Entry section of the first form/state in a workflow.

Remove the Back Transition in Complex Workflows

If a workflow performs multiple activities that create entities (such as enrollments) followed by other activities in later forms that rely on previous activities (e.g., registering for a class), remove the Back transition option to avoid duplicates (such as creating a 2nd enrollment).

Create Short Sequences and Simple Workflows

To avoid rollback issues with long sequences and complex workflows, it is best to have sequences with fewer forms and workflows designed to do only one specific thing.

Prevent DbUpdateConcurrency Exceptions

A DbUpdateConcurrency error occurs when an attempt is made to update an instance of an entity via a Save activity, but that instance has been modified by another user in the time from when the instance was initially retrieved in the workflow to the point in time when the Save activity executes.

Example of a DbUpdateConcurrency exception in a Renderer log file:

2018-02-27 13:30:16.7645 54 Error Cmc.Nexus.Crm.Workflow.SaveDocument System.ServiceModel.FaultException`1[System.ServiceModel.ExceptionDetail]: Store update, insert, or delete statement affected an unexpected number of rows (0). Entities may have been modified or deleted since entities were loaded. See http://go.microsoft.com/fwlink/?LinkId=472540 for information on understanding and handling optimistic concurrency exceptions. (Fault Detail is equal to An ExceptionDetail, likely created by IncludeExceptionDetailInFaults=true, whose value is: System.Data.Entity.Infrastructure.DbUpdateConcurrencyException: Store update, insert, or delete statement affected an unexpected number of rows (0). Entities may have been modified or deleted since entities were loaded. See http://go.microsoft.com/fwlink/?LinkId=472540 for information on understanding and handling optimistic concurrency exceptions. ----> System.Data.Entity.Core.OptimisticConcurrencyException: Store update, insert, or delete statement affected an unexpected number of rows (0). Entities may have been modified or deleted since entities were loaded. See http://go.microsoft.com/fwlink/?LinkId=472540 for information on understanding and handling optimistic concurrency exceptions. at System.Data.Entity.Core.Mapping.Update.Internal.UpdateTranslator.ValidateRowsAffected(Int64 rowsAffected, UpdateCommand source) at System.Data.Entity.Core.Mapping.Update.Internal.UpdateTranslator.Update() at System.D...).

The best practice to avoid this error is to add a TransactionScope activity to the workflow. Use the defaults of IsolationLevel = Serializable, and a timeout of 1 minute.

Within that TransactionScope, add a **GetEntity** activity to retrieve the instance of the entity **prior to** the execution of the **SaveEntity** activity. Any property values that need to be updated prior to saving can be done so via Assign statements right after the Get activity and right before the Save activity.

A transaction locks the database to give the workflow a chance to read and update with no other process simultaneously doing the same. Read about the other less aggressive isolation levels as they may be adequate for the purpose based on the type of updates being done and produce less overhead. Google “TransactionScope IsolationLevel Activities”. A “RepeatableRead” may be sufficient.

This pattern will eliminate any chance that another user will update this record in between the execution of the Get and Save activities within the workflow.

Form Data

Avoid Null References in Workflows

To avoid null reference exceptions, ensure your forms accept only valid values. When working with entities, always use a CreateEntity activity (if data is initialized in following form) or use a GetEntity activity if looking up a known item. For example, use a GetEntity activity to retrieve a StudentEntity based on the User Id supplied when a user logs in to a sequence.

Use Form Designer Properties

For all fields and components in the Layout pane, take advantage of the given Form Designer properties to ensure good data. For example, use the Required property and specify values in the supported value ranges.

E-Sign Sequences

Place View Summary Before E-Sign Component

In sequences with e-sign component, insert the View Summary component before the e-sign component so that the end user can review the responses on all forms before the e-sign process is invoked.

Include RecipientStatus Activities

Include GetAdobeSignRecipientStatus, GetDocuSignRecipientStatus, or GetSignNowRecipientStatus activities in your e-sign workflows to handle all status changes (e.g., retry, completed, denied) and to recover from error conditions including connection loss.

Application Initialization

Forms Builder Designer and Renderer can take advantage of Application Initialization which is available on Microsoft Windows Server 2012 and later as a standard part of IIS installation.

This means that when a server comes up, or an application pool is reset, or IIS is reset, the website automatically starts warming up as if a first user had launched a URL on the website. Further, when an application pool automatically recycles, IIS keeps the current process serving files while it warms up a new process. When warm, it redirects requests to the new process and kills the old one, resulting in a seamless recycle of the website.

Typically, a website that is hit by the first user has to start loading all the resources the website needs to serve webpages. This can take a while. On a website with Application Initialization, the loading of resources can happen automatically, and within a few minutes response times will be greatly reduced.

In addition, Designer and Renderer support caching. The caching is done during the warmup. This reduces trips to the database and significantly improves the user experience. Designer caching is on all the time and produces a noticeable performance increase when moving between panels and workspaces within Designer. Renderer caching is set in the Settings workspace. Refer to Enable Renderer Caching.

Warmup is implemented in Anthology Student version 19.0.5 and later. CampusNexus CRM, Staff STS, and Student STS websites are not warmed up. If your workflow uses these sites, it may still take a while for the first user of a sequence to warm up these websites.

If your server has Application Initialization installed, there are two configuration items for IIS to get this to work fully.

1. On each application pool that you are using (Designer and Renderer usually use different ones), go to **Advanced Settings** and set **Startup Mode** to *Always Running*.
2. On each website (Designer and Renderer), go to **Advanced Settings** and set **Preload Enabled** to *true*.

These settings will be configured by Installation Manager during the installation of Forms Builder 3.6 and later.

To test if this is working (non-Azure sites only):

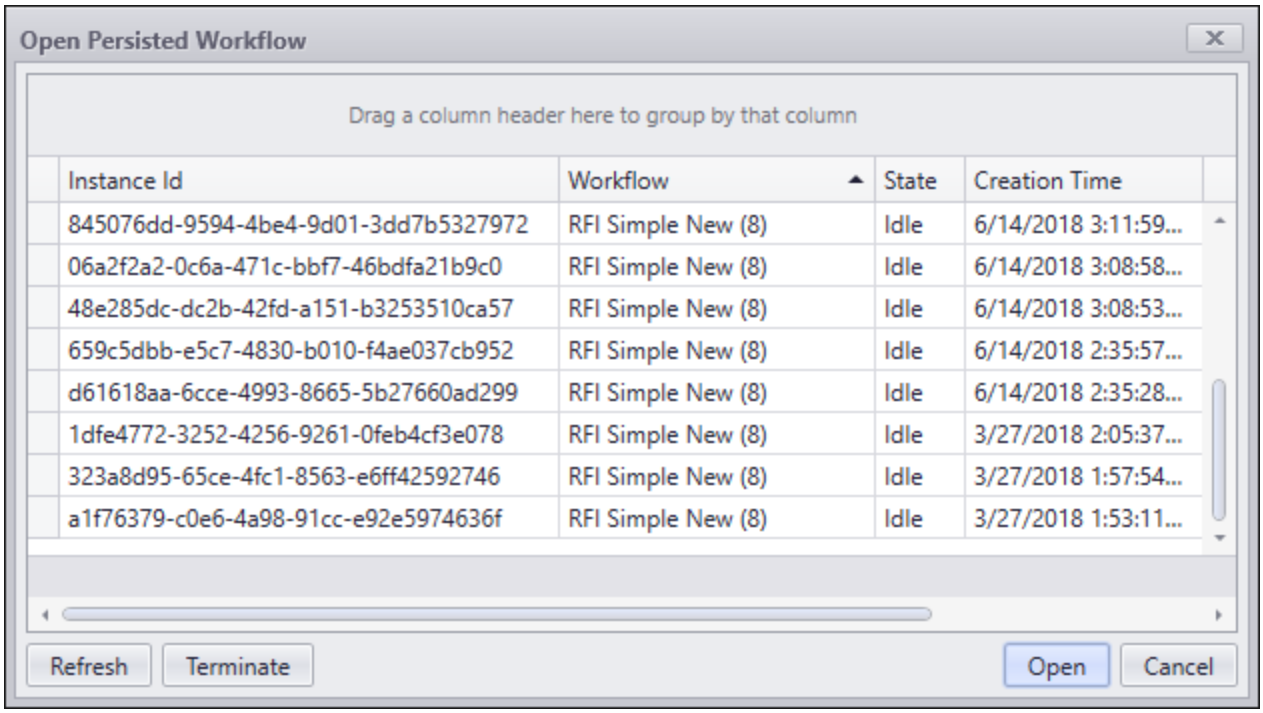
1. In Forms Builder Designer, go to **Settings** and change the NLog Level to **Debug** for both Renderer and Designer.
2. At an administrator command prompt, type **net stop w3svc**. This will stop the IIS process.

3. Delete the Designer and Renderer log files for today in **C:\Logs**.
4. At the administrator command prompt, type **net start w3svc**.
5. Designer and Renderer log files should be automatically recreated. They will contain Debug log lines with the words *Starting Warmup*.
6. Reset your NLog Levels to their original settings and restart IIS.

For details on Application Initialization, refer to https://docs.microsoft.com/en-us/iis/get-started/whats-new-in-iis-8/iis-80-application-initialization.

Persisted Workflow Instances

It is common that students begin to fill out a Request For Information form and then change their mind and exit the sequence before completing. As a result, the durable instance table will accumulate rows for abandoned sequences that never reach an end state. For a workflow administrator it can become challenging to locate a specific workflow instance when browsing the list of persisted workflows in Workflow Composer.



A script example has been provided to remove persisted workflow instances for abandoned sequences from the Durable Instancing table. Run this script periodically as needed.

Persisted workflow instances can be deleted from the Sequence Designer workspace. For more information, refer to Delete Persisted Workflow Instances.

To run the script to remove persisted workflow instances:

1. Open **Microsoft SQL Server Management Studio**.
2. Connect to the server and database indicated in the status bar of your Workflow Composer.

   You must have administrator permissions to the database.
3. Select **New Query**.
4. Copy and paste the following script example into the query window:

   You may need to adjust the maximum number of days in the statement "DECLARE @MAX_AGE_IN_DAYS INT = 20".

   - If your form sequences use DocuSign, the time period should be at least 20 days.
   - If your form sequences do not use DocuSign, reduce the time period as appropriate.

```
-- Delete instances older than the defined age
CREATE PROCEDURE [dbo].[Sproc_DeleteWorkflowInstances_20DAYS]
AS
BEGIN
    DECLARE @MAX_AGE_IN_DAYS INT = 20

    DECLARE @UsefulCursor CURSOR
        ,@SurrogateInstanceId BIGINT SET @UsefulCursor = CURSOR
    FOR
    SELECT i.SurrogateInstanceId
    FROM [System.Activities.DurableInstancing].[InstancePromotedProperties] p
    INNER JOIN [System.Activities.DurableInstancing].[InstancesTable] i ON i.id = p.InstanceId
    WHERE DATEDIFF(DAY, i.LastUpdated, GETUTCDATE()) >= @MAX_AGE_IN_DAYS
    ORDER BY i.LastUpdated DESC

    OPEN @UsefulCursor

    FETCH NEXT
    FROM @UsefulCursor
    INTO @SurrogateInstanceId

    WHILE @@FETCH_STATUS = 0
    BEGIN
        EXEC [System.Activities.DurableInstancing].[DeleteInstance] @SurrogateInstanceId

        FETCH NEXT
        FROM @UsefulCursor
        INTO @SurrogateInstanceId
```

```
        END

        CLOSE @UsefulCursor

        DEALLOCATE @UsefulCursor
    END
```

5. Select the **Execute** button to run the script.

   Expected result: `Commands completed successfully.`

6. Once the script is saved to the database, use the following command to run the script whenever needed:

   ```
   exec Sproc_DeleteWorkflowInstances_20DAYS
   ```

   You can also put the `exec` command into a scheduled SQL Server Agent job, so it runs nightly unattended.

CampusNexus CRM Integrations

Prerequisites

The **Higher Ed** and **Web Client** components of CampusNexus CRM must be installed.

Integrate Forms Builder with CampusNexus CRM 11.1 or Later

1. If you're using CampusNexus CRM 11.1:

   In the Web Client installation folder, in the **NexusCrm.config** file, set the value of the **EdmModelGeneration BuildMode** parameter to **Enabled**, and then restart the Cmc.Crm.Workspaces application pool.

   If you're using CampusNexus CRM 12.0 or later:

   In the Web Client installation folder, in the **web.config** file, set the value of the **EdmModelGeneration BuildMode** parameter to **Enabled**, and then restart the Cmc.Crm.Workspaces application pool.

2. Copy the **Cmc.NexusCrm.Contracts.dll** file from the \bin folder of Web Client to the installation folder of Workflow Composer and Forms Renderer.

   All operational and reference objects are wrapped in this file Cmc.NexusCrm.Contracts.dll. When new properties are created in CampusNexus CRM or an existing property definition (metadata) is changed, this file is regenerated. For example, it is regenerated when creating or updating an object, a tab, a property, or a relationship.

   The regenerated file will need to be copied to the installation folder of **Workflow Composer**and to the \bin folder of **Forms Renderer**.

   > ⚠ **Do not copy the Cmc.NexusCrm.Contracts.dll to the \bin folder of Forms Builder Designer.**

   The .dll file can be copied while you remain logged on to Workflow Composer. Any updates will be reflected in Workflow Composer after you log off and on again.

3. In CampusNexus CRM a maximum of **1024 properties** can be published for use with Forms Builder. If additional properties are needed, **unpublish** previously published properties and then publish new properties. The maximum count of 1024 properties cannot be exceeded.

   For more information about publishing and unpublishing object properties, refer to the description of the sproc_GetPropertiesPublishStatusForObject and sproc_SetPropertiesPublishStatusForObject stored procedures in the CampusNexus CRM Integration guide.

4. If you're using CampusNexus CRM 11.1:

To consume events triggered from Web Client and iServices in Workflow Composer, set the value of the **Workflow Integrated** parameter to “True” in the **NexusCRM.config** file in the Web Client installation folder. By default, its value is “False”.

If you're using CampusNexus CRM 12.0 or later:

To consume events triggered from Web Client and iServices in Workflow Composer, set the value of the **Workflow Integrated** parameter to “true” in the **web.config** file in the Web Client installation folder. By default, its value is “False”.

Integrate Workflow Composer with CampusNexus CRM 11.1 or Later

Anthology Student and CampusNexus CRM can use a single installation of Workflow Composer and Forms Builder to work with both applications.

1. In Workflow Composer, select **Package Manager**, and install the **Activities and Contracts (CRM)** package corresponding to the installed CampusNexus CRM version.
2. Copy the **Cmc.NexusCrm.Contracts.dll** and **Cmc.NexusCrm.WcfProxy.dll** files from the \bin folder of Web Client to the installation folder of Workflow Composer and to the \bin folder of Forms Renderer.

   ⚠ **Do not copy the Cmc.NexusCrm.Contracts.dll to the \bin folder of Forms Builder Designer.**

   The .dll file can be copied while you remain logged on to Workflow Composer. Any updates will be reflected in Workflow Composer after you log off and on again.
3. In the installation path of Workflow Composer, open the **WorkflowComposer.exe.config** file using a text editor and locate to the **<appSettings>** tag.
4. Verify that the value of the **ConfigureCampusNexusWcfProxy** key is “**true**”. Change it to “true” if a different value is set.
5. Add a new key, **CmcNexusCrmWebUrl**, and specify the Web Client URL as its value.

   Updated code in the <appSettings> tag will now be as follows:

   ```
   <appSettings>
   <add key=”ConfigureCampusNexusWcfProxy” value=”true”/>
   <add key=”CmcNexusCrmWebUrl” value=”<Web Client URL>”/>
   </appSettings>
   ```
6. Save and close the WorkflowComposer.exe.config file.

Run an OData Query in the Web Client

System integrators can view the results of a lookup query that is available in the Web Client for CampusNexus CRM. Prior to integrating with CampusNexus CRM, this functionality helps an integrator to verify the list of

values that will be displayed in their query.

View Lookup Query Results

1. Suffix the Web Client URL as follows: **http://<web client url>/nexuscrmodata/$metadata**.

   The webpage that is displayed includes lookup queries that are available by default.

2. Search for the text "**lookup**" and then navigate to the query that you want to run.

   Example

   You want to run the following query to verify the list of available Account types:

   LookupQueryName="EnumAccountAccountTypes?$select=Id,DisplayValue&$filter=IsActive eq 1&$orderby=DisplayOrder"

   a. Copy the following text from the query:

      EnumAccountAccountTypes?$select=Id,DisplayValue&$filter=IsActive eq 1&$orderby-y=DisplayOrder

   b. Append the copied text to the Web Client URL as follows:

      http://<Web Client URL>/nex-uscrmodata/EnumAc-coun-tAccountTypes?$select=Id,DisplayValue&$filter=IsActive%20eq%201&$orderby=DisplayOrder

   c. Press ENTER.

3. The list of values available in the Account Type property is displayed.

Set Up the Database Environment

Forms Builder can be used with the databases of CampusNexus CRM, Anthology Student, or both. In addition, the Workflow Composer along with appropriate packages for contracts and activities is required.

For details about the supported product version combinations, refer to the Product Compatibility Matrix (logon required).

Depending on the database environment, perform the following integration and verification steps.

CampusNexus CRM Environment

1. Use Installation Manager to install **CampusNexus CRM** (including the Web Client).
2. On the machine where the Web Client for CampusNexus CRM is installed:
   a. Navigate to **\inetpub\wwwroot\cmc.crm.workspaces**.
   b. In CampusNexus CRM version 11.1, open the **NexusCRM.config** file.

      In CampusNexus CRM version 12.0 or later, open the **web.config** file.
   c. Find "EdmModelGeneration" and make sure that BuildMode is enabled.

```
<EdmModelGeneration BuildMode="Enabled">
  <!--
Allowed Values for BuildMode
  - "Enabled" - For generating model using the latest meta from database
  - "CompileSourceFile" - For generating model using the Source Files. Used only for troubleshooting.
  - "Disabled" - For disabling model generation
"Faulted" and "CompileSucceeded" values are for internal usage
-->
</EdmModelGeneration>
```

3. Use Installation Manager to install **Workflow Composer**.
4. Open Workflow Composer, select **Package Manager**, and verify that the activities and contracts for your product versions are installed.

   For example, if you are using Forms Builder 3.5 with CampusNexus CRM 12.1, install the following packages:

   - Forms Builder Contracts 3.5.0 (3.5.0.xxx)
   - Activities and Contracts (CRM) 12.1.0 (12.1.0.xxx)

   Remove the packages for older versions when you install new versions.
5. Log in to the **Web Client** for CampusNexus CRM.
6. Open File Explorer, navigate to **\inetpub\wwwroot\Cmc.Crm.Workspaces\bin**, and copy the **Cmc.NexusCrm.Contracts.dll** file.
7. Paste the **Cmc.NexusCrm.Contracts.dll** file into the following locations:

- On the machine that hosts Forms Builder: **\inetpub\wwwroot\CMCFormsRenderer_V3\bin**
- On the machine where Workflow Composer is installed: **\Program Files (x86)\CMC\Workflow**

Every time you build new custom fields/entities in CampusNexus CRM, you need to copy the Cmc.NexusCrm.Contracts.dll to these locations.

> ⚠ **Do not copy the Cmc.NexusCrm.Contracts.dll to the \bin folder of Forms Builder Designer.**

The .dll file can be copied while you remain logged on to Workflow Composer. Any updates will be reflected in Workflow Composer after you log off and on again.

Verify the Setup

After Forms Builder has been installed:

1. Log in to Forms Builder.
2. In Form Designer, create a form that collects data for a **Contact**.
3. In Sequence Designer, create and save the sequence.
4. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
5. In Workflow Composer, add a **CreateEntity<ContactEntity>** activity to the Entry of the first form and a **SaveEntity<ContactEntity>** activity in the final transition.
6. Publish the updated workflow definition.
7. In the Sequence List, open and fill out the rendered form.
8. In the Desktop for CampusNexus CRM, verify that the new Contact is created.

In this environment, workflow definitions for sequences are saved in the database of CampusNexus CRM.

For more information, refer to CampusNexus CRM Integrations.

Anthology Student Environment

1. Use Installation Manager to install **Anthology Student** (including the Web Client).
2. Use Installation Manager to install **Workflow Composer**.
3. Open Workflow Composer, select **Package Manager**, and verify that the activities and contracts for your product versions are installed.

   For example, if you are using Forms Builder 3.5 with Anthology Student 19.0.4, install the following packages:

   - Forms Builder Contracts 3.5.0 (3.5.0.xxx)
   - Activities and Contracts (V1) 19.0.4 (19.0.4.xxx)

- Activities and Contracts (V2) 19.0.4 (19.0.4.xxx)

Remove the packages for older other versions when you install new versions.

Verify the Setup

After Forms Builder has been installed:

1. Log in to Forms Builder.
2. In Form Designer, create a form that collects data for a **Student**.
3. In Sequence Designer, create and save the sequence.
4. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
5. In Workflow Composer, add a **CreateEntity<StudentEntity>** activity to the Entry of the first form and a **SaveEntity<StudentEntity>** activity in the final transition.
6. Publish the updated workflow definition.
7. In the Sequence List, open and fill out the rendered form.
8. In the Web Client for Anthology Student, verify that the new Student is created (or check the syStudent table in the database).

In this environment, workflow definitions for sequences are saved in the database of Anthology Student.

CampusNexus CRM and Anthology Student Environment

If you are using both CampusNexus CRM and Anthology Student, perform all the steps above.

In this environment, workflow definitions for sequences are saved only in the database of Anthology Student.

Update Forms Builder URLs (HTTPS or HTTP)

You can host Form Designer, Forms Renderer, and STS on public URLs or on port 443 with HTTPS. Use the following steps to remap existing working Forms Builder websites to HTTPS and hosted on port 443. The steps remain same even when Form Designer and Forms Renderer are hosted on public HTTP URLs.

1. Determine the public URLs for Form Designer, Forms Renderer, and STS. For purpose of this example, we will assume URLs as follows:

Form Designer:	`https://design.anthology.com`
Forms Renderer:	`https://apply.anthology.com`
STS:	`https://signin.anthology.com`

2. Edit the binding information of Form Designer, Forms Renderer, and STS in IIS. (Skip this step if you are not using SSL.)

   In IIS, right-click on the site for **FormsBuilderDesigner**, and select **Edit Bindings**.

   a. Select the **Add** button.

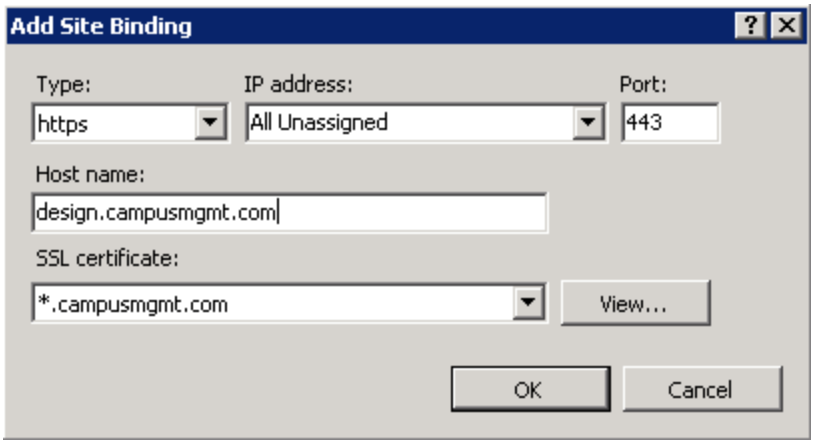
   b. Select Type **https** and specify Port **443**.

   c. The IP address is optional.

   d. Specify the **Host name**.

   e. Select an appropriate **SSL certificate**.

   The following image shows the Site Binding for Form Designer.



   Remove the previous non-HTTPS binding information after the above change.

   Repeat this step for **Forms Renderer** and **STS**.

3. Modify the following settings in the `<system.identityModel>` section in the web.config files of Form Designer and Forms Renderer:

   Update the following properties in the **web.config** file of **FormsBuilderDesigner**:

   - Audience URI:

     <audienceUris>

     <add value="https://design.anthology.com" />

     </audienceUris>

   - TrustedIssuers:

     name="https://signin.anthology.com/"

   - Copy the latest thumbprint value from the certificate used for STS and update it in the thumbprint section.

   Repeat this step for **Forms Renderer**.

4. Modify the `<system.identityModel.services>` section in the web.config files of Form Designer and Forms Renderer.

   Update following properties in the **web.config** file of **Form Designer**:

   - federationConfiguration/wsFederation:

     issuer="https://signin.anthology.com/"

     realm="https://design.anthology.com/"

     reply="https://design.anthology.com/"

   Repeat this step for **Forms Renderer**.

Upgrade Considerations

To prevent any customizations from being lost when Forms Builder is upgraded, ensure that you take the following steps.

Save Default Forms

The following forms are intended as templates should **not** be modified and used in a sequence:

- *Default-Confirmation*
- *Default-Frame*
- *Default-DocuSignWait*

> ⚠ These forms will be overwritten with the latest changes during an upgrade. If you have modified them directly, you will lose your changes.

Make a copy of each form by doing a **Save As** in Form Designer, modify the new form, and then use it in a sequence.

Applying Styles Globally

As the functionality of the /Content/Custom/CustomerIncludes.html file is not available after version 4.0, on upgrading to a new version the following changes must be done to retain CSS references and images that were previously included in the CustomerIncludes.html file:

1. Merge all the CSS files that were referenced in the CustomerIncludes.html file into a single CSS file.
2. Open the new CSS file and change references to all images as indicated below. If the CSS contains a `.navbar brand` class but must include the `!important` tag:

   If the css previously referenced an image in a file path...

   ```
   .navbar-brand {
   background: url('images/YourLogo.png') center / contain no-repeat !important;
   padding: 0;
   width: 200px;
   margin-left: 5px;
   }
   ```

   Replace the path with getCustomContentDataAsFile?src=

   ```
   .navbar-brand {
   background: url(getCustomContentDataAsFile?src=Yourlogo.png) center / contain no-repeat !important;
   padding: 0;
   width: 200px;
   margin-left: 5px;
   }
   ```

   The **YourLogo.png** image file is included as an example in the above code snippets.

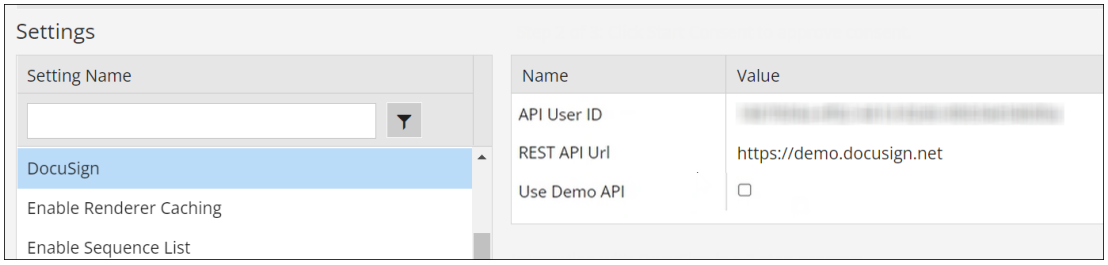
3. Select the Custom Content tile on the home page to upload the CSS file to the CustomContent table in the database. In the Custom Content tile, select the check box in the Global column. The uploaded CSS file will apply globally to all sequences.

   Only a single CSS file can be selected in the Global column. To change the CSS that will apply globally, deselect the check box of the current file and then select the check box of another CSS file.

4. Upload images that were referenced in previous CSS files to the CustomContent table in the database.
5. Save all your changes.

Upgrades to DocuSign Settings

The OAuth 2.0 framework is used to integrate with DocuSign. The DocuSign Settings page includes the following fields:



To integrate with DocuSign:

- When upgrading to version 4.1, to continue working with DocuSign, values will need to be updated in existing and new DocuSign setting fields to implement authorization in the OAuth 2.0 framework.
- When upgrading to version 4.1 the value of the REST API Url field can be copied from version 4.0.
- In a new implementation, values in all fields will need to be specified.

  In either scenario, refer to information in DocuSign Settings.

Upgrades to Multiple Renderer URLs

1. Before upgrading, in sequences for which multiple renderer URLs were configured in previous versions, back up the associated style sheets and logos and also make a note of their associated URLs. This step must not be skipped because after upgrading, it will require considerable effort to understand which CSS files, logos, and sequence URLs were previously associated with each other.
2. In each CSS file, change code as follows if the CSS contains a `.navbar brand` class but must include the `!important` tag:

   If your css previously referenced an image in a file path...

```
.navbar-brand {
background: url('images/YourLogo.png') center / contain no-repeat !important;
padding: 0;
width: 200px;
```

```
margin-left: 5px;
}
```

... replace the path in the css with getCustomContentDataAsFile?src=

```
.navbar-brand {
background: url(getCustomContentDataAsFile?src=Yourlogo.png) center / contain no-repeat !im-
portant;
padding: 0;
width: 200px;
margin-left: 5px;
}
```

The **YourLogo.png** image file is included as an example in the above code snippet.

3. Upload the CSS and image files using the Custom Content feature.
4. Select the **Apply Themes Dynamically** option in the Sequence Designer.
5. In Workflow Composer, change the name of the variable in the Switch step from its old value (cssFileName) to **dynamicCSSFileName**. Refer to step 7 of the procedure **Select Style Sheets Using Workflow Activities** in Multiple Renderer URLs.

Designer

Forms Builder Designer provides the capability to design forms and sequences which will then be rendered as webpages via Renderer.

Designer is installed on your web server under **wwwroot\CMCFormsBuilderDesigner_V3**.

`http://<server>.<domain>:<9002>/`

Designer is installed on port 9002 by default. The port number can be customized during installation.

It can be installed on port 443 with https:

`https://<server>.<domain>:<443>/`

Access your Forms Builder URL with the applicable port number to view the Designer home page.

Custom Content

The Custom Content tile on the Home page enables users to upload files to the CustomContent table in the database. These files can then be used to customize forms. In previous releases, on premises customers could also use the file system reference method, and for Azure customers the database method was the only way of customizing forms.

As the file system reference method is no longer available from version 4.1, on premises and Azure customers must only use the database method as it simplifies Forms Builder upgrades.

Custom files can include 5 different types of content: images, videos, CSS styles, JavaScript, and logos.

1. Image Files

   Currently, the following extensions are allowed: .jpg, .jpeg, .bmp, .svg, .gif, .ico, .png, and .apng.
   Please check https://en.wikipedia.org/wiki/Comparison_of_web_browsers for details on support in various browsers.

2. Video Files

   Only mp4 is supported.

3. CSS Files (Style Sheets)

   This section describes how to apply a CSS file to forms in a sequence, to a sequence, or globally across all sequences.

   In addition, to understand how CSS files can be configured in form sequences to be displayed at different URLs with different branding using a single instance of Forms Builder, refer to Multiple Renderer URLs.

4. JavaScript

   Since .js files are active scripts, you must ensure that security guidelines are followed. Several libraries that are part of Renderer that can be utilized, including JQuery, w3 (from w3schools.com), Lodash, he, jsep, kendo, and clipboard that will work outside of the Anthology Inc. Angular application.

5. Navigation Bar Logos and Sequence Logos (not to be confused with header or footer content for a form)

To upload custom content, select **Select custom content files** and select the files (or drag the files into the selection area). The files will be uploaded to the database used by Forms Builder and added to the grid showing the currently uploaded files.

To remove a custom content file, select it in the grid, select **Delete**, and select **OK** on the confirmation message. The file is deleted. If the file is in use, it will not be deleted, and a validation message is displayed in Form Designer.

If you edited a previously uploaded file and want to upload the new version of the same file, the previous version will be overwritten, i.e., you don't need to delete the previous version.

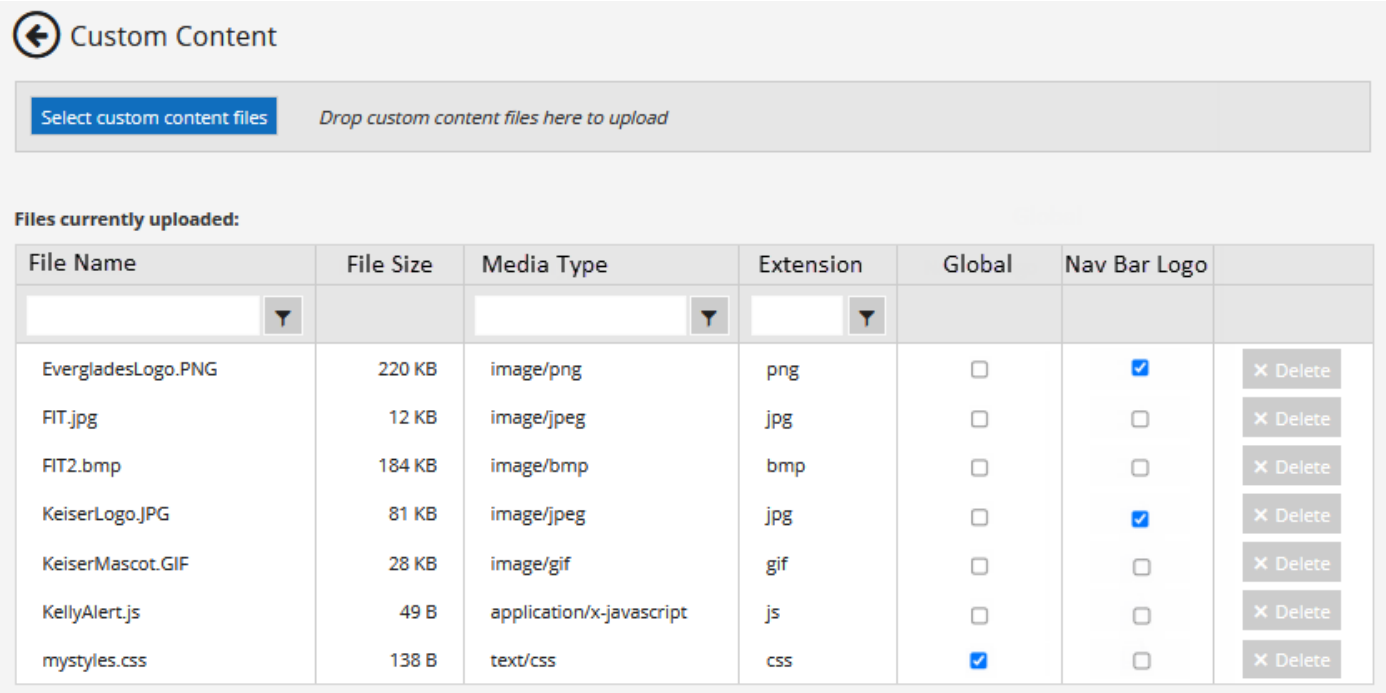


Image Files

While image files can be directly embedded in HTML by going to https://codebeautify.org/image-to-base64-converter and creating the data to embed, the Custom Content feature does the work for you. Anywhere you need an image, simply upload an image file and add an HTML component to the page. Currently, the following extensions are allowed: .jpg, .jpeg, .bmp, .svg, .gif, .ico, .png, and .apng.

Use the following HTML to refer to the uploaded file:

```
<cmc-custom-content-image src="myImage.png" model="myModel" />
```

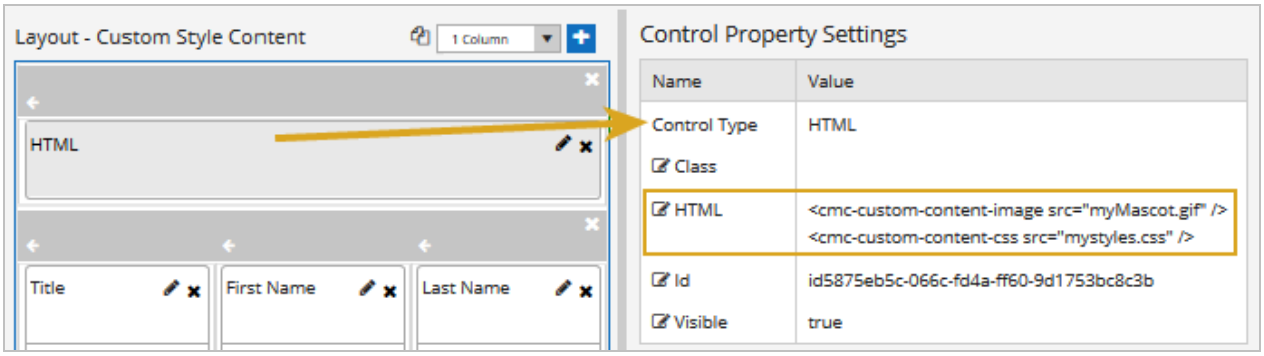
This will place the image on the page at the location of the component. If you ever need to change the image, simply upload a new file with the same name. You could also add style information for the image using CSS.

Model is an optional attribute. If it is specified and an argument in a workflow is assigned a value, it will be used to retrieve the image uploaded by name. If not, the src value is used. This allows you to programmatically decide which image to display.

> In the model value, specify either the argument name only (e.g., "myModel") or the argument name with the "vm.models." prefix (e.g., "vm.models.myModel").

The src value acts as the default if model has not been set or modified in workflow, otherwise it is ignored.

Example



You can combine multiple directives in one HTML component as shown in the example with "cmc-custom-content-image" and "cmc-custom-content-css".

Video Files

This directive has attribute "src1" and optional attributes "src2" and "src3", which refer to files in the CustomContent database table. It gets the FileData and replaces this element with an embedded video element that utilizes base 64 strings.

Use the following HTML to refer to the uploaded file:

```
<cmc-custom-content-video src1="myVideo.mp4" />
```

Optional attributes are: model="" src2="", src3="", controls, autoplay, width="", and height="".

- **model** allows a workflow argument to set the video for src1 and overrides src1 if set.
- **src2** and **src3** are optional videos to play if **src1** will not play, but usually are not necessary if mp4 is the first video.
- **autoplay** starts the video automatically.
- **controls** will display controls to stop, start, pause the video. None by default.
- **width** and **height** should be set in pixels unless the default size is correct. Usually, only one of width or height is set, not both, otherwise the video might stretch or squash.

Only mp4 is supported by HTML5 in all browsers. All other formats should be converted to mp4 to use this directive.

For other types of videos, other techniques such as embed and object are needed and the video must be a physical file (which defeats the purpose of this directive). It can be hosted on a specialized server or YouTube but may be subject to restrictions for commercial use of the site (such as payment for ad free content).

CSS Files (Style Sheets)

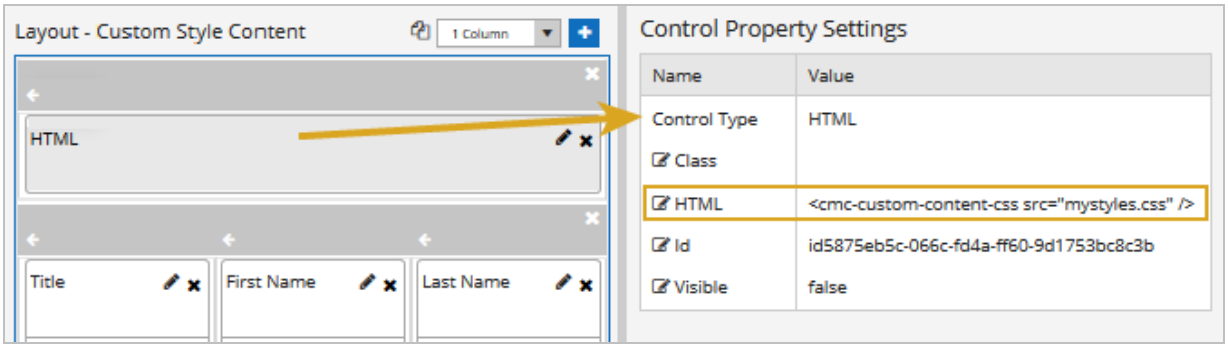
Applying a CSS to Forms in a Sequence

Upload your style sheet. It does not have to be embedded in <style></style> tags, but it can be.

Use the following directive in your HTML component. The HTML component can be made Visible=false since it does not need to be displayed on the page.

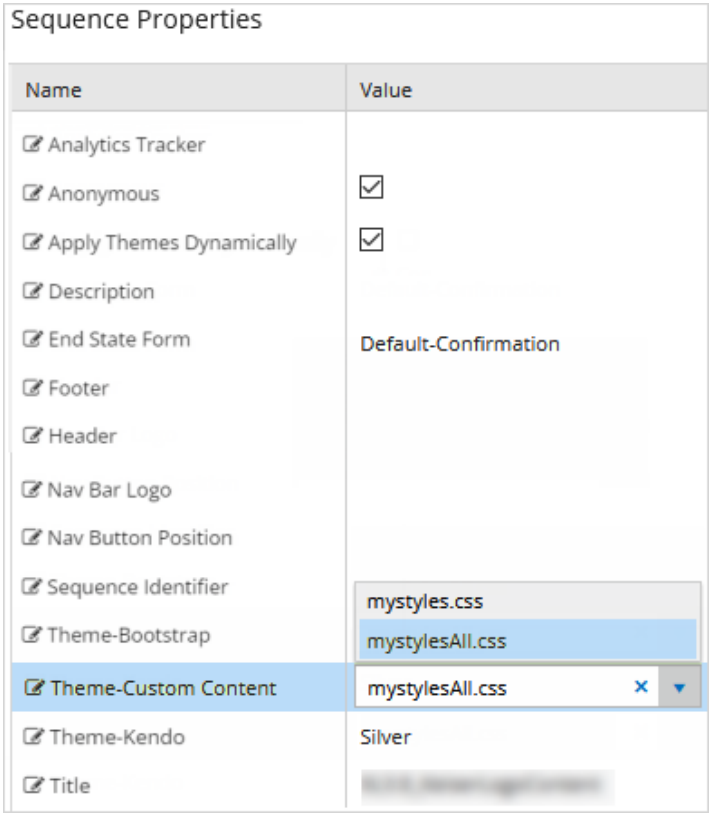
```
<cmc-custom-content-css src="myCss.css" />
```

Example



Applying a CSS to a Sequence

In Forms Builder 3.8 and later, you can upload custom css files to the database and then select a custom css file in the Theme-Custom Content drop-down list in the Sequence Properties. The selected css file will be applied to the entire sequence.



If the css file for a sequence contains references to images, the images need to be uploaded to the database and you must replace the file paths for the images using the **getCustomContentDataAsFile?src=** attribute so that the images are retrieved from the database instead of the file system.

The custom css file may also contain a `.navbar brand` class but must include the `!important` tag. However, the preferred method for the navigation bar logo is the Nav Bar Logo property in the Sequence Properties pane. Refer to Navigation Bar Logos.

Example

If your css previously referenced an image in a file path...

```
.navbar-brand {
background: url('images/YourLogo.png') center / contain no-repeat !important;
padding: 0;
width: 200px;
margin-left: 5px;
}
```

... replace the path in the css with getCustomContentDataAsFile?src=

```
.navbar-brand {
background: url(getCustomContentDataAsFile?src=Yourlogo.png) center / contain no-repeat !important;
padding: 0;
width: 200px;
margin-left: 5px;
}
```

The **YourLogo.png** image file is included as an example in the above code snippet.

Applying a Global CSS

When upgrading Forms Builder, the following changes must be done to retain CSS files and images that were referenced in the CustomerIncludes.html file in previous versions:

1. Merge all the CSS files that were referenced in the CustomerIncludes.html file into a single CSS file.
2. Open the new CSS file and change references to all images as indicated below. If the CSS contains a `.navbar brand` class but must include the `!important` tag:

   If the css previously referenced an image in a file path...

   ```
   .navbar-brand {
   background: url('images/YourLogo.png') center / contain no-repeat !important;
   padding: 0;
   width: 200px;
   margin-left: 5px;
   }
   ```

   ... replace the path with getCustomContentDataAsFile?src=

   ```
   .navbar-brand {
   background: url(getCustomContentDataAsFile?src=Yourlogo.png) center / contain no-repeat !im-
   portant;
   padding: 0;
   ```

```
width: 200px;
margin-left: 5px;
}
```

The **YourLogo.png** image file is included as an example in the above code snippets.

3. Select the Custom Content tile on the home page to upload the CSS file to the CustomContent table in the database. In the Custom Content tile, select the check box in the Global column. The uploaded CSS file will apply globally to all sequences.

   Only a single CSS file can be selected in the Global column. To change the CSS that will apply globally, deselect the check box of the current file and then select the check box of another CSS file.

4. Upload images that were referenced in previous CSS files to the CustomContent table in the database.

5. Save all your changes.

A CSS applied to a form in a sequence will override the CSS applied to the sequence. Similarly, CSS styles applied to a sequence will override a globally applied CSS.

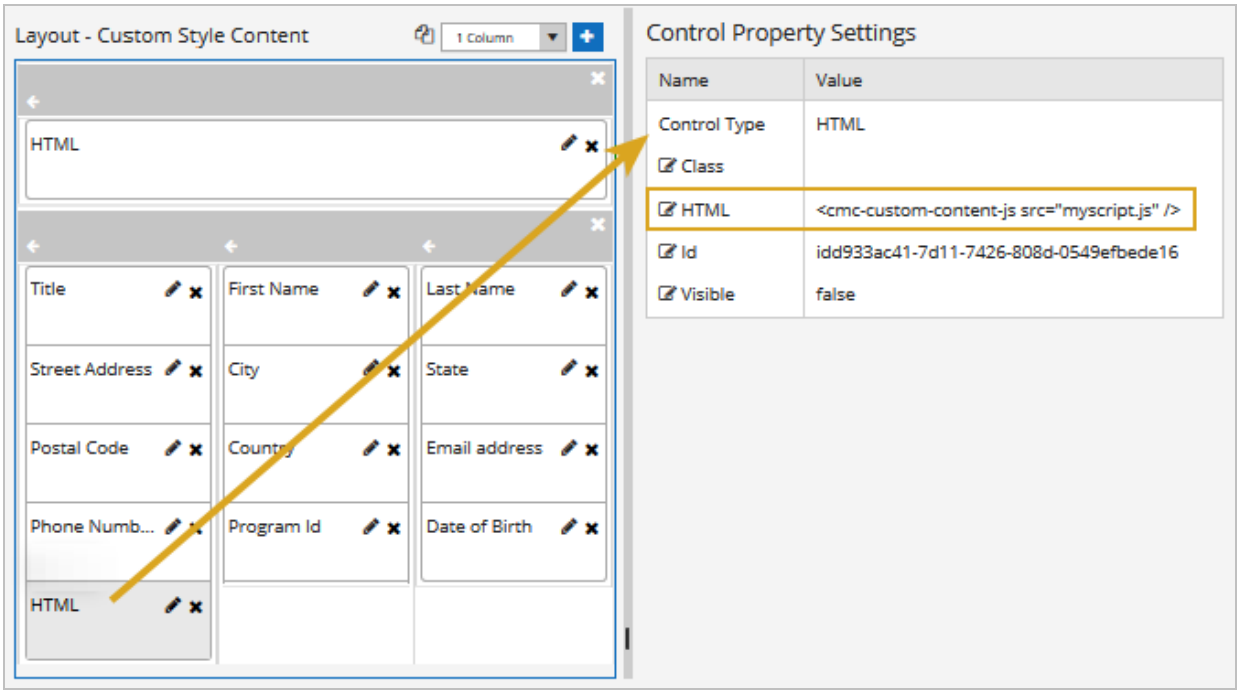
JavaScript

Upload your JavaScript file. The script does not have to be embedded in <script></script> tags, but it can be.

Use the following directive in your HTML component. The HTML component can be made Visible=false since it does not need to be displayed on the page.

```
<cmc-custom-content-js src="myJavascript.js" />
```

Example



Navigation Bar Logos

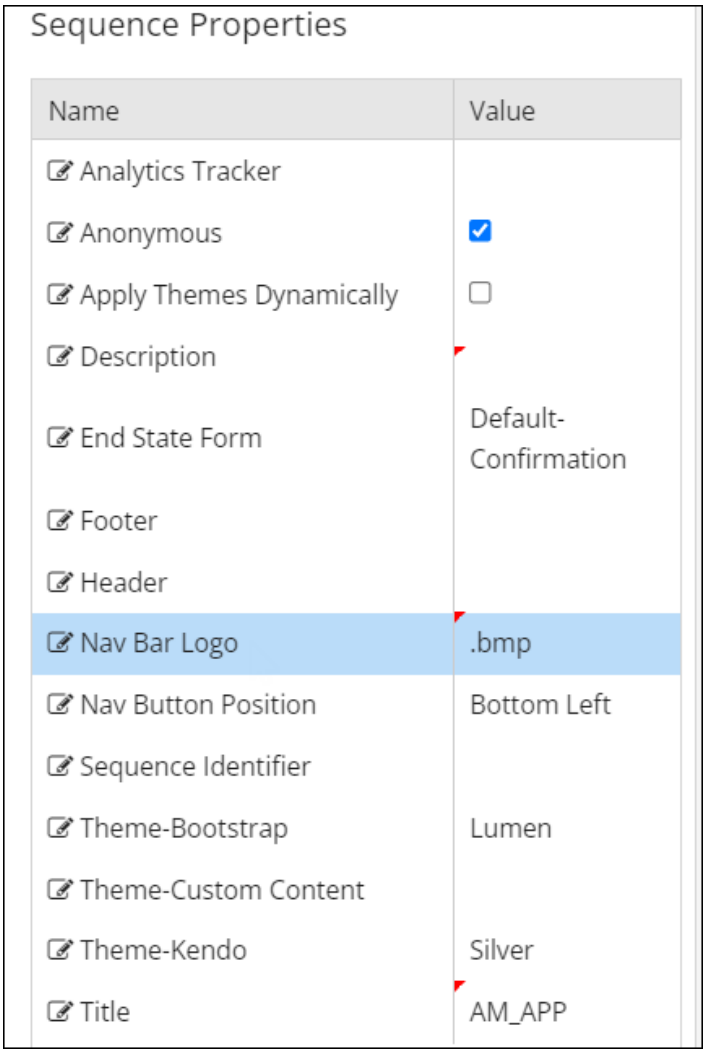
By default, the navigation bar of Forms Builder sequences displays the Anthology logo. In Forms Builder 3.7 and later, you can replace the default logo with a specific logo for your institution using the following steps:

1. **Upload** your logo image file (.png, .bmp, etc.) into the Custom Content workspace.
2. In the **Nav Bar Logo** column, select the check box to indicate that the uploaded logo image can be selected in Sequence Designer and placed in the navigation bar on rendered form sequences.

   The `FormsBuilder.CustomContent` table will show a value of `1` in the `IsLogo` column.

   > When you try to delete a Nav Bar Logo image and the file is in use, it will not be deleted, and a validation message is displayed in Form Designer.

3. In Sequence Designer, open your sequence and select your logo image from the **Nav Bar Logo** drop-down list in the **Sequence Properties**.



The `FormsBuilder.Sequence` table will show the name of the selected image in the `NavBarLogo` column.

4. **Save** your sequence and review it in Forms Renderer.

Notes

- When you are designing new sequences, we recommend using the Navigation Bar Logos method described above instead of using the Sequence Logos method described below. The Nav Bar Logo method will display the selected custom logo without loading the default logo while rendering the form.
- If you accidentally used both methods in the same sequence, the Sequence Logo method will take precedence to ensure backward compatibility. However, a known issue of the Sequence Logo method is that the default logo will briefly appear on the navigation bar before it is replaced with the custom logo. If possible, update your sequences using only the Nav Bar Logo method.
- If you created multiple sequences using different logo methods and you are reviewing these sequences within the same browser session, you will notice rendering issues when going from sequences with the Nav Bar Logo method back to the Sequence Logo method. The rendering issues will not occur in new/refreshed browser sessions for each sequence.

Sequence Logos

The method described below does not prevent the default logo from appearing briefly on the rendered form before it is replaced with a custom logo. If you want to avoid this, use the Navigation Bar Logos method.

This feature has flexibility in that the image can be set programmatically in a workflow. Simply create your logic in the workflow (for instance by campus name) that defines which upload file to use by name and assign that to a named string argument (the model value).

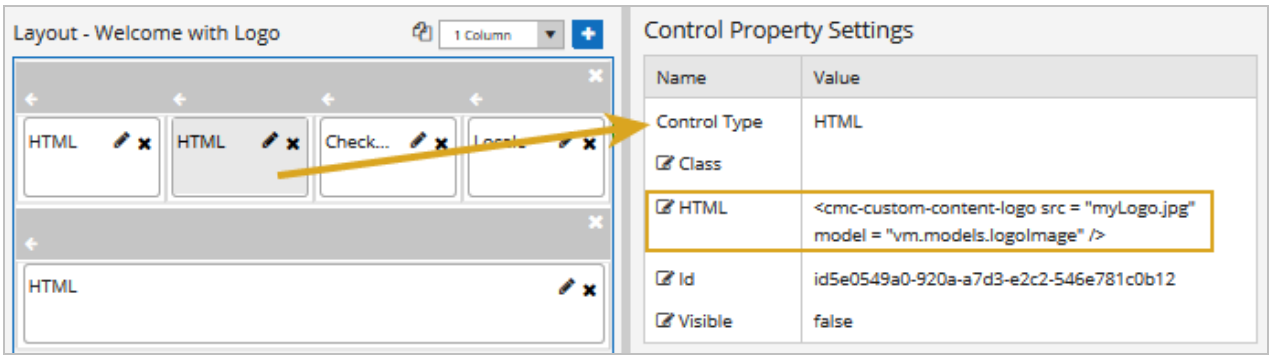
Use the following directive in your HTML component. The HTML component can be made Visible=false since it does not need to be displayed on the page.

```
<cmc-custom-content-logo src="campusA_Logo.png" model="campusSelection" />
```

The src value acts as the default if the model has not been set or modified in the workflow, otherwise it is ignored.

In the model value, specify either the argument name only (e.g., "myModel") or the argument name with the "vm.models." prefix (e.g., "vm.models.myModel").

Example



Remove Local Custom Content

As the custom content feature is no longer available, administrators upgrading to this version should have completed the activity of storing custom files (images, videos, styles and scripts) in the database. If this task is yet to be performed, before upgrading to this version, perform the steps described in any previous version of this topic.

Export/Import

The Export/Import tile on the Forms Builder home page links to a workspace that allows you to export and import sequences and associated workflows from one environment to another. For example, you may have created and tested sequences in a staging environment. You then export the sequences from the staging environment to a network location and later import them into the production environment.

If sequences have been translated in the staging environment, the .po files for the localized sequences need to be transferred to the production environment using the Import Translation tab in the Internationalization workspace of Form Designer. For more details refer to Internationalization.

Note: Custom Bootstrap and Kendo themes applied on sequences imported into Forms Builder in a previous version will be overwritten with the following default settings in version 4.1:

- Theme-Bootstrap – Lumena
- Theme-Kendo - Silver

In addition, if custom themes were used, after uploading the custom theme's .css and images to the database, update the affected sequences by selecting the same .css file in the Theme-Custom Content property of Sequence Designer. For more information, refer to Remove Local Custom Content (version 3.8).

Prerequisites

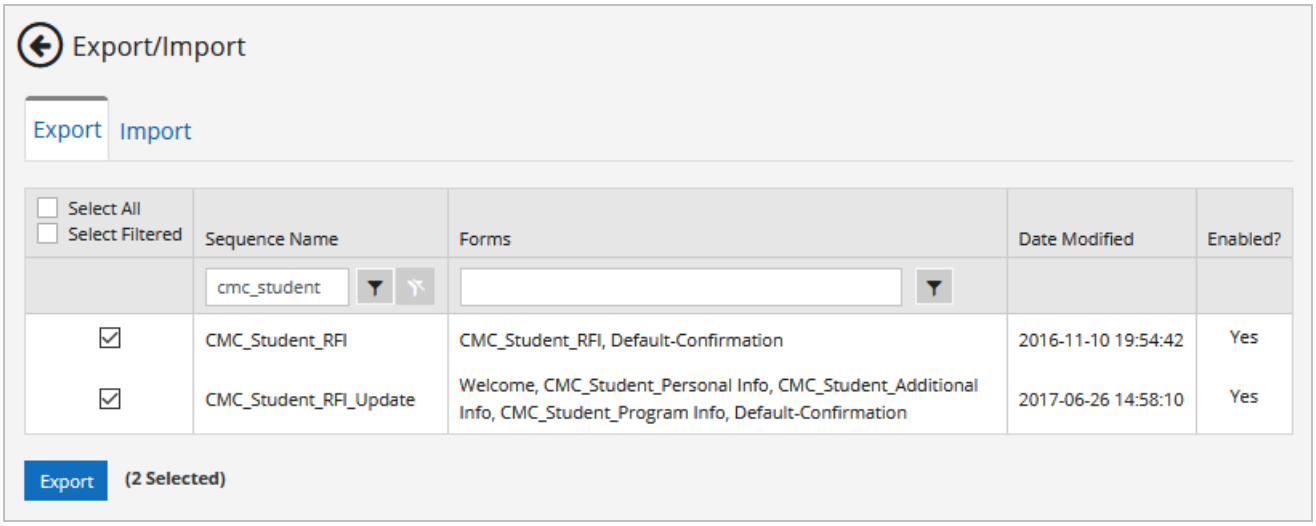
The application databases in the source and target systems must be identically configured regarding School Defined Fields (SDFs) in Anthology Student and Custom properties/objects in CampusNexus CRM. If import sequences with SDFs or Custom fields are defined only in the export environment, an error will be displayed when the sequence is rendered in the new environment and a user attempts to enter a value for an undefined SDF or Custom field.

The Forms Builder environment must be identically configured regarding the integration of CampusNexus CRM and/or Anthology Student. The export/import process does not check for mismatched environment entities. For example, the export/import process will not detect if a sequence from a CampusNexus CRM environment is imported into an Anthology Student environment. The sequence will fail when rendered.

Export Sequences

1. On the Forms Builder home page, select the **Export/Import** tile.

   The Export tab is displayed by default. It contains a grid listing sequence names, forms contained with the sequences, the date modified, and the workflow status.

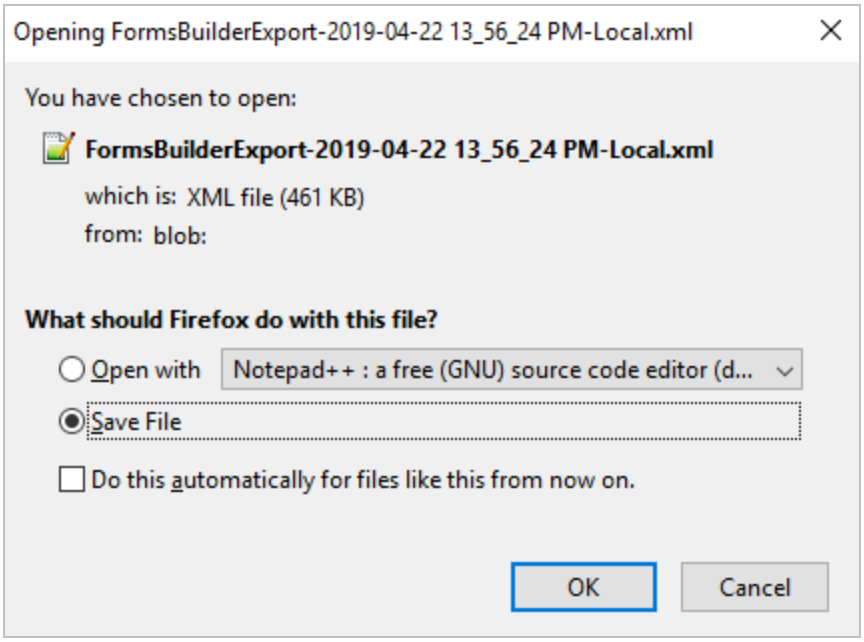


2. Use one of the following options to select sequences. The number of selected sequences will be displayed next to the Export button.

   - Select **Select All**.
   - Use the column filters to narrow your search and select **Select Filtered**.
   - Clear "Select All" and "Select Filtered" and select **individual sequences**.

   Check the **Enabled** column to determine if the workflow for a sequence is enabled. Only sequences that have enabled workflows can be exported.

3. Select **Export**. When export is complete, your browser displays a message with the name of the export file.



4. Select **Save File** and select **OK** to save the export file to the **\Downloads** folder on your local machine. You can copy the file to a new name or location if desired; just note that it must remain an **.xml** type file.

Forms Builder creates an export file in XML format and makes the file available as a stream on the server. The export file is saved with the specified name in the specified location. The export file contains the Forms Builder version and the export date.

If you modify the exported Forms Builder file and then attempt to import it again, the file will not be uploaded.

For every sequence that is exported, data from the following tables is saved:

- Formsbuilder.Sequence
- Formsbuilder.Form
- WorkflowDefinition
- WorkflowDefinitionVersion

The data from the above tables include Name, Title, Description, JSON, XAML and so on. The Ids, DateCreated/Modified are not in the export file and will be generated on import. The exported sequences carry with them any saved Forms Builder elements and configurations required by the sequence (e.g., fields, components, properties, and workflow definitions). End state form data for each sequence is included in the export file.

Import Sequences

1. On the Forms Builder home page, select the **Export/Import** tile.
2. Select the **Import** tab, select **Select exported file**, and navigate to the location where the export file was saved.

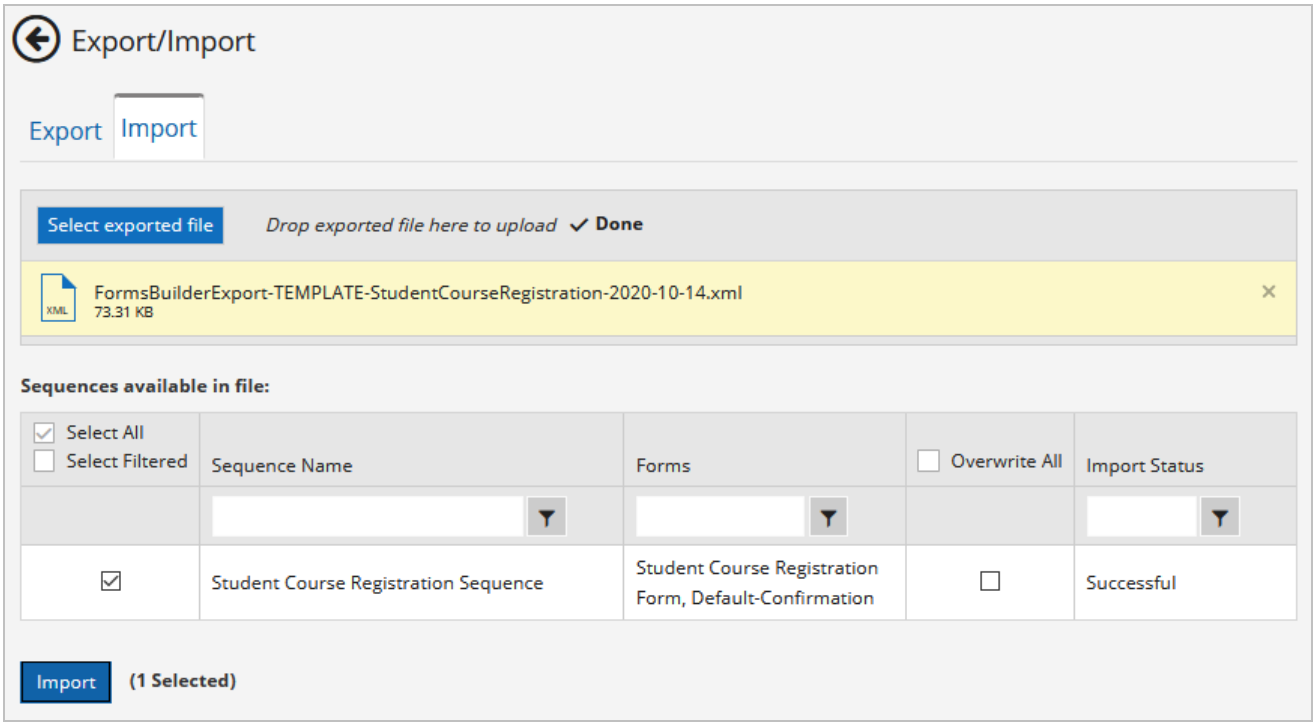
   After selecting the export file, the sequences that are available for import are listed in the grid.
3. Use one of the following options to select sequences. The number of selected sequences will be displayed next to the Import button.
   - Select **Select All**.
   - Use the column filters to narrow your search and select **Select Filtered**.
   - Clear "Select All" and "Select Filtered" and select **individual sequences**.
4. If you want to force an overwrite of sequences and workflows with the same name, select the check box in the header of the **Overwrite All** column OR select this option for individual sequences.

   If the check box is not selected, sequences with no name conflicts will still be imported but those with duplicate or conflicting data will not be.

   The *Default-Confirmation*, *Default-Frame*, and *Default-DocuSignWait* forms are handled as special cases during the import. These forms will remain unchanged in the new environment and will not be flagged as duplicates. This allows for form reuse in multiple sequences and import of sequences with default forms without forcing an overwrite. If, however, the Overwrite option is selected, the forms will be updated.
5. Select **Import**. The import process deserializes the XML of the export file into the appropriate entities. If there are any issues with the export data causing errors with deserialization, an error is displayed and logged.

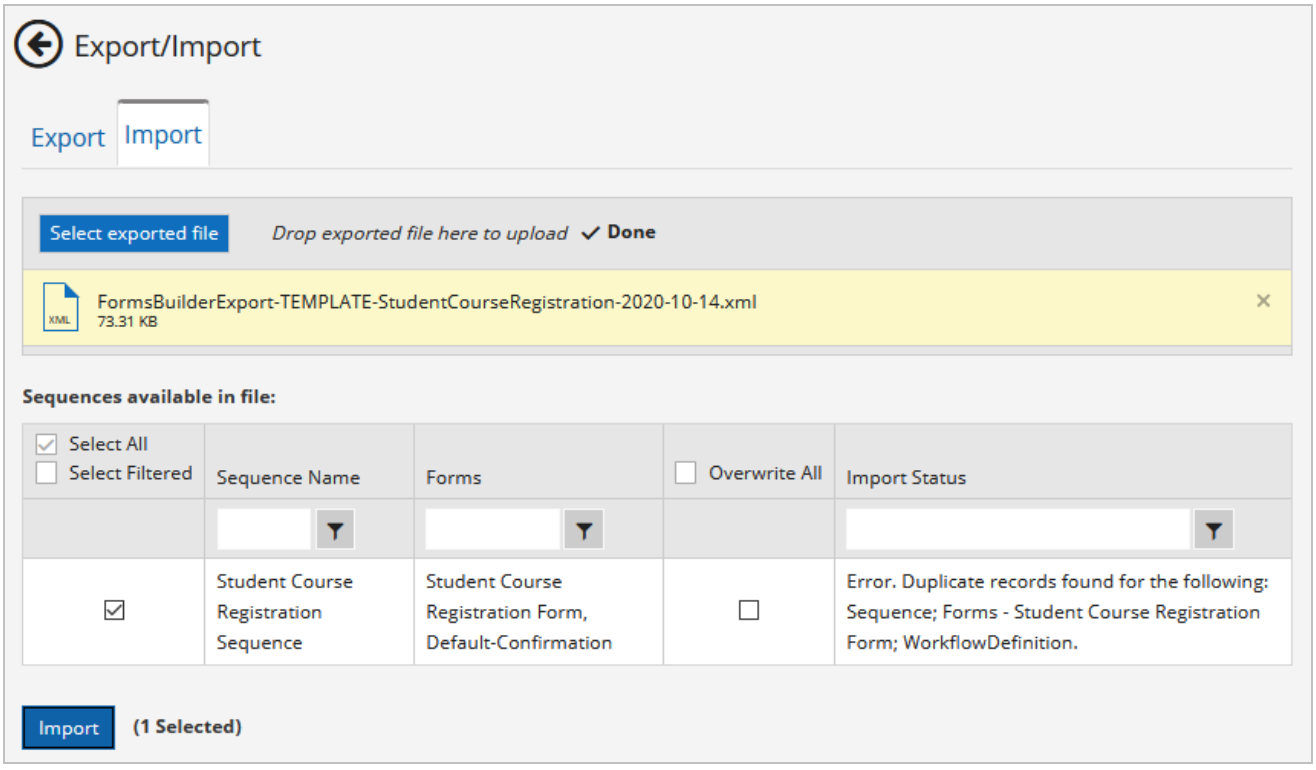
If a theme is missing in the new environment, the imported sequence is rendered using the default styles.

The Import Status column displays the import result, for example: *Successful*



— OR —

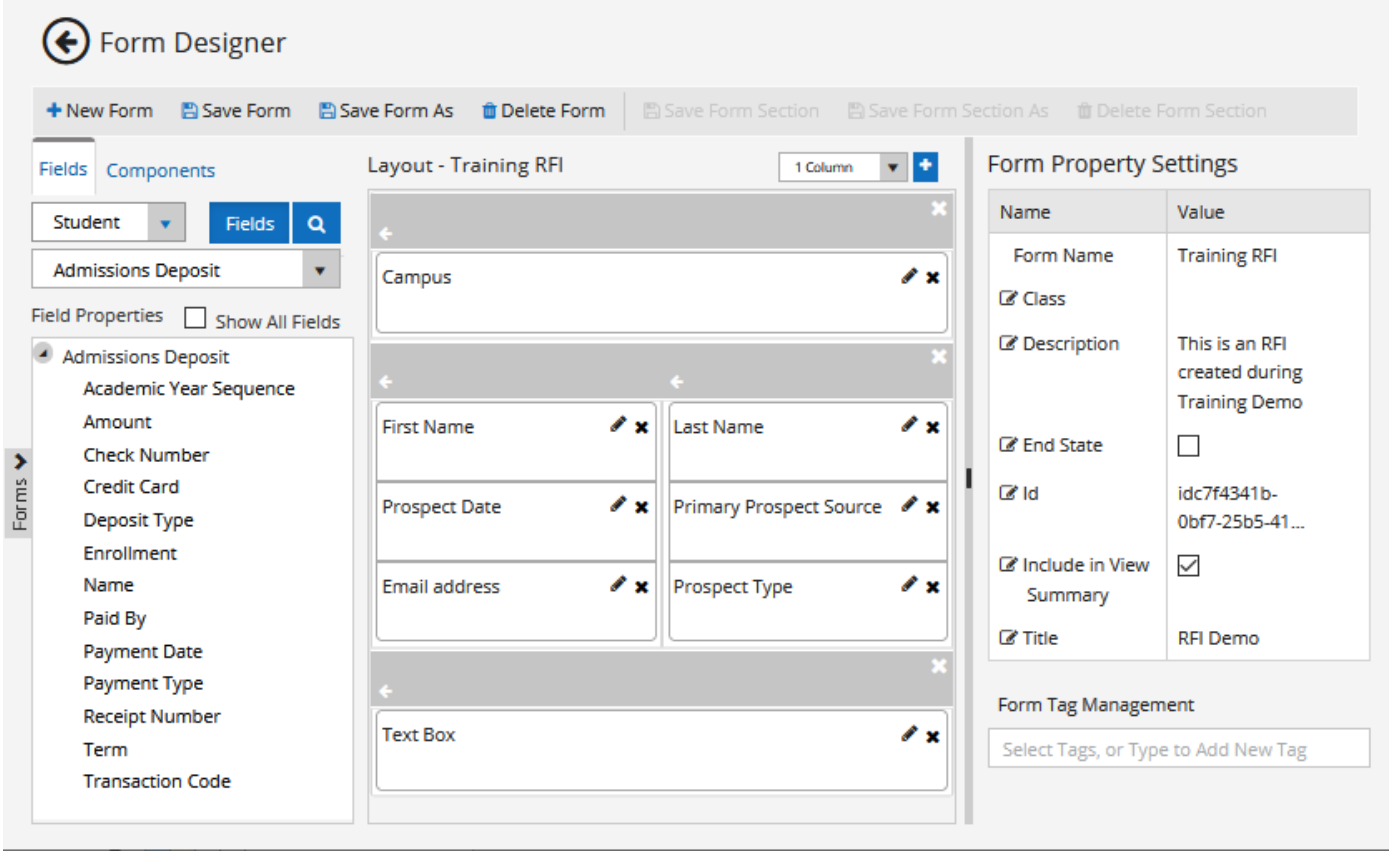
The Import Status column indicates if errors occurred during the import or duplicate records were found.



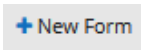
When you have completed the import process, check the imported sequences in Renderer to make sure that they are imported correctly.

Form Designer

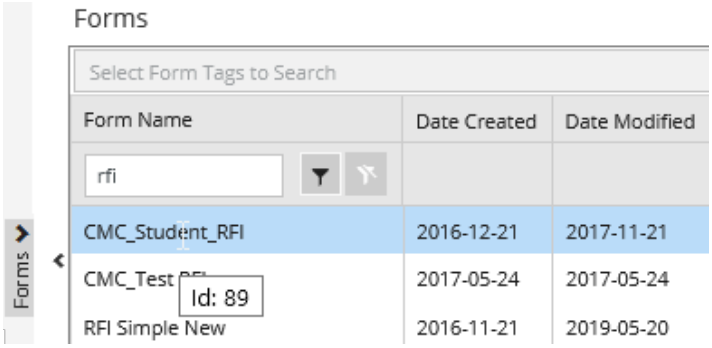
The Form Designer workspace is displayed when you select the Form Designer tile in the home page. This workspace enables you to create and edit forms. The workspace presents database entities with associated fields and custom controls available for the form design.

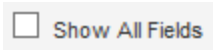
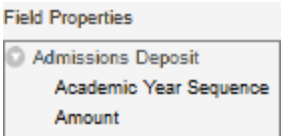


Form Designer UI Elements

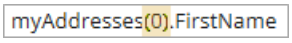
Element	Description
	Return to the Forms Builder home page.
	Create a new form and clear the Layout pane. On the initial entry to the page, the Layout pane is disabled until **New Form** is selected.
	Save a form. This button is enabled only if a form is selected. When you select the Save button, the Unsaved Changes Dialog is displayed. The form is validated before it is saved. Refer to Validation on Form Save.

Element	Description
Save Form As	Save a form with a different Form Name, Title, Description, or End State. This button is enabled only if a form is selected.
Delete Form	Delete a form. This button is enabled only if a form is selected. Refer to Delete Forms.
Save Form Section	Save a form section. This button is enabled only if a control or form section is selected in the Layout pane. The Unsaved Changes dialog is displayed prompting you to specify the **Form Section Name** (required), **Title** (optional), and **Description** (optional). The Title, if assigned, appears on the section divider in the Layout pane. Refer to Form Sections.
Save Form Section As	Save a form section with a different Form Section Name, Title, or Description. This button is enabled only if a form section is selected.
Delete Form Section	Delete a form section. This button is enabled only if a form section is selected.
Forms (flyout)	

Element	Description
Forms	Select the vertical tab on the left to access saved forms. The list of all saved forms is folded out. The Forms flyout provides the following search and filter options:  • Tags – Select one or more tags created in the Form Property Settings pane. • Date Created – Select the column header to filter the list. • Date Modified – Select the column header to filter the list. • Text search by form name – Select to filter the list. When you hover over a form name, the form Id is displayed. When you select a form: • The form fields/controls are displayed in the Layout pane. • The form properties and tags are displayed in the Form Property Settings pane.
Fields Tab	
Select the Fields tab to access controls for database fields. When you select a control in the Layout pane, the Property Settings pane shows the attributes for the selected control. You can edit the attribute values. For more information, refer to Fields.	
<Select Provider>	Use this drop-down list to select the product that provides the database. The drop-down list is not displayed if only one database provider is available. The options are **CRM** and **Student**. Depending on the selected option, the Entities drop-down list will contain the entities associated with the data source.

Element	Description
	Select the Fields button and select an entity in the drop-down list below the Fields button to populate the Field Properties pane. Select the search tool to locate database fields. The entered search/-filter string not only lists properties (fields) that match what is entered but will also include any entities that contain the entered string. Enter at least 3 characters of a field name. The Field Properties pane will display the fields found in the database. The fields are categorized by entities.
	Select an entity from the drop-down list. The Field Properties pane will list the available fields for the selected entity. For more information, refer to Anthology Student Entities and CampusNexus CRM Entities.
	The Show All Fields check box controls whether a subset or the entire list of fields (categorized by entities) in the CampusNexus data model is exposed. The Show All Fields check box is cleared by default. When cleared, a prescribed list of entities is exposed. The prescribed list of entities was determined by analysis of the fields available in Forms Builder 2.x. The prescribed list of entities is aligned as much as possible with what is exposed in Forms Builder 2.x. • If you want to access a property that is not in the prescribed list, select the **Show All Fields** check box to expose the entire CampusNexus data model. • If you want to set up a custom filter for exposed entities and entity properties, configure the Entity and Entity Properties Visibility under **Settings**.
	The Field Properties pane displays the fields available for a selected entity. Depending on the selected entity, navigation properties may be available. Navigation properties signify a relationship between the selected entity and another entity in the data model. Select to expand the navigation property entity. Drag items from the Field Properties pane into the Layout pane to build a form. When a field is brought into focus (select the item in the Layout pane), the Property Settings pane is refreshed and shows the attributes for the selected field.
Components Tab	
Select the Components tab to access custom controls. When you select a control in the Layout pane, the Control Property Settings pane shows the associated properties. You can edit the property values. When you save a form, any custom controls (and their property values) are persisted with the rest of the form data if the custom controls are bound to an `In/Out` argument in the workflow using the Model property. For more information, refer to Components.	

Element	Description
Form Sections Tab	
Select the Form Sections tab to access form sections. Form sections contain fields and components that are reused in multiple forms. For more information, refer to Form Sections.	
Layout Pane	
Use the Layout pane to assemble and organize controls/fields and form sections. Drag controls from the Fields and Components tabs or form sections into the Layout pane and rearrange them in the Layout pane. You must drag a control/field over another control. You cannot drag a control over an empty space in the Layout pane. If you drag a control over the top half of an existing control, it will be inserted before the existing control; if you drag it over the bottom half, it will be inserted after.	
	Select the maximum number of columns for a form section (1-12 columns). For more details about the form section design, refer to Layout Enhancements. The default value for the number of columns in a new form is 1. For a form that was previously saved, the default value for the number of columns is the last value specified when saving the form.
	Add a form section.
	Remove a form section.
	Remove a field.
	Use this tool to specify an optional short description for each control. This is user information about the purpose of the control. It does not affect the rendered component. Overflow text is visible within the tooltip.
Property Settings Pane	
The Property Settings pane displays the property settings of a form, control, or form section. Use the slider to the left of the Properties Settings pane to adjust the width of the pane. You can also adjust the widths of the Name and Value columns.	
Form Property Settings	When a form is loaded from the Forms tab, the Property Settings pane displays the form properties including Form Name, Title, etc. These properties are saved with the form. For more information, refer to Save Forms and Form Properties. When the control or form section properties are displayed, select the form title above the Layout pane to recall the form properties in the Property Settings pane.
From Section Property Settings	When a form section is selected in the Layout pane, the Form Section Property Settings are shown. For more information, refer to Form Sections.

Element	Description
Control Property Settings	When a control (Field or Component) is selected in the Layout pane, the Control Property Settings are shown. The Property Settings pane exposes the metadata for the selected control. Metadata include default values. Some property values are editable; others are read-only. When a form instance is saved, the property values associated with the controls on that form instance are persisted to the database. For more information, refer to Control Property Settings. **Notation for array variables** Array variables in the Property Settings pane of Form Designer use AngularJS notation with "square brackets" []. *Example*  Array variables in Workflow Composer require VB.NET notation with "rounded brackets" (). *Example* 

AutoSave for Forms

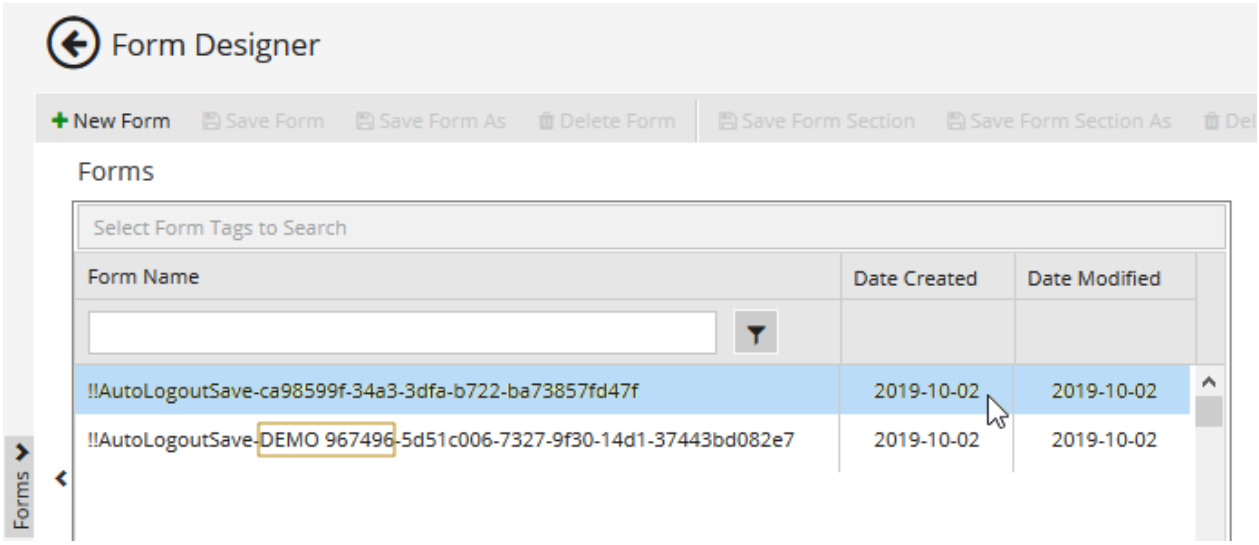
Forms Builder 3.7 and later provides an autosave feature for forms being created in Form Designer. This helps to recover any work in progress if a timeout occurs and you did not have a chance to save.

If you were in the process of creating a new form or editing an existing form, the autosaved form will be listed in the Forms flyout tab named as `!!AutoLogoutSave-<GUID>`. The GUID value ensures that each autosaved form has a unique file name. The date stamp enables you to identify the form to be recovered.

- In case of a new form, select the autosaved form and save it with the desired name.
- In case of an existing form, you can select the autosaved form and resave the entire form with the changes made before the timeout occurred or copy/paste a section from the autosaved form to the original form.

  When an existing form was being edited, the autosaved form name will include the original form name, i.e., `!!AutoLogoutSave-<previous form name>-<GUID>`.

  In the image below the original form was named "DEMO 967496".



Binding

The process of associating an attribute with a name (or method) is called binding. An attribute can be classified according to the time during the execution process when it is bound to a name. Binding times can be classified into two categories: static binding and dynamic binding.

Static binding occurs prior to the execution. Static binding occurs when the code is compiled.

Dynamic binding occurs during the execution. Dynamic binding occurs at run time when a function is called, or a value is assigned to variable.

An attribute that is statically bound is a static attribute, while an attribute that is dynamically bound is a dynamic attribute.

Binding in Forms Builder

In Forms Builder we refer to a bound property when a property value is set by a workflow value **before** the form is rendered. A property is dynamically bound if a property value, when set, can be changed by some other component, custom JavaScript, or data entry **after** the form is rendered.

Initial values can be set using bindings. For bound properties, initial values can be set with custom JavaScript, (refer to Set Default Values for Form Fields), if the value is set before the form renders (e.g., in the previous form). For dynamically bound properties, initial values can be set in the same form.

Model Property and Arguments

Component properties can be bound to workflows by specifying a value for the **Model** property. The workflows for form sequences can then be manipulated using activities and VB code to implement specific business rules.

In addition to the Model property, other properties can be bound to workflows. For example, you can set default values for controls, validate input, make controls visible or invisible depending on conditions defined in the workflow, and so on.

The binding of properties is done using **arguments**. The arguments created in Form Designer and Workflow Composer enable the flow of data into and out of Forms Builder and workflow activities.

The syntax for bindings is: `vm.models.<argument>` where <argument> is case sensitive and must be unique on the form and within the sequence.

When binding controls, **String** and **Integer** properties require the Model value to be enclosed in **double curly braces**, for example, `{{vm.models.myTooltip}}` for Tooltip or `{{vm.models.myMinValue}}`for a Text Box of Type Number.

Boolean properties do not need the curly braces, for example, `vm.models.myRequired`.

Specify Model Value and Create Argument

1. In Form Designer, specify a value for the Model property (`vm.models.<argument>`).

   > ⚠ Keep in mind that arguments are passed in JSON format and that JSON elements are case sensitive. **Be sure to match the casing of argument names in Workflow Composer and Form Designer.**

2. Specify bindings for any other properties as needed and save the form.
3. In Sequence Designer, add the form to a sequence and save the sequence.
4. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
5. In Workflow Composer, add an **argument** with the following attributes:
   - **Name**: Use the `<argument>` value specified in Form Designer.
   - **Direction**: Use the `In/Out` direction.
   - **Argument type**: Select the appropriate data type for the data that is passed in from Forms Builder (e.g., Boolean, String, DateTime).

Every workflow for a form sequence contains the formInstance, entity, and event arguments which are created automatically by Forms Builder.

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*

Default Arguments

Forms Builder creates default arguments in Workflow Composer for most of the component properties including the Model bindings on initial save of the sequence. Bindings that are added to forms after the initial save of the sequence must be added manually in Workflow Composer.

The default arguments include the In/Out direction and the argument type appropriate for the component.

Default Argument Types for Components

Component	Default Argument Type in Forms Builder 3.5 and later	Comments
⚠ It is good practice to review all argument types to ensure that there are no issues, especially for complex object types in Grids and Calendars. You must update the argument type for query initialized Calendar and Grid components, otherwise the sequence will fail on load.		
Calendar/Scheduler	String[]	

Component	Default Argument Type in Forms Builder 3.5 and later	Comments
Calendar/Scheduler with Model Data	SerializableDynamicObject[]	
Calendar/Scheduler with OData Query	String[]	Update the string array to specify the entity type, for example, CRMEventEntity[].
Checkbox	Boolean	
Date Picker	DateTime	If the Required property is set to false, the default argument type is Nullable<DateTime>.
Date Time Picker	DateTime	If the Required property is set to false, the default argument type is Nullable<DateTime>.
Drop-down List	Int32	If the Lookup Value Member is an Id, the default argument is Int32, otherwise it is String.
Drop-down List with Value List of type "Value List"	String and SerializableDynamicObject[]	Use arguments of type String for selections and SerializableDynamicObject[] for a full custom value list.
Drop-down List with Value List of type "Workflow Initialized List"	String	Verify and/or update the argument type on a case by case basis.
Grid	String[]	Grid is only component that doesn't give a validation warning for an empty Model binding. When a Model binding is not specified, the default workflow argument is not created.
Grid with Model Data	SerializableDynamicObject[]	
Grid with OData Query	String[]	Update the string array to specify the entity type, for example, StudentRelationshipAddressEntity[].
Masked Text Box	String	
Multiselect	Int32[]	If the Lookup Value Member is an Id, the default argument is Int32[], otherwise it is String[].
Multiselect with Value List of type "Value List"	String[] and SerializableDynamicObject[]	Use an argument of type String[] for selections and SerializableDynamicObject[] for a full custom value list.
Multiselect with Value List of type "Workflow Initialized List"	String[]	Verify and/or update the argument type on a case by case basis.
Numeric Text Box	Nullable<int32>	
Radio Button	String	

Component	Default Argument Type in Forms Builder 3.5 and later	Comments
Single-select Search	Int32	If the Lookup Value Member is an Id, the default argument is Int32, otherwise it is String.
Text Box	String	
Time Picker	DateTime	If the Required property is set to false, the default argument type is Nullable<DateTime>.
All other Components	String	Verify and/or update the argument type on a case by case basis.

Casing of Arguments

If the same argument is used with different casing in multiple forms within a sequence, a default argument will be assigned only to the first occurrence.

Example

Form A in sequence A has the argument: `vm.models.verify`

Form B in sequence A has the argument: `vm.models.Verify`

Each component has its own unique id and should have a unique binding. This is currently not validated in Workflow Composer.

Components

The control types listed on the Components tab in Form Designer greatly enhance the flexibility of Forms Builder. The Components tab provides control types such as drop-down lists, grids, date pickers, hyperlinks, text areas, and more. The properties of these controls can be customized as needed.

You can drag the controls from the Components tab into the Layout pane and build forms that include both custom controls and controls based on fields/entities of the Anthology data model.

You must drag a control/field over another control. You cannot drag a control over an empty space in the Layout pane. If you drag a control over the top half of an existing control, it will be inserted before the existing control; if you drag it over the bottom half, it will be inserted after.

The properties of the control types listed on the Components tab can be bound to workflows using arguments. For more information, refer to Binding.

Adobe Sign

You can use the Adobe Sign component to integrate Adobe Sign fields into a form. Forms Builder supports several types of Adobe Sign fields, e.g., Signature, Date Signed, and Initial. The fields are selected in the **Type** property of the Control Property Settings.

Before using the Adobe Sign component, you must set up and account with Adobe and configure the Adobe Sign Settings in Forms Builder.

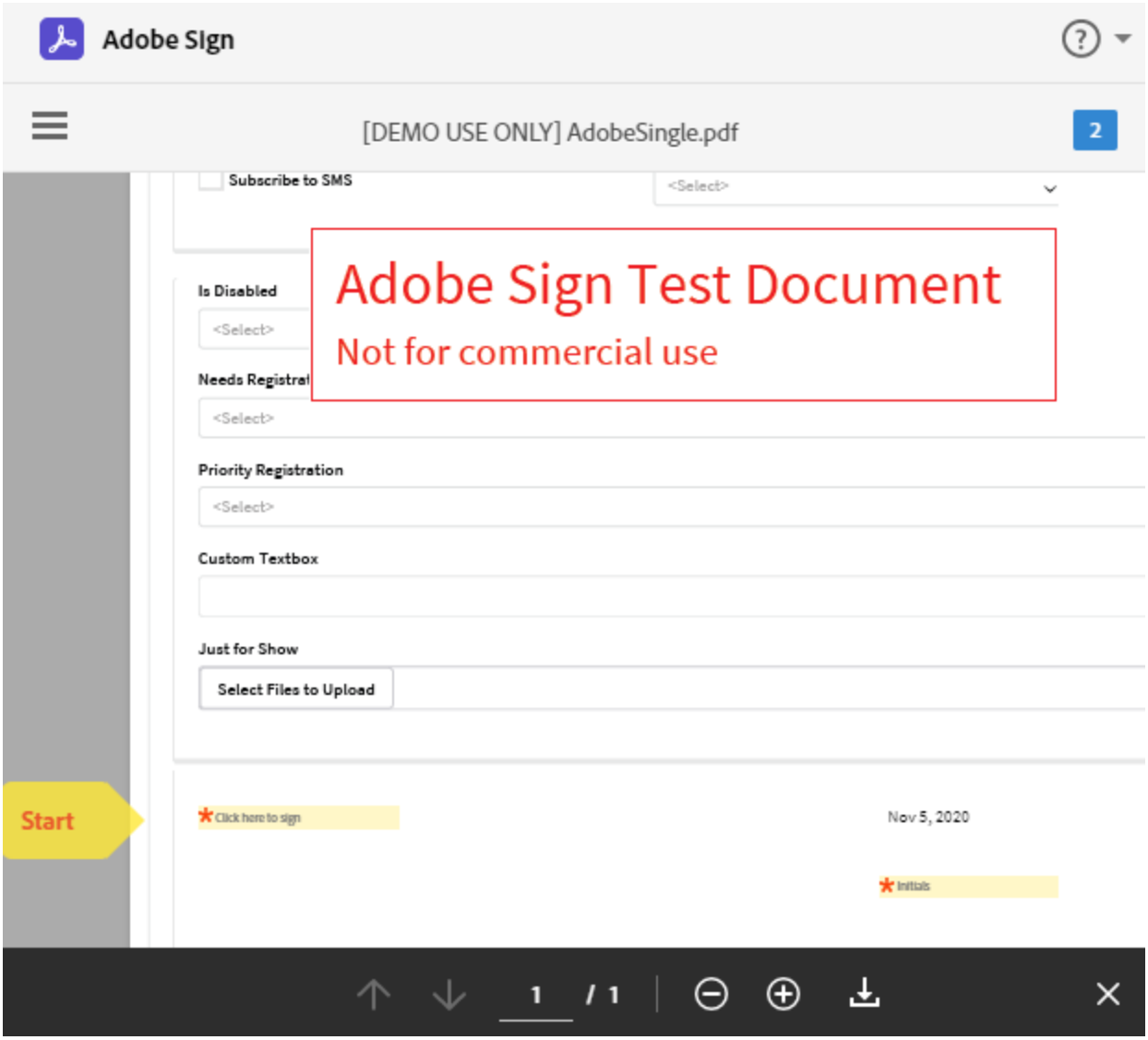
Properties

Control Property Settings

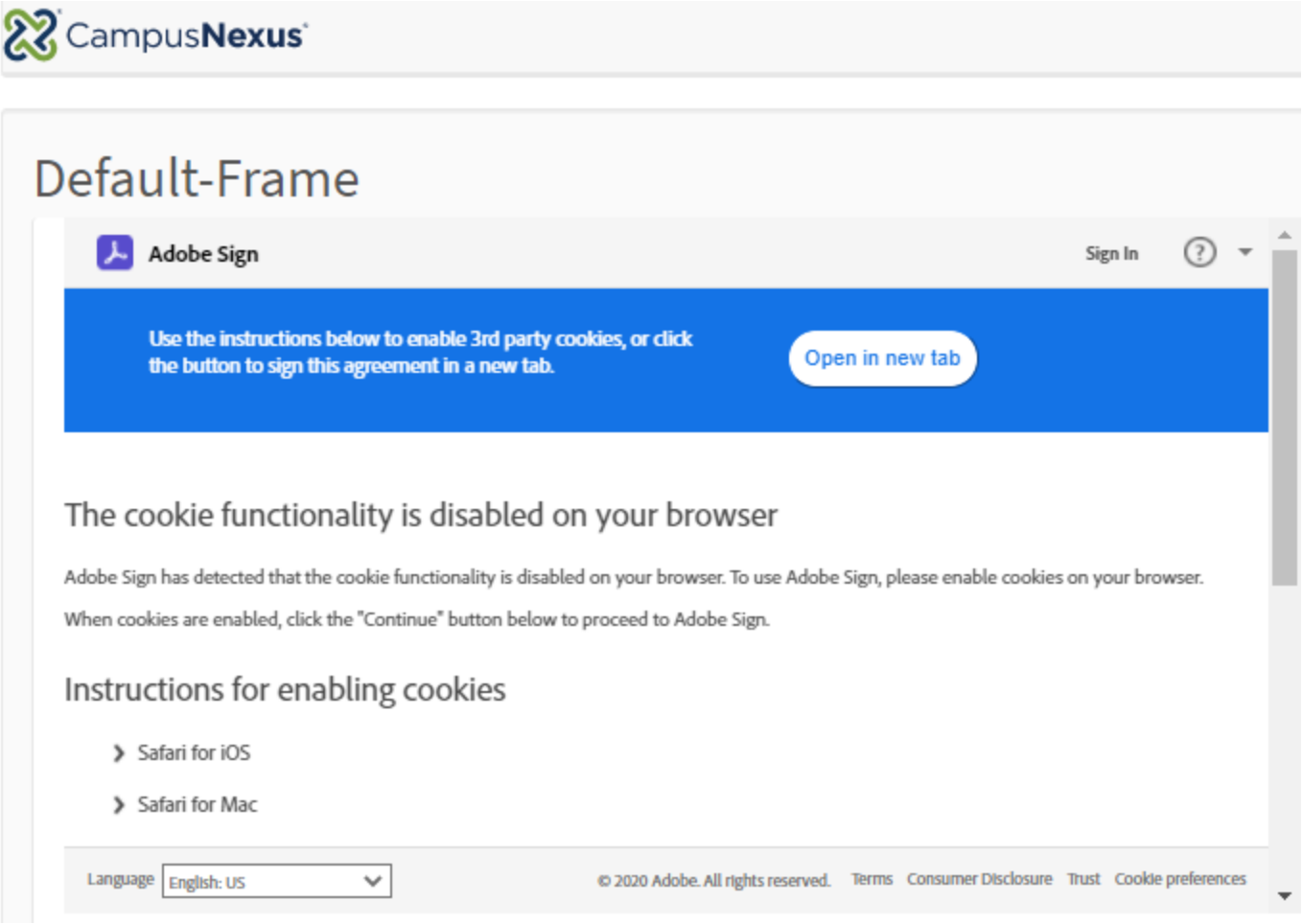
Name	Value
Control Type	Adobe Sign
Class	
Signer	Self - Signer 1
Type	Signature
Visible	true

Name	Value
Control Type	Adobe Sign
Class	
Signer	Self - Signer 1
Type	Initial
Visible	true

Rendered Components (Type=Signature and Type=Initial)



Depending on the user's browser settings, a page with instructions to enable cookies may be displayed before accessing the e-signature document.



- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Font Size** allows to modify the size of the Adobe Sign signature text. This property helps to modify the Font size to make the signatures look bigger when rendered and executed in any document. Below is the default size defined for Font Size and recommended minimum and maximum sizes of the Adobe Sign Field being used as a Signature / Initial / Date Signed.
    - Default - 17
    - Minimum (Recommended) - 8
    - Maximum (Recommended) - 30
- **Height** allows to modify the height of the Adobe Sign signature text. This property helps to modify the Height to make the signatures look bigger when rendered and executed in any document. Below is the default size defined for Height and recommended minimum and maximum sizes of the Adobe Sign Field being used as a Signature / Initial / Date Signed.
    - Default - 10
    - Minimum (Recommended) - 8

- Maximum (Recommended) - 20

The Font-Size and / or height are auto-populated to the default defined values. However, if the fields are kept blank, then the system takes the default values defined in Adobe Sign.

- **Id** is required property. It can be any valid javascript id attribute value (Must start with a letter followed by 0 o 9, a to z, dash or underscore characters). It should not have any spaces. Binding is not supported for this property.
- **Signer** provides the following options:
  - Self - Signer 1
  - Signer 2
  - Signer 3
  - Signer 4
  - Signer 5
- **Type** provides the following Adobe Sign fields:

Field	Description	Required / Optional
Date Signed	Allows the recipient to specify the date the document was signed.	Optional
Initial	Allows the recipient to initial the document.	Required
Signature	Allows the recipient to sign a document.	Required

- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)

Working with the Adobe Sign Component

You are designing a form in which you want the person completing the form to sign and date the form electronically.

1. Drag two Adobe Sign components into the Layout pane of your form in Form Designer.
2. On the first Adobe Sign component, set the Signer to **Self - Signer 1** and set the Type to **Signature**.
3. On the second Adobe Sign component, set the Signer to **Self - Signer 1** and set the Type to**Date Signed**.

When Anthology Student fields are inserted into a form, the student is automatically directed to the section that needs an e-signature.

If the fields are not inserted into a form but e-signature is enabled, the student is directed to electronically sign the document and can drag out the signature anywhere on the document.

Calendar/Scheduler

You can use the Calendar/Scheduler component to design forms with a calendar control that is populated with events retrieved from the CampusNexus CRM or Anthology Student database. The rendered form will enable users to register for an event by selecting a link in the calendar control.

The calendar can be read-only or editable. Added entries can be determined by their Id value. Updated and deleted entries can only be determined by storing the original items to a separate list and comparing to the updated list after the user interaction with the calendar is complete.

Enhancements in Forms Builder 3.7

The Calendar component has been enhanced to handle add, edit, and delete cases for calendar entries. The component now returns the EntityState property for the affected entity. The sequence workflow can check if the EntityState property has been added, deleted, or modified. For more information, refer to Calendar with EntityState Property.

Enhancements in Forms Builder 3.5

The code for the Calendar/Scheduler component was completely revised to provide enhanced functionality. Several properties related to time zone offsets were added (refer to Time Zone properties), the Group Header Template property was added, the Model and Model Data properties were added, properties to map data property names to Calendar/Scheduler built-in names were added, and the Autobind and Data Source properties were removed.

When you drag and drop the Calendar/Scheduler into the Layout pane, the underlying functionality will be that of the new component. Existing forms that already include the Calendar/Scheduler continue to execute the previous version of the code. For these forms, if you want to use the enhanced functionality, remove the Calendar/Scheduler from the previously created form and then re-drag and drop the Calendar/Scheduler into the Layout pane.

Mapped Properties

Special properties were added in Forms Builder 3.5 to map built-in names of the Calendar/Scheduler control to names of OData properties. The mapped properties allow for field names coming from an OData query to not match hard-coded values, i.e., the event fields can be "mapped" to new values.

Mapped Properties

Component Property	Default	Mapped to
Mapped Description	EventDescription	description
Mapped End	EventEndDate	end
Mapped ID	EventId	id
Mapped Start	EventStartDate	start
Mapped Title	EventName	title

Component Property	Default	Mapped to
Mapped URL	EventUrl	eventUrl and sequenceUrl

Example

When a CRM query for events returns the Event Data URL as "RegistrationURL", the Mapped URL property would be updated to "RegistrationURL".

When mapped names are matched to data properties in an OData query, the properties in the OData query are not case sensitive because SQL is not case sensitive.

When mapped names are used in templates, they are case sensitive. When the mapped names are matched, they create properties which will be used in templates. The default properties in templates are cased as follows:

- start
- end
- id
- description
- title
- eventUrl (also "sequenceUrl" which is the same property with a different variable name. It is maintained for compatibility with existing forms.)

Either the property name or the mapped name can be used in a template. The start, end, and id properties are not usually used in a template but are necessary to render the Calendar/Scheduler. You can use any other property name found in the data in a template. No mapping is necessary for them.

⚠ All mapped value properties need to match templates, Model Data references, and/or OData query result fields.

Basic Properties for a Calendar/Scheduler

Since the Calendar/Scheduler has an overwhelming set of properties, we have added the steps to build a basic Calendar/Scheduler and provided the following examples of property settings to initialize a calendar control:

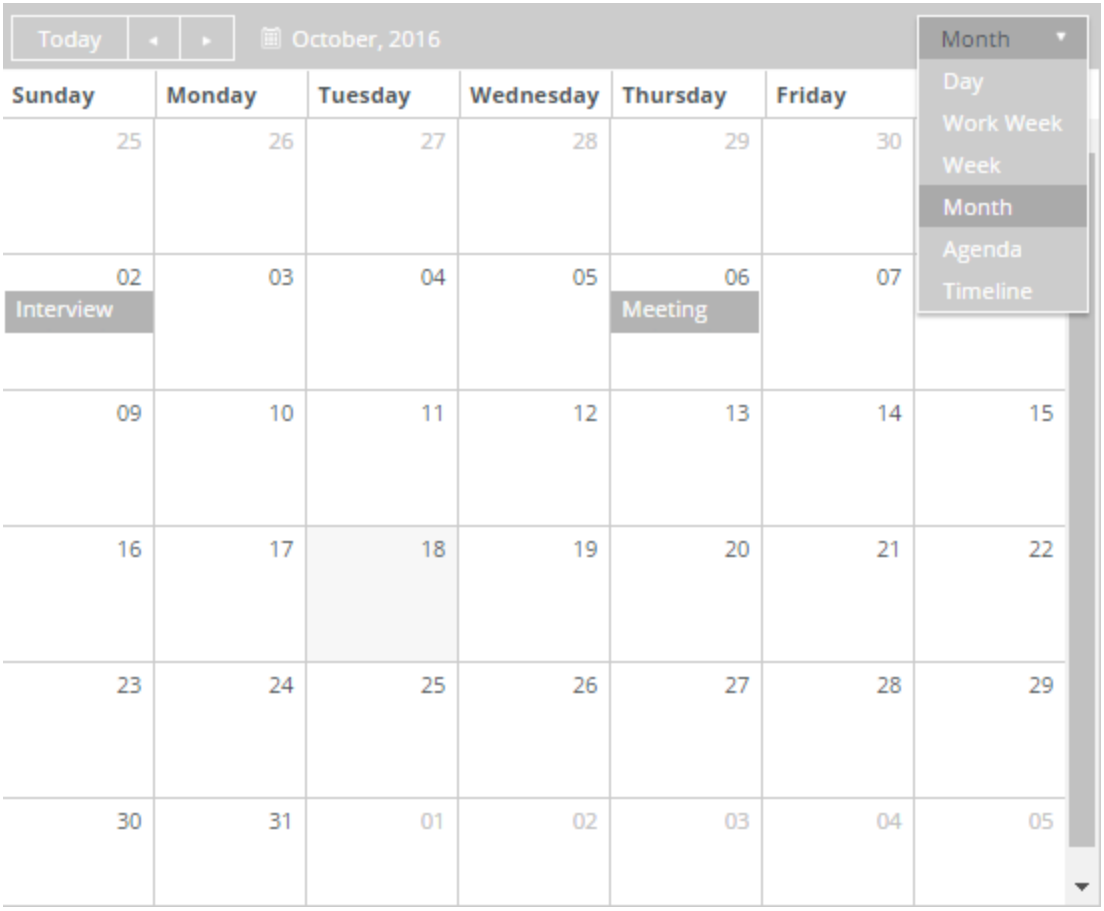
- Minimal Calendar/Scheduler
- Calendar/Scheduler Initialized by Model Data
- Calendar/Scheduler Initialized by OData Query

For workflow arguments used with the Calendar/Scheduler in Forms Builder 3.5 and later, refer to Default Argument Types for Components.

Control Property Settings

Control Type	Calendar/Scheduler
All Day Event Template	
All Day Slot	false
Class	
Current Time Marker	false
Current Time Marker Update Interval	
Current Time Marker Use Local Timezone	true
Date	
Date Header Template	
Default Event URL	
Delete Confirmation	true
Edit Create	☑
Edit Destroy	☑
Edit Move	☑
Edit Recurring Events	dialog
Edit Resize	☑
Edit Template	Add or Edit meeting <label for="title">Title</label> <input id="title" name="title" class="k-input k-textbox" required="required" type="text" data-required-msg="Title is required for an event" /> <label for="start">Start</label> <input id="start" data-role="datetimepicker" name="start" required="required" data-required-msg="Start date/time is required" /> <label for="end">End</label> <input id="end" data-role="datetimepicker" name="end" required="required" data-required-msg="End date/time is required"/> <label for="eventUrl">Sequence URL</label> <input name="eventUrl" data-bind="text:eventUrl" class="k-input k-textbox" name="eventUrl" required="required" data-required-msg="Event URL is required for an event." type="url" data-url-msg="URL must be a validly formatted URL" />
Edit Update	☑
Editable	☐

Rendered Component



Properties

The Calendar/Scheduler component provides numerous property settings, the majority of which are for more complex scenarios and are not needed for a simple calendar. The default component has an example of the only properties that are needed. The following descriptions reflect the rollover text in the Property Settings pane.

Properties values in quotes must use double quotes. Template is an HTML string.

Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **All Day Event Template** is the template used to render "all day" scheduler events. The fields which can be used in the template are:
    - description - the event description (String)
    - end - the event end date (Date)
    - isAllDay - if true, the event is "all day" (Boolean)
    - resources - the event resources (Array)

- start - the event start date (Date)
- title - the event title (String)

- **All Day Slot** — If set to true, the scheduler will display a slot for "all day" events. Default: false.
  - If this property is bound, it must start with "vm.models.".
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
  - If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Auto Bind** — In Forms Builder 3.5, this property is removed. It is always set to true.
- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Current Time Marker** — If this property is set to true, the "current time" marker of the scheduler will be displayed. Default: false.
  - If this property is bound, it must start with "vm.models.".
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
  - If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Current Time Marker Update Interval** — This is the update interval of the "current time" marker in milliseconds. Default: 100000 (10 seconds)

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Current Time Marker Use Local Timezone** — If this property is set to false, the "current time" marker will be displayed using the scheduler time zone. Default: true

- If this property is bound, it must start with "vm.models.".
- A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
- If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Date** sets the current date of the event scheduler. The date is used to determine the period which is displayed. If date is not specified, the current date is assumed.

  To provide forward compatibility, the date string should be ISO 8601 compatible format, e.g., 2017-09-23 (which is a universal non-ambiguous format).

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Date Header Template** is the template used to render the date header cells. Template is an HTML string.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Default Event Url** — If no Event Url is contained in the provider event data, this URL will be used for the event.

  The provider EventId will be added to the end of the URL as ?EventId=27 or &EventId=27. The former will occur if the URL ends with a / or ? and the latter otherwise. A workflow can use it to look up the event.

  This information will be available in the workflow as formInstance.QueryParams.DataDictionary ("EventId"), e.g.,
  - http://myserver/#/renderer/27/?EventId=1020 or
  - http://myserver/#/renderer/27/?Mydata=xxxx&EventId=1020

- **Delete Confirmation** — If this property set to true, the scheduler will display a confirmation dialog when the user selects the "destroy" X button on an event. If this is neither true nor false, then a value is interpreted as a string value that becomes the confirmation message. Default: true.
  - This property is not used for event calendars where Editable is false.
  - This property is not bindable.
  - (true and false must be all lowercase)

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Edit Create** — If this property is set to true (default), the user can create new events. This property is not used for event calendars where Editable is false. This property is not bindable.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Edit Destroy** — If this property is set to true (default), the user can delete events. This property is not used for event calendars where Editable is false. This property is not bindable.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Edit Move** — If this property is set to true (default), the scheduler allows event moving. Dragging the event changes the start and end time. This property is not used for event calendars where Editable is false. This property is not bindable.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Edit Recurring Events** — This property sets the edit mode for recurring events. The available modes are: "dialog" (default), "occurrence", and "series". This property is not used for event calendars where Editable is false. A blank Edit template will contain this property.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Edit Resize** — If this property is set to true (default), the scheduler allows event resizing. Dragging the resize handles changes the start or end time of the event. This property is not used for event calendars where Editable is false. This property is not bindable.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Edit Template** allows the user to edit the template which renders the popup editor. Template is an HTML string.

  The default template is:

```
<h3 style="margin-left: 100px">Add or Edit meeting</h3>
    <div class="k-edit-label">
       <label for="title">Title</label>
    </div>
    <div data-container-for="title" class="k-edit-field">
       <input id="title" name="title" class="k-input k-textbox" required="required"
type="text" data-required-msg="Title is required for an event" />
    </div>
    <div class="k-edit-label">
       <label for="start">Start</label>
    </div>
    <div data-container-for="start" class="k-edit-field">
       <input id="start" data-role="datetimepicker" name="start" />
    </div>
    <div class="k-edit-label">
       <label for="end">End</label>
    </div>
    <div data-container-for="end" class="k-edit-field">
```

```
        <input id="end" data-role="datetimepicker" name="end" />
    </div>
    <div class="k-edit-label">
        <label for="eventUrl">Sequence URL</label>
    </div>
    <div data-container-for="eventUrl" class="k-edit-field">
        <input name="eventUrl" data-bind="text:eventUrl" class="k-input k-textbox" name-
e="eventUrl" required="required" data-required-msg="Event URL is required for an
event." type="url" data-url-msg="URL must be a validly formatted URL" />
    </div>
```

The title has a validation and “required” has a message to override the default validation message. The eventUrl has two validations, a “required” and a “url” type validation which validates a URL. Both have override messages for their validation. It would be unusual to use anything except the standard HTML5 validations. Special code would be required for custom validations, so this would take some JavaScript expertise. Refer to Custom Validations for more information about validation.

This default template matches the Event Template and the Schema Model for property names. Failure to match property names to mapped names or use of properties not defined in the model can lead to JavaScript errors during render.

Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Edit Update** — If this property is set to true (default), the user can update events. This property is not used for event calendars where Editable is false. This property is not bindable.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Editable** — If this property is set to true, the user can create new scheduler events and modify or delete existing events. For event calendars this is normally false. Default: false.

  - This property is not bindable.
  - When an event is editable, it cannot use custom property names. It must use the start, end, id, title, and description properties.

  If a user selects Cancel while adding or editing a new appointment, it is deleted. If user selects Cancel on an existing appointment, the popup editor closes without saving changes.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **End Time** sets the end time of the week and day views. Overridden by data from the schedule provider if supplied.

  Example

  2016-10-1 19:00 or 2016-10-1 7:00 PM.

  You must specify a date with the time because the time by itself will not be parsed correctly.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Event Template** is the template used to render the scheduler events. The fields which can be used in the template are:
    - description - the event description (String)
    - end - the event end date (Date)
    - resources - the event resources (Array)
    - start - the event start date (Date)
    - title - the event title (String)

  Additional fields defined in the Event data can be used in a template. They MUST match the properties returned in the data. Additional properties cannot be used if Editable is true without defining them in the Schema Model.

  Template is an HTML string.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

  Example

```
<div class="event-template">
  <p>
    <span class="event-template-title">#: title #</span>
      
    <span class="event-template-url">
      <a href="#= EventUrl #" target="_top">Registration</a>
    </span>
  </p>
  <p>
    <span class="event-template-description">#: description #</span>
  </p>
</div>
```

  If this property (or any other in this template) is missing in the data (normal), it will cause an error. This should match the data property that you are using for this value. The Mapped URL default is “EventUrl, so this matches the template.

- **Footer** — If this property is set to false, the footer of the scheduler will not be displayed.
    - If this property is bound, it must start with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)

Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Footer Command** sets the command which will be displayed in the scheduler footer. Currently, only the "workDay" option is supported. If the option is set to false, the "workDay" button will be removed from the footer.
    - Properties values in quotes must use double quotes.
    - (true and false must be all lowercase)

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Group Header Template** is the template used to render the group headers of scheduler day, week, workWeek, and timeline views. Template is an HTML string.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Group Object** sets the configuration of the scheduler resource(s) grouping. Properties values in quotes must use double quotes.

  This is an extensive object with many properties. Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Height**sets the height of the widget. Numeric values are treated as pixels. Default: 600.
    - If it is bound it must begin with {{vm.models. and end with }}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Major Tick** sets the number of minutes represented by a major tick. Default: 60.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Major Time Header Template** is the template used to render the major ticks. Template is an HTML string.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Mapped Description** – sets the data property name that maps to “description”. The default is “EventDescription”.
- **Mapped End** – sets the data property name that maps to “end”. The default is “EventEndDate”. “end” is a built-in name required by the Calendar/Scheduler, so if your data does not have this name, set the mapped name.
- **Mapped ID** – sets the data property name that maps to “id”. Default is “EventId”. The “id” in the event data must be unique for every event, or the calendar may fail to work as expected.
- **Mapped Start** - sets the data property name that maps to “start”. The default is “EventStartDate”. “start” is a built-in name required by the Calendar/Scheduler, so if your data does not have this name, set the mapped name.
- **Mapped Title** – sets the data property name that maps to “title”. The default is “EventName”.
- **Mapped URL** – sets the data property name that defines the data item that contains a URL to which the EventId= in the data is appended. The default is “EventUrl”, e.g., the data “EventUrl”:”http://mysite.com/#/renderer/myform” will become http://mysite.com/#/renderer/mysequence?EventId=2232. This “EventId” can be used in a workflow to look up the original event to get more information than was possible to put in the scheduler. Obviously, you can add your own parameters to cause the workflow to do something based on a parameter value. This URL can be embedded in an <a> HTML link in the template.
- **Max Date** constrains the maximum date which can be selected via the scheduler navigation. To provide forward compatibility, the date string should be ISO 8601 compatible format. e.g. 2017-09-23 (which is a universal non-ambiguous format).

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Messages Object** — The configuration of scheduler messages. Use this option to customize or localize scheduler messages.

  Properties values in quotes must use double quotes.

  This is an extensive object with many properties. Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Min Date** constrains the minimum date which can be selected via the scheduler navigation.

  To provide forward compatibility, the date string should be ISO 8601 compatible format, e.g., 2017-09-23 (which is a universal non-ambiguous format).

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Minor Tick Count** sets the number of time slots to display per major tick. Default: 2.
  If set, the minorTickCount value should be set to a number greater than 0.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Minor Time Header Template** is the template used to render the minor ticks. Template is an HTML string.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Mobile Rendering** — If set to true and the Calendar/Scheduler is viewed on a mobile browser, it will use adaptive rendering. The value can be set to a string **"phone"** or **"tablet"** to force the widget to use adaptive rendering regardless of browser type. Default: false.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

  Forms Builder currently does not support rendering of the Calendar/Scheduler component on mobile devices.

- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.

  - The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
  - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
  - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
  - The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
  - If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

  If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

  If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

  Construction of the model in the workflow is done by assigning data from a provider.

  Editing the Calendar/Scheduler currently does not update the Model value. This functionality will be added in a future release.

- **Model Data** — The initial data specified as a JSON object. This will be ignored if data is directly provided by an OData Query from a data provider or when a bound Model is provided.

  Property names and values must use double quotes. Names of properties must match those in a template if they are to be used by the template. The default template has several default property names.

  Dates are entered as ISO 8601 compatible strings and Date objects will be created out of them, e.g., 2016-06-13 or 2016-06-13T09:08:01-05:00 (Google ISO 8601 for explanation).

Refer to Telerik Scheduler documentation for more general information on the Scheduler.

When the Model Data property is used in conjunction with the Locale component and the user navigates back to select a different locale, the data displayed in the sequence will not be updated to reflect the new locale selection.

- **OData Query** — The URL specific part of an OData URI. Base URL and Product will be supplied by the configuration. Does not include the web address or product, e.g., Students?$top=10. This part is the implementation of the specifications for the provider API to get event data.
- **OData Translation Members** is a comma separated list of property names in an OData query string to be translated. You should always validate the query will work in a browser. Only basic errors can be detected in Form Designer.
- **PDF Object** — Configures the Kendo UI Scheduler PDF export settings and is specified by a Json string.
    - Must be specified as a JSON string.
    - Properties values in quotes must use double quotes.

  This is an extensive object with many properties. Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Product** indicates the product from which OData query results are returned. Select from:
    - Student
    - CRM
    - Occupation Insight

  The selected product must be configured in the <products> section of the Renderer web.config file.

  The default Product value will be "Student" if "Student" is selected in the <Select Provider> list on the Fields tab.

  The default Product value will be "CRM" if "CRM" is selected in the <Select Provider> list on the Fields tab.

  Select "Occupation Insight" in the Product property if the source of the query will come from a different data source other than Student/CRM. For more information, refer to Build Queries for Occupation Insight.

  A form can have multiple controls that retrieve data from different providers. For example, a form can have a control that is populated by a query to the Student database. The same form can have another control that retrieves data from Occupation Insight.

  Specify the query to retrieve data from the selected provider using the Lookup Query or OData Query property (as applicable for the control). The query contains only the URL specific part of an OData URI. The Base URL and Product will be supplied by the configuration.
- **Resources Object** — The configuration of the scheduler resource(s). A scheduler resource is optional metadata that can be associated with a scheduler event. Must be specified as a JSON string.

Properties values in quotes must use double quotes.

Overridden by data from the schedule provider if supplied.

This is an extensive object with many properties. Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Schema Model** is specified as a JSON object. The Schema Model is only necessary if using a template such as the Event Template and you wish to add or edit events in the calendar with a **non-standard** data property. Fields that are standard properties like `id`, `start`, `end`, `title`, or `description`, need not be redefined in the schema to be edited; however, non-standard properties (like `eventUrl`) will throw an error (displayed in an F12 console).

  A Schema Model must contain all non-standard properties used in the Event Template. Refer to documentation for the scheduler for the required format of the schema. A default one is provided and includes `eventUrl`, which is a non-standard property that matches the property used in the default Event Template. `id`, `title`, `start`, `end`, and `description` are built-in and only need to be redefined to override them with different values. If the data source contains properties with their mapped name (refer to Mapped Properties), they are populated by those values.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

  Example

  The Schema Model JSON object defines the dataSource schema.model. The JSON which matches the default Event Template is:

```
{
  "id": "id",
  "fields": {
    "id": {
      "type": "number"
    },
    "title": {
    },
    "start": {
      "type": "date"
    },
    "end": {
      "type": "date"
    },
    "description": {},
    "sequenceUrl": {}
  }
}
```

  This JSON code defines two properties: `id` and `fields`. The latter is an object which defines each field in an Event Template. Experimentation has shown that it is not necessary to define the built-in fields `id`, `start`, `end`, and the optional `title` and `description`, unless you want to add something to the

definition like a default value or a type. But you do have to define additional fields you want to use like `sequenceUrl`. Casing is important. For more information, refer to https://docs.telerik.com/kendo-ui/api/javascript/data/model/methods/define

The properties that can be used for each field are `type`, `editable`, `nullable`, `from`, and `validation`. However, `validation` is not useful unless there is some active JavaScript to call the validator.validate() method. Instead validation is defined with attributes in the Edit Template. Refer to Edit Template above.

About the only one that is necessary in most circumstances is `type`, but it defaults to "`string`", which is why some have an empty object definition.

The Editable property has a warning on it. If it is set to true, the Schema Model should be defined, otherwise you will get an F12 error when you use an undefined template variables during add or edit, i.e., `sequenceUrl` is undefined, but the built-in ones `id`, `title`, `description`, `start`, and `end` can be redefined along with `sequenceUrl`. If your data uses the mapped names, the built-in ones will be populated with values from them.

- **Selectable Event** — If this property is set to true, the user can select scheduler cells and events. By default, selection is disabled. This is not currently used by any feature and can be left false. This property is not bindable.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Show Work Hours** — If this property is set to true, the view will be initially shown in business hours mode. By default, the view is displayed in full day mode.
  - This property is not bindable.
  - (true and false must be all lowercase)

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Snap To Slot** — If this property is set to true, the scheduler will snap events to the nearest slot during dragging (resizing or moving). Set it to false to allow free moving and resizing of events. For event calendars, where Editable is false, this is not used.
  - This property is not bindable.
  - (true and false must be all lowercase)

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Start Time** — The start time of the week and day views. The scheduler will display events starting after the startTime, e.g., "2016-10-22 9:00 PM or 2016-10-22 21:00".

  You must specify a date with the time because the time by itself will not be parsed correctly.

  To provide forward compatibility, the date string should be ISO 8601 compatible format. e.g. 2017-09-23 (which is a universal non-ambiguous format).

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
    - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
    - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
    - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

      The tab index value should not match another control's tab index.
    - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.
- **Time Zone** — The time zone which the scheduler will use to display the scheduler appointment dates. Setting this pins the browser relative time zone no matter where the browser is.

  When set and the date does not contain an offset, then the time will not be changed and will be the same in any browser worldwide.

  Examples:
    - "2018-06-15T17:00:00" or "2018/06/15 05:00 PM" which is 5 PM will remain 5PM in every time zone.

      This is also true if a time zone is present and "Time Zone - Ignore Input" is true because the offset will be removed first.

      Otherwise, if an offset is present in the date, then the date-time will be translated to the set time zone first. This could change the date.
    - "2018-06-15T23:00:00-04:00" which is 11 PM EDT (Florida during daylight saving hours) and the set time zone is PST8PDT (California), then this will be translated to "2018-06-15T02:00-07:00" which is 2 AM the next day.

      The scheduler will display this as 2AM in any browser worldwide.

      If not set, the current time zone of the browser is used. This means that while date-times without offsets will not be affected, date-times with offsets will be different in different browser time zones, because the time will be translated to the local time zone.
    - "2018-06-15T17:00:00-08:00" which is 5 PM PDT, will be translated to 8PM EDT for a browser in Florida, but midnight in London.

  A list of IANA Canonical, Alias and Deprecated time zones can be found here.

The list of actual supported time zones is available in the time zone property of the moment-timezone source.

Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Time Zone - Ignore Input** — If true, a time zone (offset) that is present on the incoming data will be removed.

  Example

  "2018-06-15T17:00:00-04:00" which represents 5PM EDT (daylight savings date), will be truncated to "2018-06-15T17:00:00". This actually represents 5PM in any time zone.

  This property is not bindable.

  Use cases

  - Set this property to **true** when the form uses an OData query to retrieve CRM events for an onsite event such a as fixed time onsite campus orientation. A person in a different time zone would need to show up at that fixed time in the destination time zone.
  - Set this property to **false** when the form is used for registration to an online event where you want the user to access the appropriate time zone in the user's location. Online the time is going to be different in each time zone, but the same universal time for everyone.

- **Time Zone - Remove Output** — If true, the time zone is removed from the model date-time.

  Example

  "2018-06-15T17:00:00-04:00" which represents 5PM EDT (daylight savings date), will be truncated to "2018-06-15T17:00:00". This actually represents 5PM in any time zone and information on which time zone it might have been created in is not available.

  - This property is not bindable.

  Use cases

  The setting for this property would almost always match the **Time Zone - Ignore Input** setting, i.e., both true or both false. But if desired, you can set the properties differently.

- **Toolbar Object** — List of commands that the scheduler will display in its toolbar as buttons. Currently supports only the "pdf" command. Properties values in quotes must use double quotes.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **View For Agenda Selection** — On an agenda view, when a non-editable event is selected, this is the view that will be shown, if available.
- **View For Month Selection** — On a month view, when a non-editable event is selected, this is the view that will be shown, if available.

- **View For Week Selection** — On a week or workWeek view, when a non-editable event is selected, this is the view that will be shown, if available.
- **Views Object** — The views displayed by the scheduler and their configuration. The array items can be either objects specifying the view configuration or strings representing the view types (assuming default configuration).
    - If not specified, the Kendo UI Scheduler widget displays "day" and "week" view.
    - Properties values in quotes must use double quotes.
    - If using the "eventTemplate" property, it is an HTML string.
    - If single quotes are required in HTML elements, use '

  This is an extensive object with many properties. Refer to Telerik Scheduler documentation for more general information on the Scheduler.

  Example 1

  This example shows 5 views: day, workweek, week, month, and agenda in JSON format. Each of them can be a simple string if no other properties are required. In this example both month and agenda have extra properties, so they are set as an object with properties, 2 in each case. “type” is required to set the view name. Month is selected by default by the “selected”: true. For other properties which can be added to each of them, refer to the Telerik documentation for views property of the Scheduler.

```
[
  "day",
  "workWeek",
  "week",
  {
    "type": "month",
    "selected": true
  },
  "agenda",
  {
    "type": "timeline",
    "eventHeight": 50
  }
]
```

  Example 2

  In this example, day and month have templates for different views (others could too instead of the default).

  Since the entire property value is in single quotes, single quotes cannot be used again inside the template. Anywhere a single quote is needed inside double-quote HTML elements, the HTML substitute `'` is used.

```
[
  {
    "type": "day",
    "eventTemplate": "<div class=&apos;event-template&apos;><p>Click this link to
go to <span class=&apos;event-template-url&apos;><a href=&apos;#= eventUrl
#&apos; target=&apos;_blank&apos;>Registration</a></span></p></div>"
  },
  "workWeek",
  "week",
  {
    "type": "month",
    "selected": true,
    "eventTemplate": "<div class=&apos;event-template&apos;><p><span class-
s=&apos;event-template-title&apos; title=&apos;#: title #&apos;>#: title
#</span></p></div>"
  },
  "agenda",
  {
    "type": "timeline",
    "eventHeight": 50
  }
]
```

- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)
- **Width** — Sets the width of the widget. Numeric values are treated as pixels. This property is not bindable.

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Work Day End** — Sets the end of the work day when the "Show business hours" button is selected, e.g., "2016/10/1 19:00".

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Work Day Start** — Sets the start of the work day when the "Show business hours" button is selected, e.g., "2016/10/1 09:00 AM".

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.
- **Work Week End** — Sets the end of the work week (index based). Default: 5

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

- **Work Week Start** — Sets the start of the work week (index based). Default: 1

  Refer to Telerik Scheduler documentation for more general information on the Scheduler.

Minimal Calendar/Scheduler

The properties required for a minimal Calendar/Scheduler are as follows (all properties not set use default values).

Step 1: Decide what data to display and understand the data

Once you understand what your data and its properties are, set one of three properties.

- **Model Data**

  Example

  Calendar/Scheduler Initialized by Model Data

  If you are hard-coding the events on a calendar, create the data in this property. An example of data that matches the default Event Template follows. If you copy this data and use Chrome to paste it in the JSON Viewer or JSON Validator on the site https://codebeautify.org/, you will find that it is well formed data.

  It is critical that the values used in the Event Template property match properties in your data. If that is not the case, you must use the Mapped Properties to define the required start, id, and end properties. The title, description, and eventUrl should be mapped as well. The template can use either the built-in id, end, start, title, description, eventUrl or the mapped names. (You must use the built-in ones if the calendar is editable). The mapped names are not case sensitive, but their use in templates is. You will get errors if you use template names that do not exactly match either built-in properties or data properties.

  This data is an object signified by matching braces {} with one property “data”. “data” is an array signified by matching square brackets [] that contains comma-separated objects. Each object in the array has 6 properties. Notice that commas separate multiple objects and multiple properties within the objects.

  For information on JSON syntax, search the internet for JSON Syntax. A good introduction is at https://www.w3schools.com/Js/js_json_intro.asp. Select **Next** to access JSON Syntax rules.

```
{
  "data": [
    {
      "id": 1,
      "start": "2018/6/25 10:00 PM",
      "end": "2018-06-25T23:00",
      "title": "College Fair",
      "description": "This is an all day event",
      "RegistrationUrl": "http://po-170.anthology.com:9003/#/renderer/1224"
    },
    {
      "id": 2,
      "start": "2018-06-25T22:00-04:00",
      "end": "2018-06-25T23:30-04:00",
      "title": "Career Fair",
      "description": "This is a meeting",
      "RegistrationUrl": "http://po-170.anthology.com:9003/#/renderer/1234"
```

```
    },
    {
      "id": 3,
      "start": "2018-06-25T22:00-07:00",
      "end": "2018-06-25T23:00-07:00",
      "title": "Climate Change: Current Policy Challenges ",
      "description": "This is a meeting",
      "RegistrationUrl": "http://po-170.anthology.com:9003/#/renderer/1234"
    },
    {
      "id": 4,
      "start": "2018/6/25 10:00 AM",
      "end": "2018/6/25 11:00 AM",
      "title": "African Heritage Celebration",
      "description": "This is a meeting",
      "RegistrationUrl": "http://po-170.anthology.com:9003/#/renderer/1234"
    },
    {
      "id": 5,
      "start": "2018/6/25 10:00 AM",
      "end": "2018/6/25 11:00 AM",
      "title": "Free Wellness Classes!",
      "description": "This is a meeting",
      "RegistrationUrl": "http://po-170.anthology.com:9003/#/renderer/1234"
    },
    {
      "id": 6,
      "start": "2018/6/25 11:00 AM",
      "end": "2018/6/25 11:30 AM",
      "title": "Philharmonic Orchestra",
      "description": "This is a meeting",
      "RegistrationUrl": "http://po-170.anthology.com:9003/#/renderer/1234"
    }
  ]
}
```


- **Model**

  This value is set as vm.models.xxxx. The xxxx will correspond to a case sensitive argument in a workflow. The data is retrieved from a database or built manually in a workflow variable and then assigned to an argument. One object that can be used if the properties do not correspond to an Entity is a SerializableDynamicObject.

  Example

  Calendar/Scheduler Initialized by Model Data

- **OData Query**

The value is set with an OData query to a provider. You should build this with the provider interface as documented elsewhere and then run the query in a browser. Examine the data closely to make sure that the Event Template and Mapped names are correct for the data. Mismatching the names used in an Event Template to the data properties can produce hard to debug issues in running a sequence.

Example

Calendar/Scheduler Initialized by OData Query

Step 2: Decide on a view

Views are constructed from a JSON object. The view name can be the string name of the view or an object with properties which define the view. Keep it simple to start. Start with one view like a day view. This is an array with only one string that can be verified at https://codebeautify.org/.

```
[
      "day"
]
```

Refer to Telerik Scheduler documentation for more general information on the Scheduler.

Step 3: Look at your data

If your data is not today's date, you will not access the events. Therefore, set the "Date" property to the same day as the data you want to access. It is desirable that the date be set as an ISO 8601 compatible string because this is a universal non-ambiguous format.

Example

2018-06-25 is always year, month, day and can always be parsed non-ambiguously.

Step 4: Decide on an event template

The default event template may not be appropriate for most cases. The template must be well formed HTML with placeholders for the data retrieved for each event. Its display must also fit in a small space, so it can't be a lot of HTML. If you are creating views like a month with very little space per day, use the Event Template property on those views to create a smaller template for them.

In the following default template, notice that the values found in the data above have placeholders bound by either #: xxx # or #=xxx #. The #: renders the string verbatim, while the #= renders the string with HTML encoding (valuable if there are spaces in a URL). If you have custom classes to style the template or images, refer to documentation elsewhere which describes custom styling of HTML.

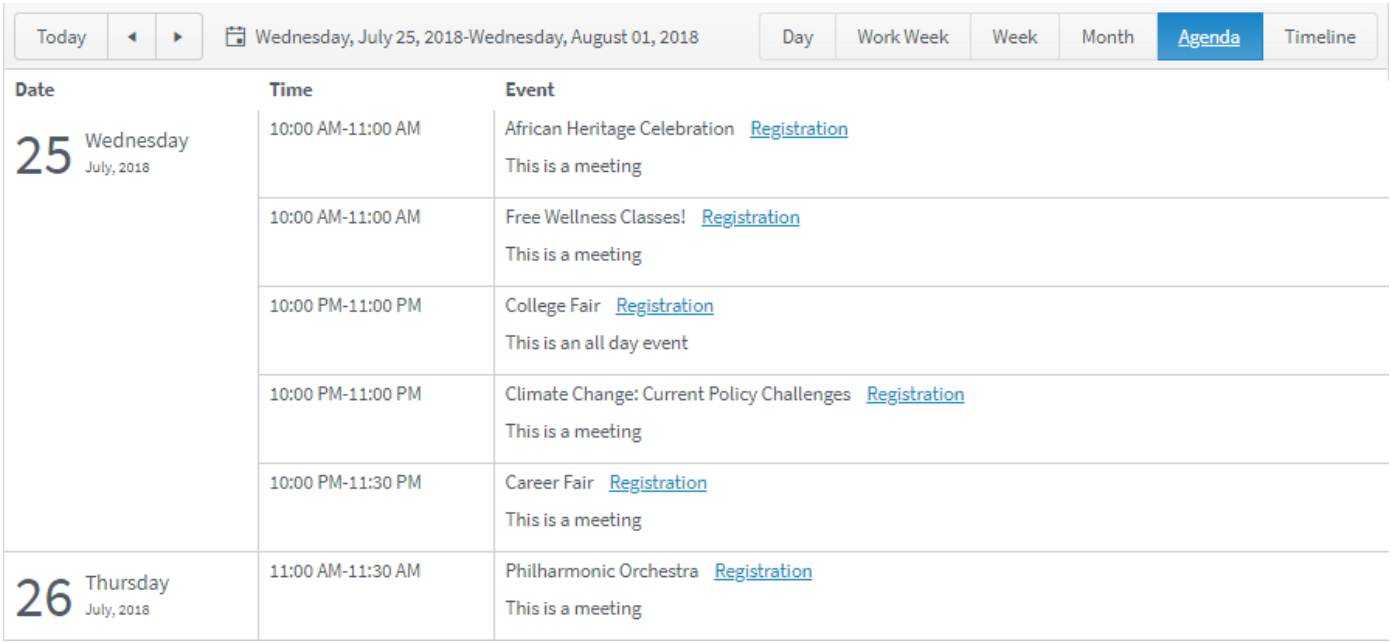
Note that there is a glaring error here. eventUrl does not match any property in the data above, and the default Mapped URL is "EventUrl". The Mapped URL property must change to "RegistrationUrl" to match the data, otherwise an error will occur. Both sequenceUrl and eventUrl will be properties in the internally parsed data, so either can be used in the template below. And in fact, RegistrationUrl could be used instead. However, eventUrl is a special property that causes the eventId={id from the data} to be appended to the URL. That way, the website (or Forms Builder Sequence) you send it to can know which event was selected and can look up other information about the event.

```
<div class="event-template">
	<p>
		<span class="event-template-title">#: title #</span>
		  
		<span class="event-template-url">
		<a href="#= eventUrl #" target="_top">Registration</a>
		</span>
	</p>
	<p>
		<span class="event-template-description">#: description #</span>
	</p>
</div>
```

That should be enough to view a calendar with some events on it.

Calendar/Scheduler Initialized by Model Data

The initial data in the Calendar/Scheduler control can be populated using the Model Data property. In this example the Model Data JSON object defines 6 events as shown in the Agenda view below. Each event defined in Model Data must have at a minimum **id**, **start**, and **end** properties which are required to properly render the calendar.



This example also illustrates how to make a calendar entry editable and how to ignore timezones. The table lists the properties that have been specified for this example. All other properties not listed below use the default values.

Properties Specified for Calendar/Scheduler Initialized by Model Data

Property	Value Specified
Date	2018-07-25
Edit Create	false

Property	Value Specified
Edit Template	Add or Edit meeting <label for="title">Title</label> <input id="title" name="title" class="k-input k-textbox" required="required" type="text" data-required-msg="Title is required for an event" /> <label for="start">Start</label> <input id="start" data-role="datetimepicker" name="start" /> <label for="end">End</label> <input id="end" data-role="datetimepicker" name="end" /> <label for="eventUrl">Sequence URL</label> <input name="eventUrl" data-bind="text:eventUrl" class="k-input k-textbox" name="eventUrl" required-d="required" data-required-msg="Event URL is required for an event." type="url" data-url-msg="URL must be a validly formatted URL" />
Edit Update	true
Editable	true
Event Template	 #: title # Registration #: description #
Model	vm.models.NewCalendarData2

Property	Value Specified

Model Data	{
	"data": [
	{
	"id": 1,
	"start": "2018/7/25 10:00 PM",
	"end": "2018-07-25T23:00",
	"title": "College Fair",
	"description": "This is an all day event",
	"eventUrl": "http://<server>:<port>/#/renderer/3025"
	},
	{
	"id": 2,
	"start": "2018-07-25T22:00-04:00",
	"end": "2018-07-25T23:30-04:00",
	"title": "Career Fair",
	"description": "This is a meeting",
	"eventUrl": "http://<server>:<port>/#/renderer/3025"
	},
	{
	"id": 3,
	"start": "2018-07-25T22:00-07:00",
	"end": "2018-07-25T23:00-07:00",
	"title": "Climate Change: Current Policy Challenges ",
	"description": "This is a meeting",
	"eventUrl": "http://<server>:<port>/#/renderer/3025"
	},
	{
	"id": 4,
	"start": "2018/7/25 10:00 AM",
	"end": "2018/7/25 11:00 AM",
	"title": "African Heritage Celebration",
	"description": "This is a meeting",
	"eventUrl": "http://<server>:<port>/#/renderer/3025"
	},
	{
	"id": 5,
	"start": "2018/7/25 10:00 AM",
	"end": "2018/7/25 11:00 AM",
	"title": "Free Wellness Classes!",
	"description": "This is a meeting",
	"eventUrl": "http://<server>:<port>/#/renderer/3025"
	},
	{
	"id": 6,
	"start": "2018/7/26 11:00 AM",
	"end": "2018/7/26 11:30 AM",
	"title": "Philharmonic Orchestra",
	"description": "This is a meeting",
	"eventUrl": "http://<server>:<port>/#/renderer/3025"
	}
	]

Property	Value Specified
	}
Product	Student

Schema Model	{
	"id": "id",
	"fields": {
	"id": {
	"type": "number"
	},
	"title": {},
	"start": {
	"type": "date"
	},
	"end": {
	"type": "date"
	},
	"description": {},
	"eventUrl": {}
	}
	}

Time Zone - Ignore Input	true
Time Zone - Remove Output	true
View For Month Selection	agenda
View For Week Selec-tion	agenda
Views Object	["day", "week", { "type": "month", "selected": true, “eventHeight”: 50 }, "agenda", { "type": "timeline", "eventHeight": 50}]

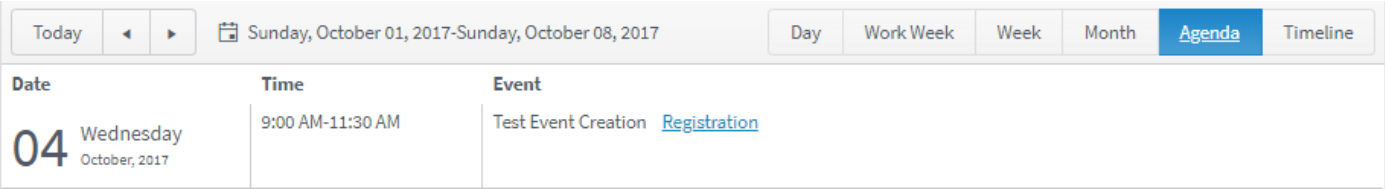
Argument Definition in Workflow Composer (optional if it is desired to use the Model Data in the workflow as well):

CalendarData2	In/Out	SerializableDynamicObject[]

For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

Calendar/Scheduler Initialized by OData Query

The initial data in the calendar control can be populated using the OData Query property. In this example the OData Query retrieves all events from the CampusNexus CRM database for a specific date.



The table lists the properties that have been specified for this example. All other properties not listed below use the default values.

The OData Query uses the value "RegistrationURL". This value is hard-coded in the CampusNexus CRM database. The "RegistrationURL" is mapped to the "eventUrl" value using the Mapped URL property.

The remaining values in the OData Query in this example happen to match the default values of the mapped properties. EventId, EventStartDate, and EventEndDate are required to properly render the calendar.

Properties Specified for Calendar/Scheduler Initialized by OData Query

Property	Value Specified
Date	2017-10-31
Edit Create	false
Event Template	 #: title # Registration #: description #
Mapped Description	EventDescription
Mapped End	EventEndDate
Mapped ID	EventId
Mapped Start	EventStartDate
Mapped Title	EventName
Mapped URL	RegistrationURL
Model	vm.models.NewCalendarQuery
OData Query	Events?$select=**EventId**,**EventName**,**EventStartDate**,**EventEndDate**,**RegistrationURL**&$top=10
Product	CRM

Property	Value Specified
View For Month Selection	agenda
View For Week Selection	agenda

Argument Definition in Workflow Composer:

Name	Direction	Argument type
CalendarQuery	In/Out	Event[]

Where Event is **Cmc.NexusCrm.Events.Entities.Event**

For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

Calendar with EntityState Property

In editable calendars, you can use the EntityState property to determine if calendar entries have been added, deleted, or modified on a form.

Calendar Component Properties

In our example, the sequence contains a form with a Calendar component with the following properties:

Control Property Settings for Calendar Example

Name	Value
Control Type	Calendar/Scheduler
All Day Event Template	
All Day Slot	false
Class	
Current Time Marker	false
Current Time Marker Update Interval	
Current Time Marker Use Local Timezone	true
Date	12/1/2019
Date Header Template	
Default Event URL	http://<server>.<domain>:9003/#/sequencelist
Delete Confirmation	true
Edit Create	
Edit Destroy	☑
Edit Move	☑
Edit Recurring Events	dialog
Edit Resize	☑
Edit Template	Add or Edit meeting <label for="title">Title</label> <input id="title" name="title" class="k-input k-textbox" required="required" type="text" data-required-msg="Title is required for an event" /> <label for="start">Start</label> <input id="start" data-role="datetimepicker" name="start" required="required" data-required-msg="Start date/time is required" /> <label for="end">End</label> <input id="end" data-role="datetimepicker" name="end" required-d="required" data-required-msg="End date/time is required"/>

Name	Value
Edit Update	☑
Editable	☑
End Time	
Event Template	#: title # #: description #
Footer	false
Footer Command	false
Group Header Template	
Group Object	
Height	600
Id	id7d083ee0-6162-41a3-77cd-0a7247bdf93d
Major Tick	60
Major Time Header Template	
Mapped Description	Note
Mapped End	TaskEndDate
Mapped ID	Id
Mapped Start	TaskStartDate
Mapped Title	Subject
Mapped URL	RegistrationURL
Max Date	
Messages Object	
Min Date	
Minor Tick Count	2
Minor Time Header Template	
Mobile Rendering	true
Model	vm.models.NewCalendarQuery2
Model Data	
OData Query	
OData Translation Members	

Name	Value
PDF Object	
Product	Student
Resources Object	
Schema Model	{ "id": "id", "fields": { "id": { "type": "number" }, "title": { "defaultValue": "", "validation": { "required": true } }, "start": { "type": "date" }, "end": { "type": "date" }, "description": {} } }
Selectable Event	☐
Show Work Hours	true
Snap To Slot	false
Start Time	
Tab Index	
Time Zone	
Time Zone - Ignore Input	☑
Time Zone - Remove Output	☑
Toolbar Object	
View For Agenda Selection	action disabled
View For Month Selection	agenda
View For Week Selection	agenda
Views Object	["day", "workWeek", "week", { "type": "month", "selected": true }, "agenda", { "type": "timeline", "eventHeight": 50}]
Visible	true
Width	
Work Day End	
Work Day Start	
Work Week End	5
Work Week Start	1

Workflow

Variables

Name	Variable type	Scope	Default
EventIds	Int32[]	Welcome	*Enter a VB expression*
renderedFormImage	String	StateMachine	*Enter a VB expression*
TmpTask2	TaskEntity	StateMachine	*Enter a VB expression*

Arguments

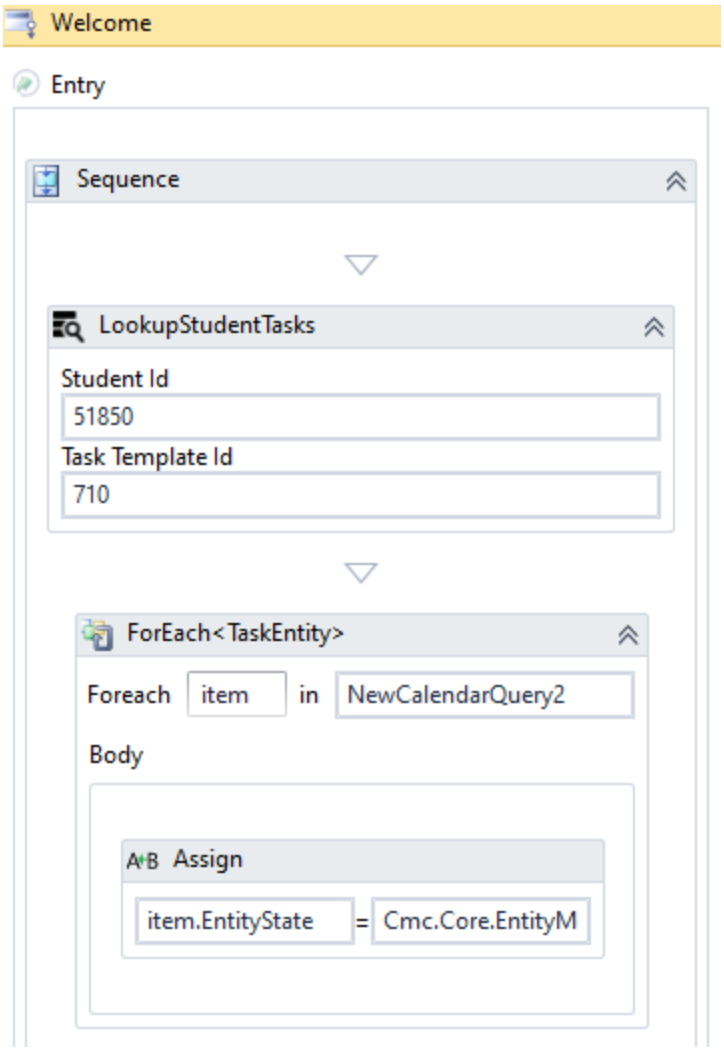
Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*
bottomRequired	In/Out	Boolean	*Default value not supported*
NewCalendarData2	In/Out	SerializableDynamicObject[]	*Default value not supported*
NewCalendarQuery2	In/Out	TaskEntity[]	*Default value not supported*

Welcome State/Form

This example uses the LookupStudentTasks activity to find the TaskEntity.

This is followed by a ForEach<TaskEntity> activity that loops through the each "item" in the argument "NewCalendarQuery2". This argument is bound to the Calendar component via the Model value.

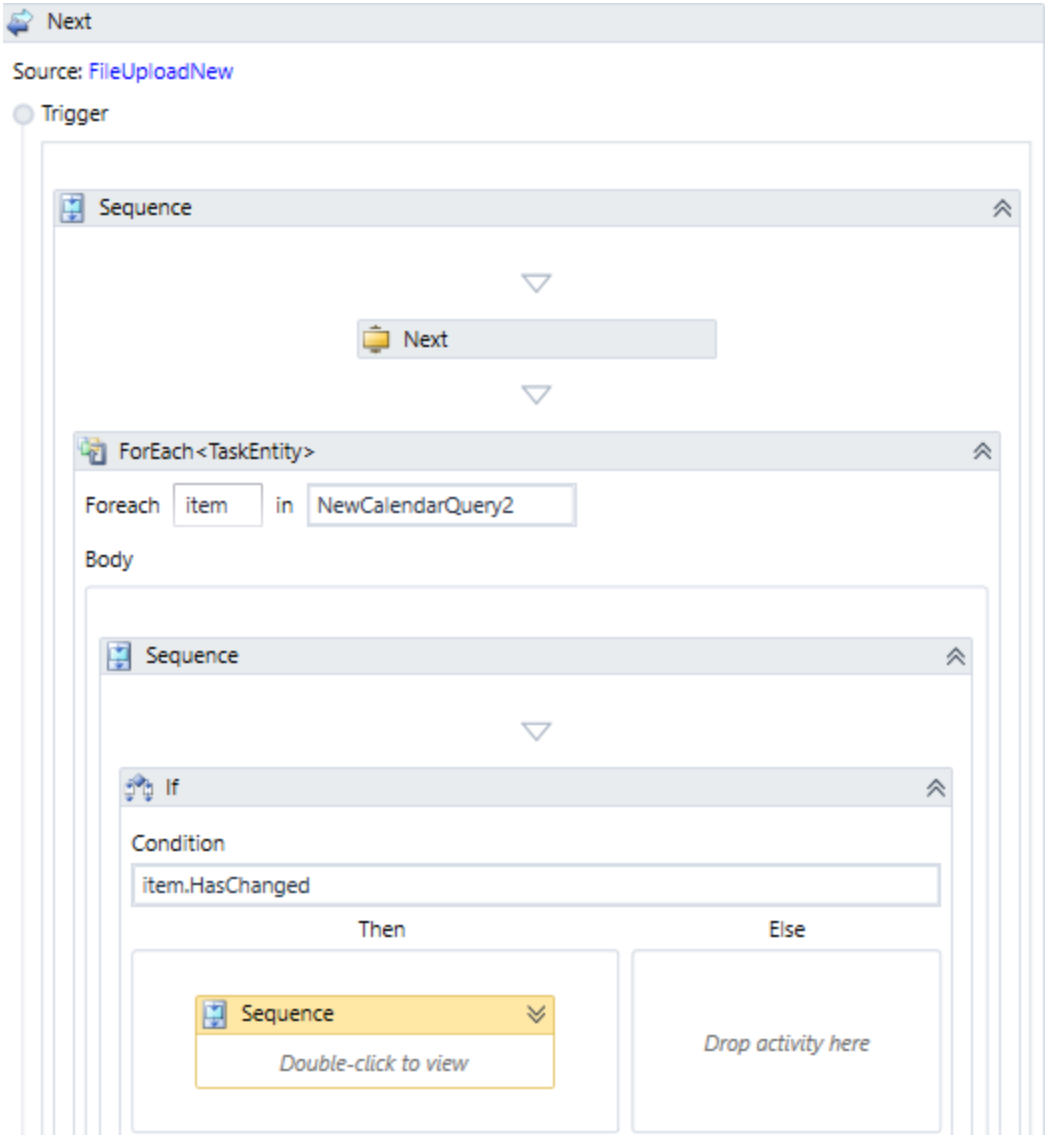
Note the assignment for **item.EntityState = Cmc.Core.EntityModel.EntityState.Unchanged** in the ForEach<TaskEntity> activity. This initializes the calendar when it is loaded on the form.



Final Transition/Next

The final transition contains a sequence with another ForEach<TaskEntity> activity on the "NewCalendarQuery2" argument.

The sequence detects any changes that have been made to the calendar by the user. If a change is detected, another sequence is executed to handle the Modified, Removed, and Added cases.

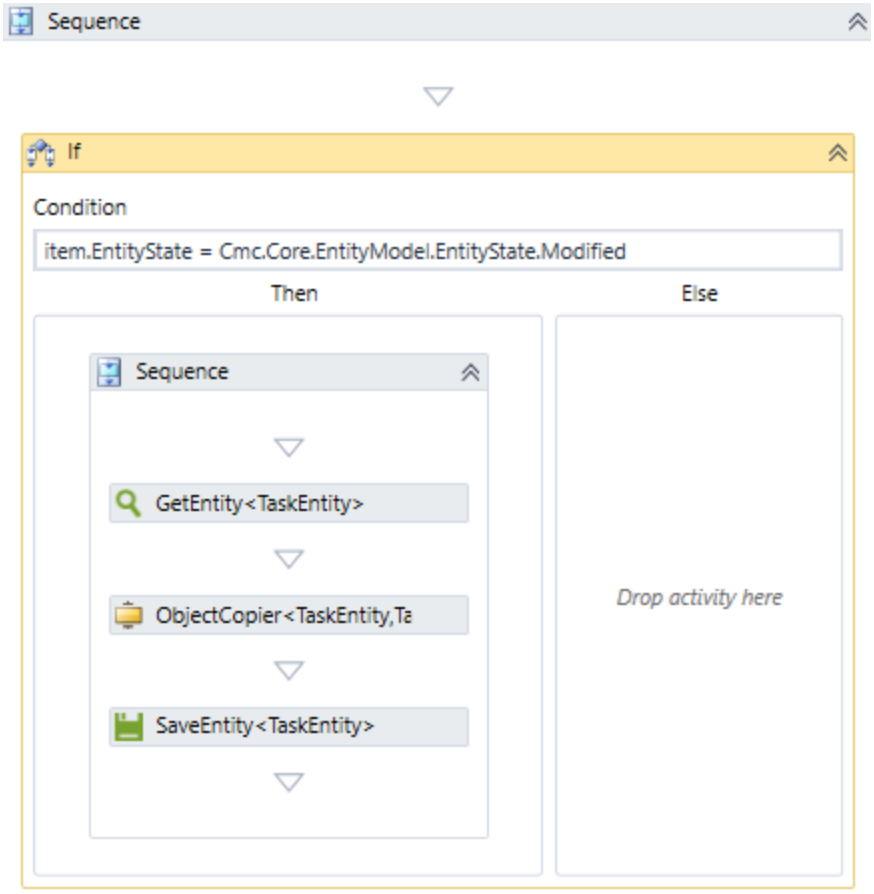


Modified Case

The Modified case is detected with the following condition: **item.EntityState = Cmc.Core.EntityModel.EntityState.Modified**

In this case, the GetEntity, ObjectCopier, and SaveEntity activities are executed.

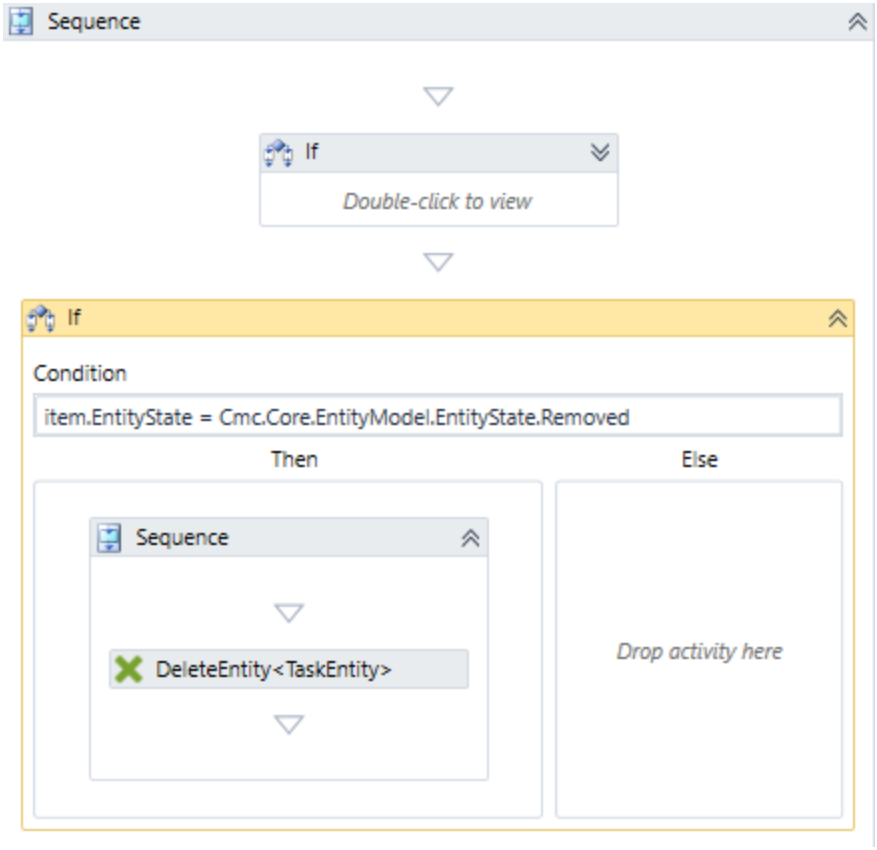
- GetEntity<TaskEntity> retrieves the "item.Id" value and returns the "TmpTask2" value (refer to Variables).
- ObjectCopier(TaskEntity,TaskEntity> retrieves the Source Object value "item" and returns the TargetObject value "TmpTask2".
- SaveEntity<TaskEntity> saves the "TmpTask2" value.



Removed Case

The Removed case is detected with the following condition: **item.EntityState = Cmc.Core.EntityModel.EntityState.Removed**

In this case, the DeleteEntity<TaskEntity> activity is executed to remove the "item" value.

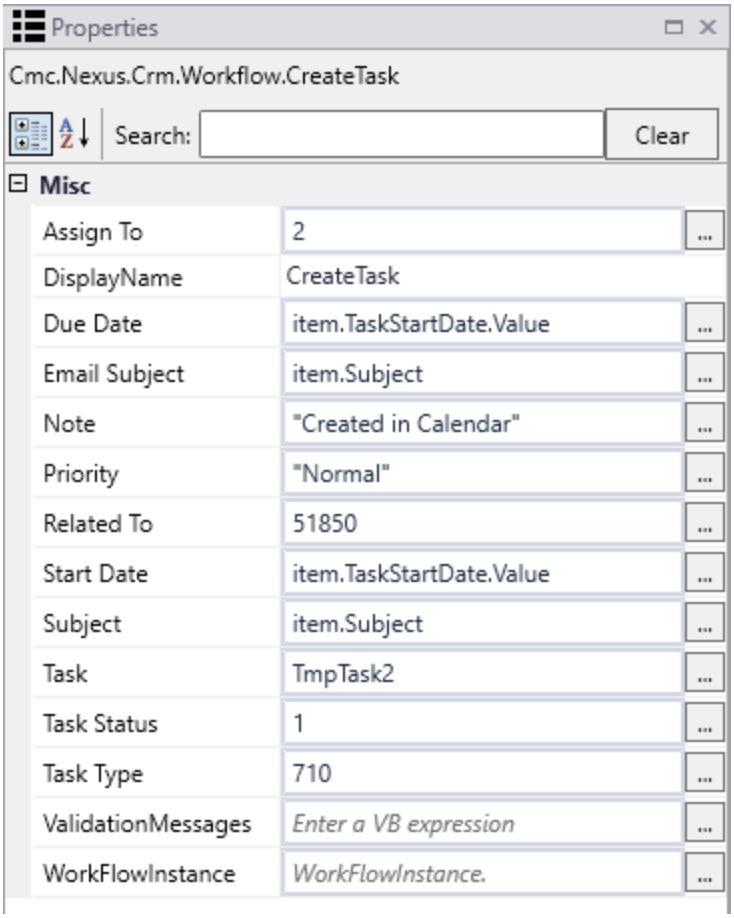


Added Case

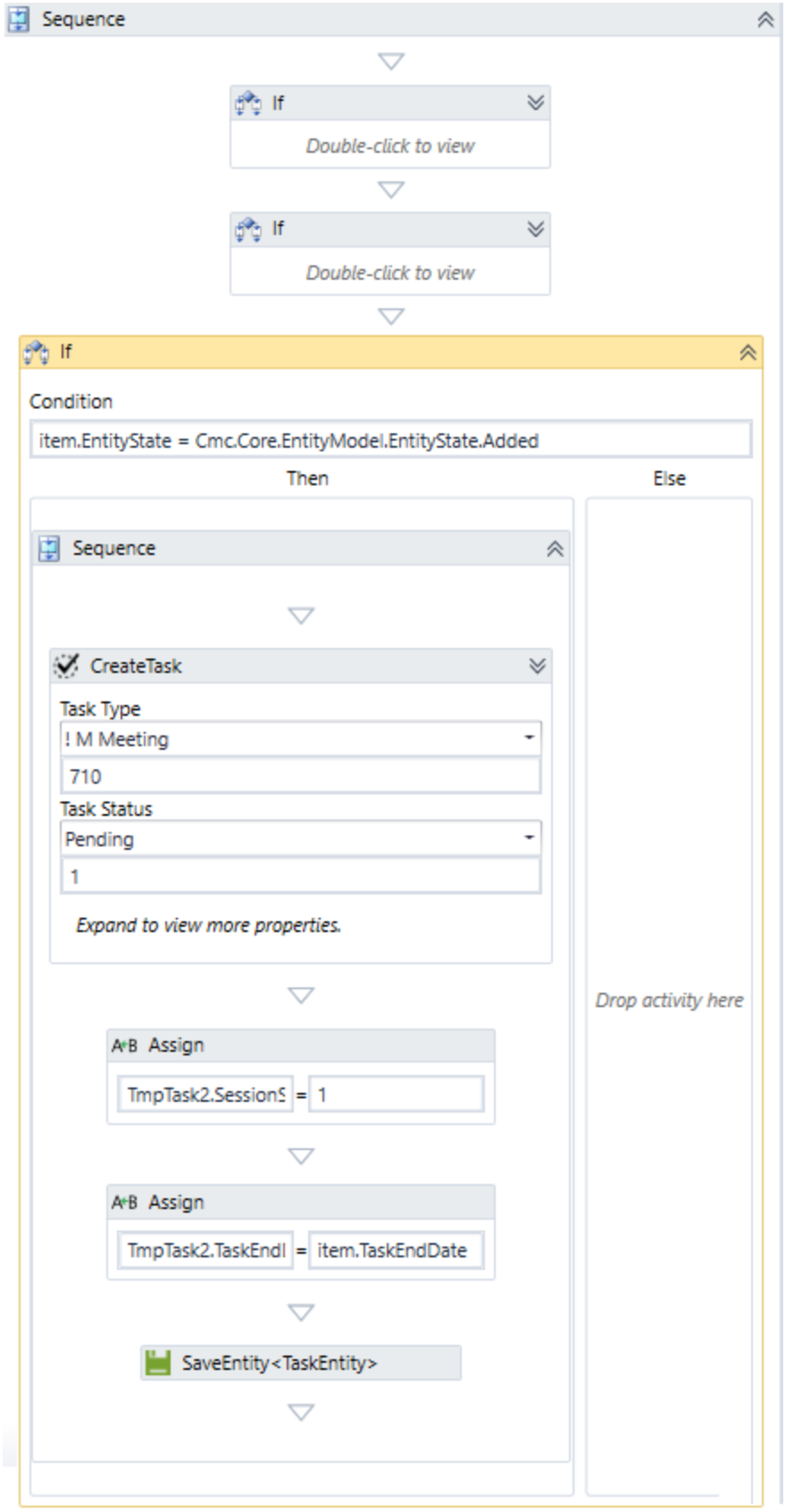
The Added case is detected with the following condition: **item.EntityState = Cmc.Core.EntityModel.EntityState.Added**

In this case, CreateTask, Assign, and SaveEntity activities are executed.

- CreateTask creates the out argument <TaskEntity> with a value of "TmpTask2".

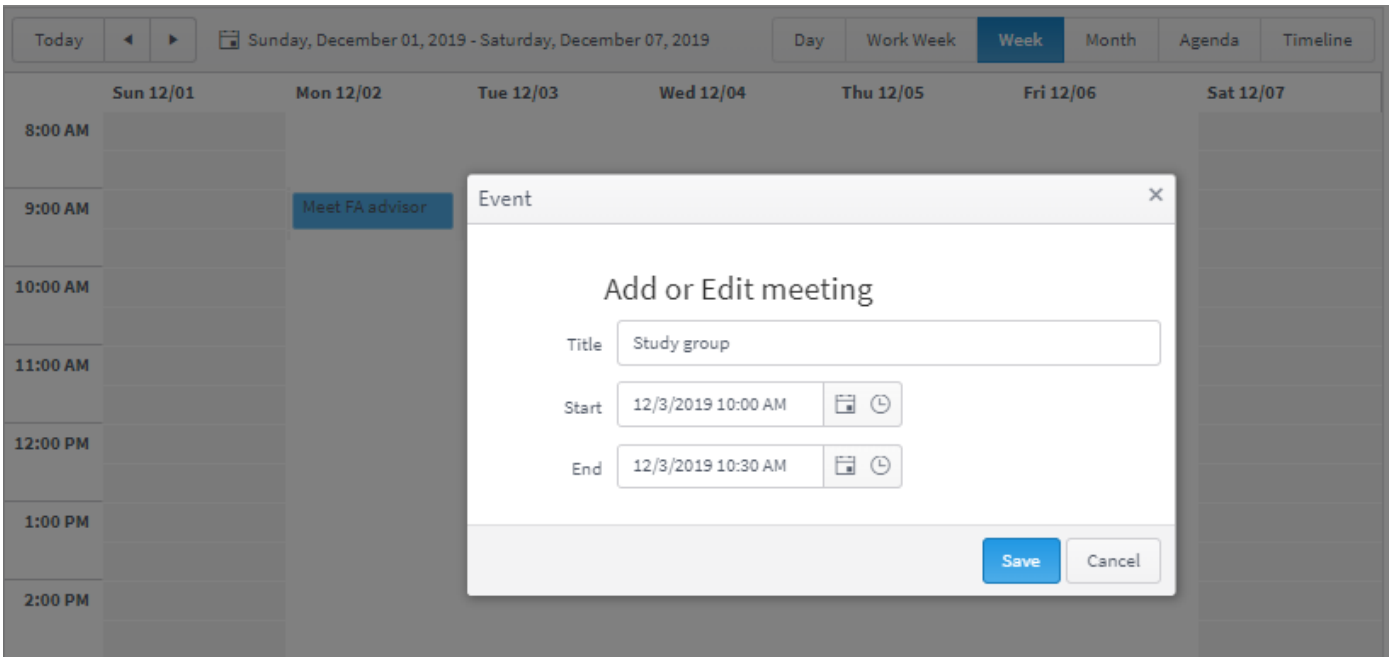


- The first Assign activity below CreateTask assigns a value of "1" to "TmpTask2.SessionSyCampusId".
- The second Assign activity assigns a value of "item.TaskEndDate" to "TmpTask2.TaskEndDate".
- SaveEntity<TaskEntity> saves the "TmpTask2" value.


Sequence
If
Double-click to view
If
Double-click to view
If
Condition
item.EntityState = Cmc.Core.EntityModel.EntityState.Added
Then
Else
Sequence
CreateTask
Task Type
! M Meeting
710
Task Status
Pending
1
Expand to view more properties.
Drop activity here
Assign
TmpTask2.SessionS = 1
Assign
TmpTask2.TaskEndI = item.TaskEndDate
SaveEntity<TaskEntity>

Rendered Form

The rendered form now enables users to add, modify, or delete events in the calendar.



The End time field must include both date and time, e.g., 2019-06-25 14:30:00.000 . Check the value in Generated Model for Debugging.

CAPTCHA

You can use the CAPTCHA component to add a challenge-response test to a form to determine whether the user is human. The CAPTCHA component links to the free Google reCAPTCHA service which supplies subscribing websites with site verification challenges.

When the CAPTCHA component is added to a form, the user must select a check box labeled "I'm not a robot" and solve a visual or auditory challenge. The user can select the challenge type by selecting the icon or the icon. The user must complete the selected challenge before submitting the form.

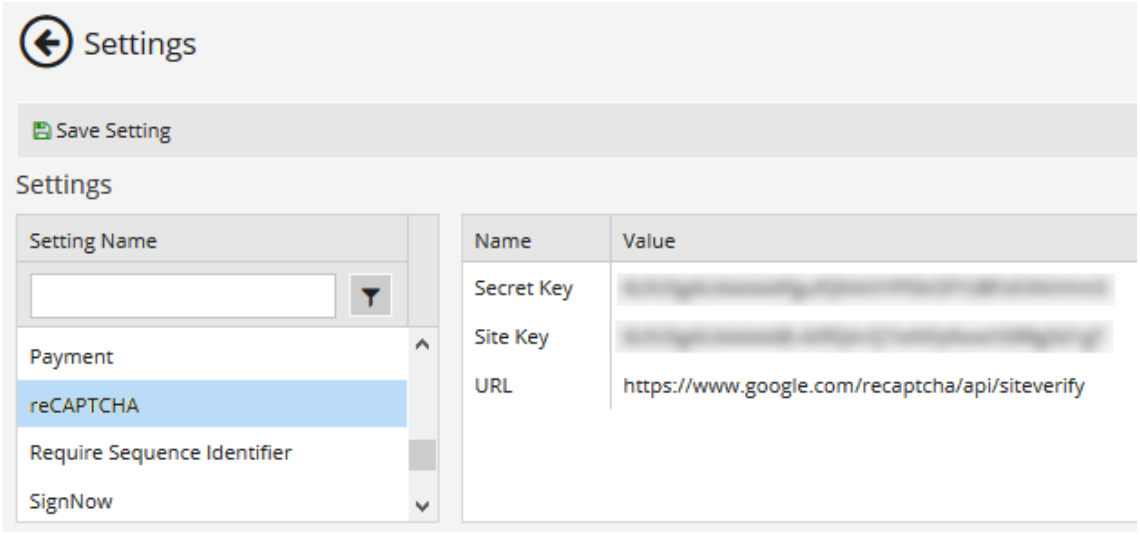
The CAPTCHA verification will expire after some time and the user will need to start over if this occurs. The user will be notified if the session expires.

If a server-side validation error occurs while the user is completing the form, the CAPTCHA component is reset, and the user is forced to resolve it.

Prerequisites

1. To use the Google reCAPTCHA service, sign up for an API key pair for your site. The key pair consists of a site key and secret key. The site key is used to display the widget on your site. The secret key authorizes communication between Forms Builder and the reCAPTCHA server to verify the user's response.
2. In the Forms Builder Settings workspace, select **reCAPTCHA** and populate the **Secret Key**, **Site Key**, and **URL**.

   You will need to re-enter the reCaptcha Site Key and Secret Key values after upgrading Forms Builder.



> Whenever you edit a field in a Settings panel with hidden fields (i.e., sensitive data that are removed from the client-side view like passwords, keys, and secrets), you must re-enter the hidden values, otherwise they will get cleared upon update and resave of all the Settings.

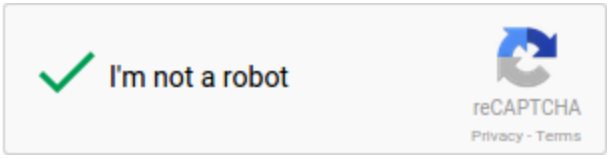
3. Select **Save Setting**.

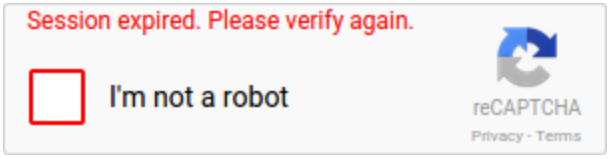
Properties

Control Property Settings

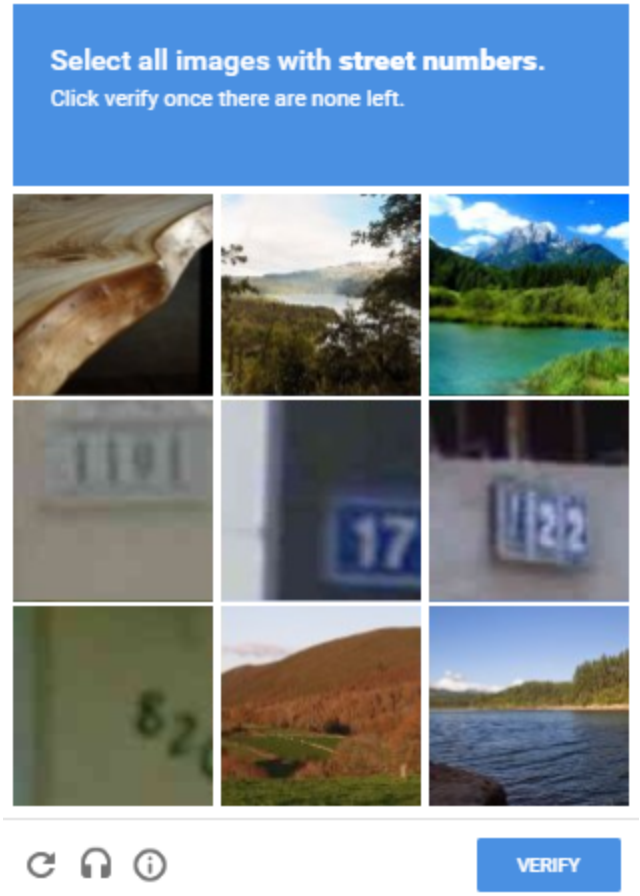
Control Type	CAPTCHA
Class	
Id	idde4a61df-4a63-967b-7dcf-6b35f10c8ed3

Rendered Component

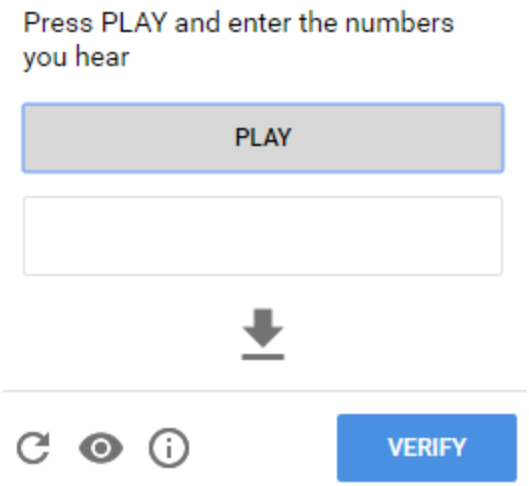




Visual challenge



Auditory challenge



- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.

- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.

Checkbox

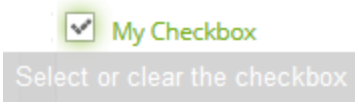
You can use the Checkbox component for binary choices like yes/no or true/false.

Properties

Control Property Settings

Control Type	Checkbox
Class	
Disabled	false
Id	bfec0f69-06ac-ed80-80c6-e8acb71f91d1
Label	My Checkbox
Model	vm.models.myCheckbox
Read-only	false
Tab Index	
Tooltip	Select or clear the checkbox
Tooltip Duration	750
Visible	true

Rendered Component



Workflow Argument

Name	Direction	Argument type	Default value
myCheckbox	In/Out	Boolean	*Default value not supported*

For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Disabled** sets a control to disabled.

- Must be true or false, or a binding beginning with "vm.models.".
- A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
- If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
  - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
  - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
  - Binding is not supported for this property.
- **Label** is the value displayed in the label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.
  - The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
  - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
  - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
  - The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
  - If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

  If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

Construction of the model in the workflow is done by assigning data from a provider.

- **Read-only** makes the control read-only. It is set to false by default. If you want the component to be read-only, set the property value to true. It is typically used for an input box.
    - Can be bound to a workflow argument or another control's value.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
    - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
    - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
    - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

      The tab index value should not match another control's tab index.
    - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.
- **Tooltip** is the value to display when hovering over the control's label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
    - When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

Credit Card Payment

You can use the Credit Card Payment component to build a form that supports credit card processing functionality based on the Payment Card Industry Data Security Standard (PCI DSS). A requirement for PCI compliance is that credit card information such as card number, expiration date, and CVV (Card Verification Value) is stored securely with a PCI compliant hosting provider (i.e., not stored in the institution's database) and that the user interface of the application, (i.e., Forms Builder) does not capture credit card information. Payflow Pro (by PayPal), ACI, IATS and TouchNet are the currently supported payment gateways for credit card payment processing with Forms Builder.

The Credit Card Payment component is used to redirect the flow of the sequence to the hosted payment form. Data that will be defaulted on the hosted form can also be included within the property settings of this component. The user enters the credit card information on the payment processor's webpage. The payment is processed and a response containing the transaction identifier is sent back to Forms Builder.

The Credit Card Payment component requires specific configuration settings in the Settings workspace of Forms Builder and a workflow activity that captures the transaction identifier returned from the payment processor's webpage. You also need to set up connectivity to the payment gateways and configure the hosted payment page on the PayPal or ACI websites.

- Depending on the payment gateway used, refer to the following topics:
  - Payment Processing with PayPal
  - Payment Processing with ACI
  - Payment Processing with IATS
  - Payment Processing with TouchNet
- For details about the required workflow activity, refer to VerifyCardPayment.
- For an example of a form sequence with credit card processing functionality, refer to Credit Card Payment Form.

You can use only **one** Credit Card Payment component (i.e., one transaction) per sequence.

Always place the Credit Card Payment component on a separate form.

If the form contains any other fields, these fields should be read-only. Changes to those fields will not be retained when the user returns to the form after making the payment.

Credit Card Payment forms using PayPal, ACI, IATS, or TouchNet payment gateways will not be translated as these vendors do not support localization at this time.

Properties

When you build a form sequence for credit card payment, ensure that you place the Credit Card Payment component on a separate form, i.e., create one form to gather the payment data that will prepopulate the payment site form (e.g., name, address, state, country, etc.) and another form for the Credit Card Payment component.

Communication to the payment site occurs on the form with the Credit Card Payment component and prior to binding any data entered on same form. Therefore, the input for name, amount, etc. must be on a previous form.

Notes:

- The bindings in the properties for the Credit Card Payment component correspond to arguments that need to be created in the workflow.
- If optional fields are not specified in Forms Builder, these fields will not be prepopulated on the payment site. The user will have to fill in the fields if the fields are configured as required on the payment site.
- For Payment Country and Payment State to be prepopulated, the names of the country/state in the institution's database need to match the lists in PayPal. If the names of the country/state do not match, the user will have to select the values from the drop-down lists on the payment form if the fields are required for payment processing.

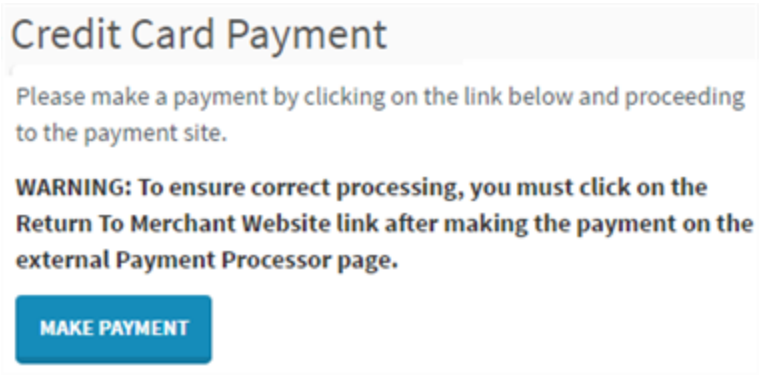
  Example: Country must be "United States of America", not "USA".

  Known Limitation: The Credit Card Payment component currently does not pass Payment Country values other than "Unites States of America" to the PayPal site. If any other Payment Country values are specified, either the Country drop-down list will not be displayed, or the user will have to select the country in the payment form on the PayPal site.
- In Forms Builder 3.5, the code for the Credit Card Payment component was completely revised to provide new functionality such as the "Payment Comment" property and enhanced validation of properties/bindings. When you drag the Credit Card Payment into the Layout pane, the underlying functionality will be that of the new component. In existing forms that already include the Credit Card Payment, the component is automatically updated to the correct new values and can be saved immediately.

Control Property Settings

Control Type	Credit Card Payment
Class	
Id	idd94b9bcf-ce30-ef3c-ed1e-d7cb7c142245
Payment Address	{{vm.models.studentEntity.StreetAddr...
Payment Amount	{{vm.models.depositEntity.Amount}}
Payment City	{{vm.models.studentEntity.City}}
Payment Comment	
Payment Country	United States of America
Payment Disabled	false
Payment Disabled Reason	
Payment Email	{{vm.models.studentEntity.EmailAddr...
Payment Error Message	Must make a payment to continue.
Payment Firstname	{{vm.models.studentEntity.FirstName}}
Payment Instructions	Please make a payment by clicking on the link below and proceeding to the payment site.
Payment Lastname	{{vm.models.studentEntity.LastName}}
Payment Link Class	btn btn-primary
Payment Link Text	Make Payment
Payment State	{{vm.models.studentEntity.State}}
Payment Warning Message	WARNING: To ensure correct processing, you must click on the Return To Merchant Website link after making the payment on the external Payment Processor page.
Payment Zip	{{vm.models.studentEntity.PostalCode}}

Rendered Component



Workflow Arguments

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event	In/Out	ConstructedEvent	*Default value not supported*
studentEntity	In/Out	StudentEntity	*Default value not supported*
depositEntity	In/Out	DepositEntity	*Default value not supported*
countryName	In/Out	String	*Default value not supported*

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Payment Address** is optional. If you want to prepopulate this field on the hosted payment page, enter the binding corresponding to the card holder address. Example: `{{vm.models.address}}`
- **Payment Amount** is required. Enter the binding corresponding to the card payment amount.

  Example

  `{{vm.models.studentEntity.myAmount}}`

This property must be bound and does not take a string.

- **Payment City** is optional. Enter binding for city corresponding to the card holder address.

  Example

  `{{vm.models.myPaymentCity}}`

  This property must be a binding if set

- **Payment Comment** is optional. Enter a string or a binding for a comment which can appear in reports. This is normally set in a workflow by binding. If it is bound, it must begin with `{{vm.models.` and end with `}}`.

  Example

  You want to pass a comment indicating that the credit card payment is for a student identified by name and student ID,

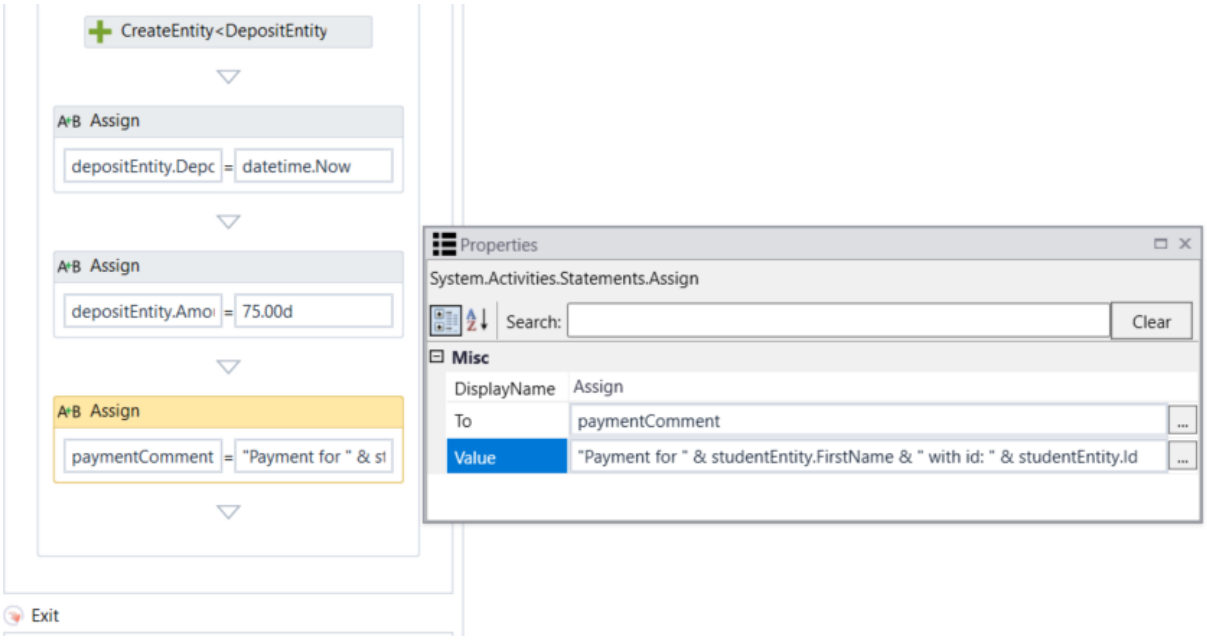
  1. In the Payment Comment property, specify the following value:

     `{{vm.models.paymentComment}}`

  2. In the workflow for the sequence on the form prior to the form with the Credit Card Payment component, create a `paymentComment` argument of type String,

     Assign a value to the argument using an expression to identify an authenticated student:

     `"Payment for " & studentEntity.FirstName & " with id: " & studentEntity.Id`



  3. Save the workflow and render the sequence.
  4. Complete the sequence and make a payment using a test account.

5. Log in to the payment provider's site as administrator and locate the transaction.
6. Review the transaction report and verify that the paymentComment with the student's name and ID is included in the report.

- **Payment Country** is optional. Enter a string or a binding for country corresponding to the card holder address.

  Example

  `{{vm.models.myCountry}}` which can be set in the workflow. This property might be required to be a specific string for the payment provider. Refer to Notes:.

- **Payment Disabled** sets the payment button or link to disabled. It is set to false by default.
  - Must be true or false (default), or a binding beginning with "vm.models.".
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - This property can be dynamically bound.
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<,!-!=,==,>=,<=).
  - If comparing to a string, it must be in single quotes. (true or false must be all lowercase)

  Use case

  A form accepts credit card payments only if the user resides in the United States. The Payment Disabled value is set to (vm.models.studentEntity.CountryId !=7), where CountryId = 7 is United States.

  The workflow for the sequence has a LookupReferenceItem activity with Reference Item Type=Country and ReferenceItem Id="studentEntity.CountryId.Value. If the user's studentEntity.CountryId.Value is not 7, the credit card payment option is disabled.

- **Payment Disabled Reason** - Enter HTML or a binding which will be shown when "Payment Disabled" is true. It is empty by default.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myDisableReason}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
  - HTML is supported.
    - If you want translatable HTML, make sure the element that contains the text has the "translate" attribute added. This will be picked up when the POT file is generated and will be available for translation.

- A string with no HTML does not need a "translate" attribute as it will be embedded in a div with the attribute.

Use case

A form accepts credit card payments only if the user resides in the United States. The Payment Disabled value is set to (vm.models.studentEntity.CountryId !=7), where CountryId = 7 is United States, and the Payment Disabled Reason is set to <h4 style="color:red;">**Make Payment Disabled - Cannot accept foreign credit cards.
 Contact your administrator for alternate payment options.</h4>

The workflow for the sequence has a LookupReferenceItem activity with Reference Item Type=Country and ReferenceItem Id="studentEntity.CountryId.Value. If the user's studentEntity.CountryId.Value is not 7, the credit card payment option is disabled, and the following text is displayed above the "Make Payment" button:

**Make Payment Disabled - Cannot accept foreign credit cards.
Contact your administrator for alternate payment options.

- **Payment Email** is optional. Enter binding corresponding to the card holder email address.

  Example

  `{{vm.models.myEntity.email}}`. This property must be a binding if set.

- **Payment Error Message** is optional. Enter the error message that is displayed if the user selects Next without paying or when an error occurs while receiving the data from the card processor.

  Example

  Payment could not be processed. Please contact customer service.

- **Payment Firstname** is required. Enter the binding corresponding to the card holder first name.

  Example

  `{{vm.models.studentEntity.cardHolderFirstName}}`. This property must be bound and does not take a string.

- **Payment Instructions** are optional. Enter the payment instructions to be displayed in the Credit Card Payment component above the Payment Link.

  Example

  Please make a payment by clicking on the link below and proceeding to the payment site. After completing the payment, click the Return to Merchant Website link.

  You can specify a variable in the payment instructions using the syntax `{{vm.models.amount}}`. This will display the exact amount to be paid. For example: *Please make a payment for ${{vm.models.amount}} by clicking on the link below and proceeding to the payment site. Click on the Return to Merchant Website link after completing the payment.*

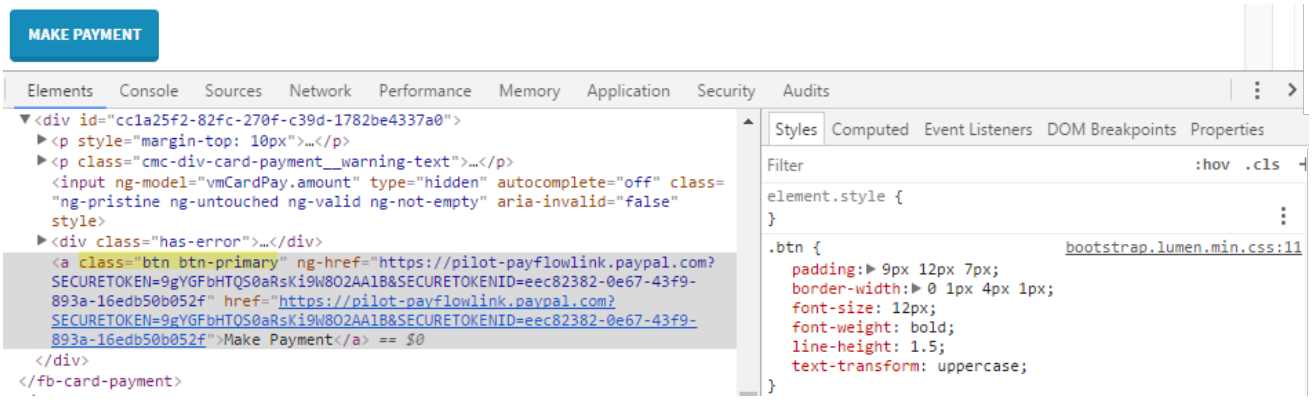
- **Payment Lastname** is required. Enter the binding corresponding to the card holder last name.

Example

`{{vm.models.studentEntity.LastName}}`. This property must be bound and does not take a string

- **Payment Link Class** is optional. Enter the CSS class (or space separated classes) for the payment link URL. It must be defined in a Renderer CSS file. Default: `btn btn-primary`

  The class `btn btn-primary` is a Bootstrap class that renders the Payment Link as a blue button with white text.



  You can customize the Payment Link Class by selecting a different Bootstrap class or adding your own class in a custom CSS.

- **Payment Link Text** is required. Enter the display text for the link to the payment site.

  Example

  Make Payment. The default Payment Link Text is the label on the button.

- **Payment State**is optional. Enter the binding for state corresponding to the card holder address.

  Example

  `{{vm.models.myState}}`. This property must be a binding if set. Refer to Notes:.

- **Payment Warning** is optional. Enter a warning message that will be displayed in the Credit Card Payment component above the Payment Link.

  Default: *WARNING: To ensure correct processing, you must click on the Return To Merchant Website link after making the payment on the external Payment Processor page.*

  > ⚠ If the user does not select a link shown on the payment processor site to return to the Forms Builder site, Forms Builder will have no record of the payment. Hence, the *Payment Warning* message has to make this clear. If a user does not select the return link and accidentally closes the window, the payment can only be manually verified through Reports on the payment processor's website.

- **Payment Zip** is required or optional depending on how the payment flow provider is configured. If provided, it must be a binding.

Example

```
{{vm.models.myZip}}.
```

Payment Processing with PayPal

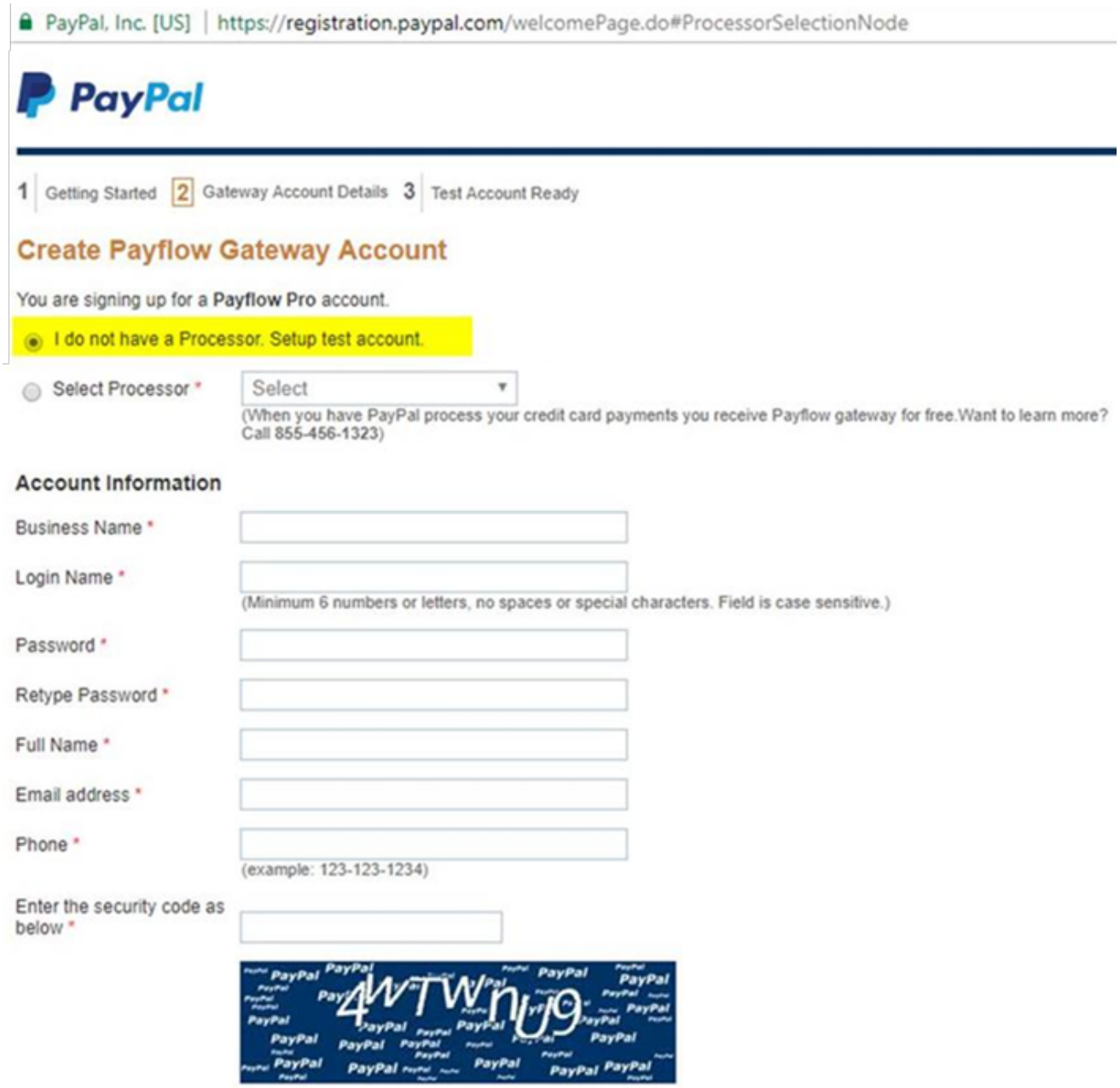
This help topic contains several links to the paypal.com website. The links may change without notice. If a link is broken, please search the PayPal documentation or contact PayPal for the latest information.

Create a Payflow Pro Account

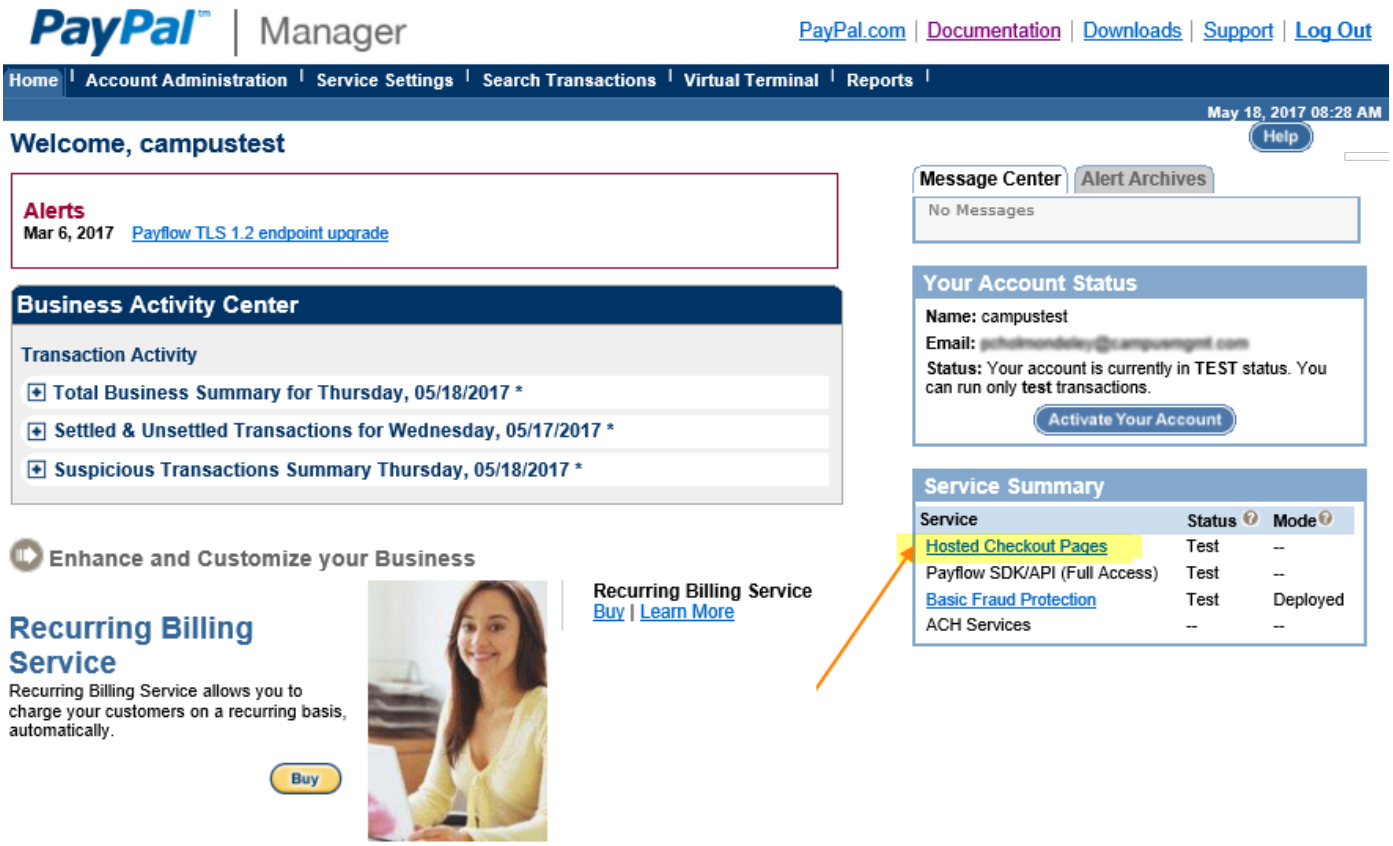
Set up an account (in test or live mode) at https://manager.paypal.com/.

1. Select the link to set up a **new account**.
2. On the next page, select **Get Started Today**.
3. Select the **PayFlow Pro** option. The *Create Payflow Gateway Account* page is displayed.

4. Select the option **I do not have a Processor. Setup test account** and follow the instructions.



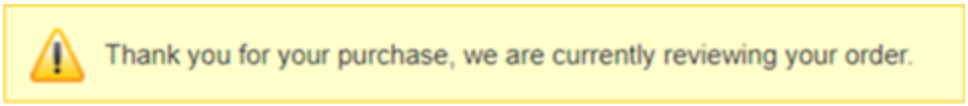
If you already have a PayFlow Pro account, ensure that Hosted Checkout Pages is enabled when you log in to https://manager.paypal.com/.



Contact a PayPal representative if you have any issues with doing this.

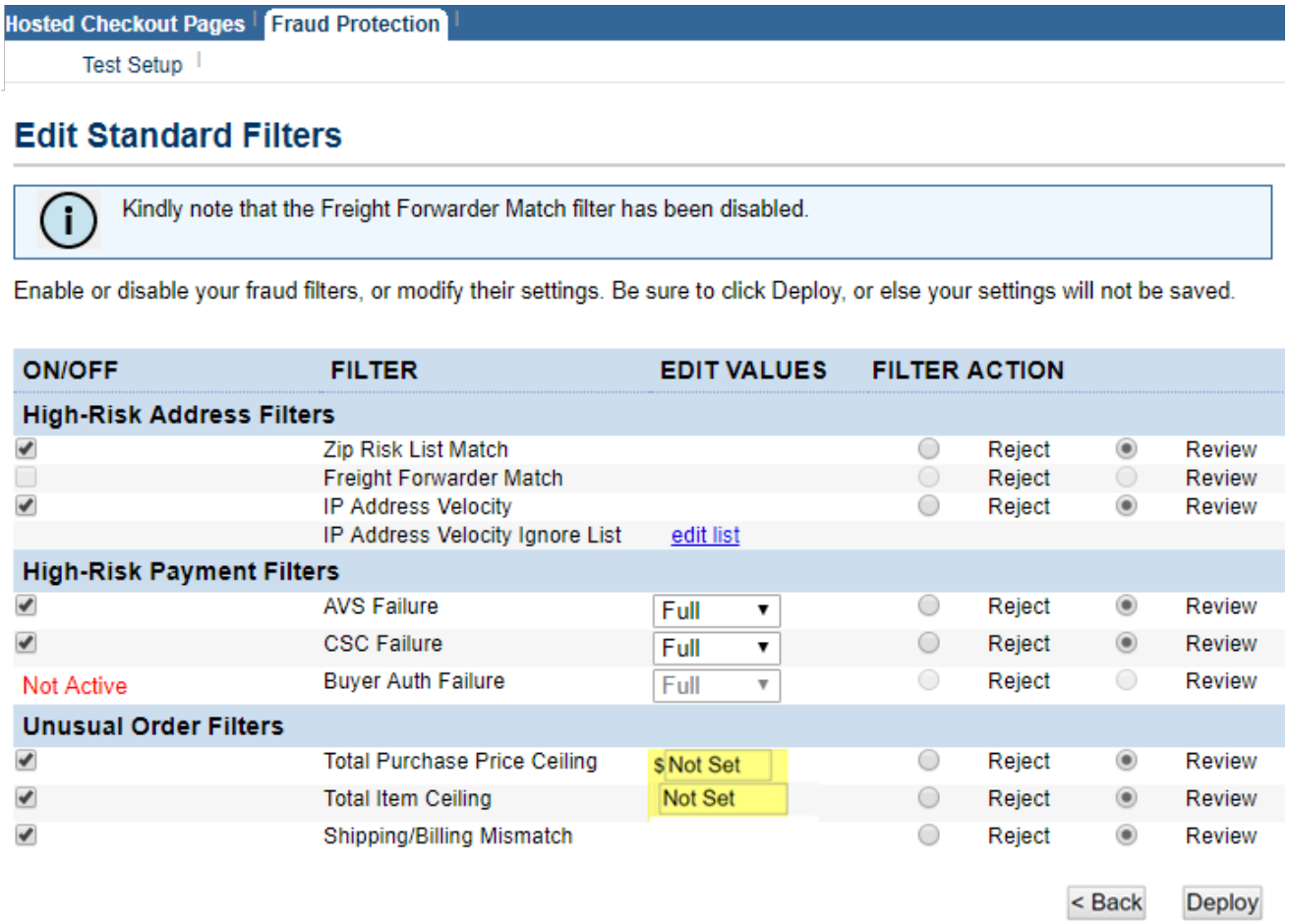
Set Fraud Protections Filters for Your Account

When you create a test account, all the fraud filters are turned on by default. When the account is used in Forms Builder, transactions will fail with the following message when users try to enter their card information on the PayPal checkout page:

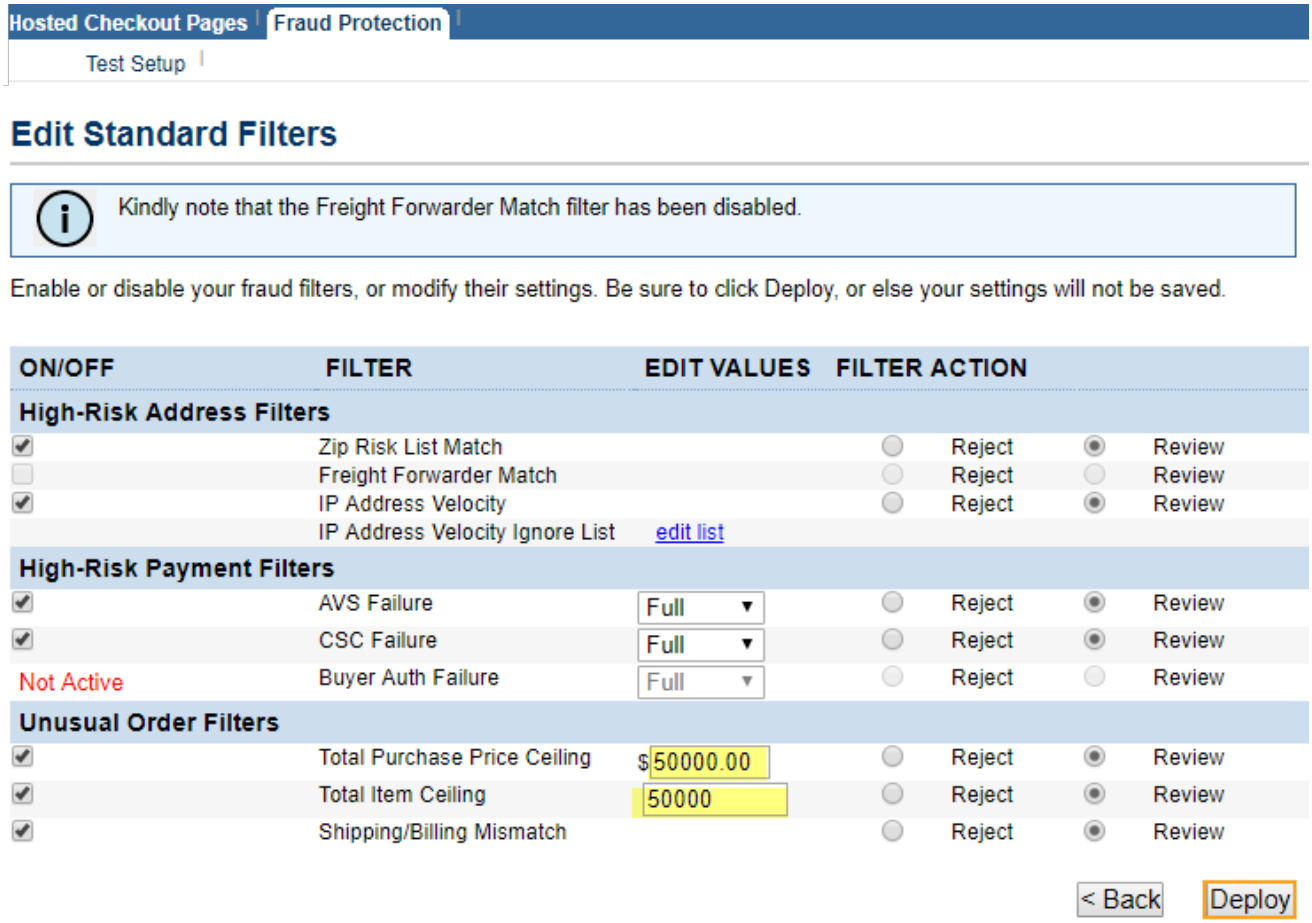


This is because the Total Purchase Price ceiling and Total Item Ceiling filters are enabled, but the amount is not set. The filters kick in and the transactions will need to be reviewed and manually approved. To fix this for future transactions, perform the following steps:

1. Go to your PayPal test account and navigate to **Service Settings > Fraud Protection > Test Setup**. The Edit Standard Filters page is displayed.

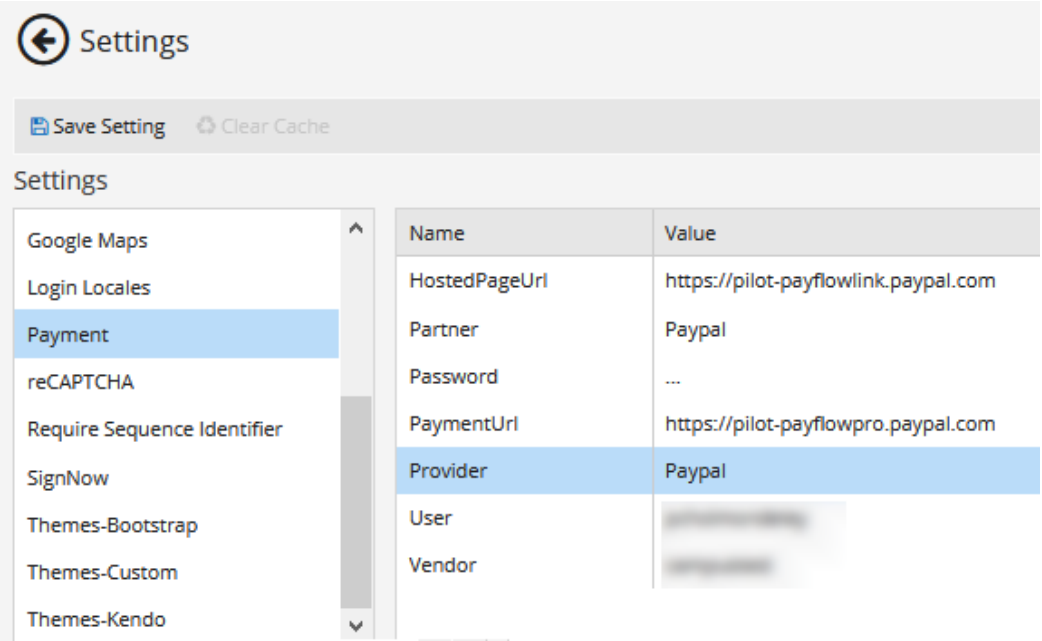


2. Enter some $ values in the **Total Purchase Price Ceiling** and **Total Item Ceiling** filters and select **Deploy**.



Populate Forms Builder Settings

In the Forms Builder Settings workspace, populate the HostedPageUrl, Partner, Password, PaymentUrl, User, and Vendor fields.



Provide the **Password**, **User**, and **Vendor** values. The remaining fields are preset. If you change the credit card payment provider later, you must update the URLs, remove the Partner entry, and update the Password, User, and Vendor. The values specified in the Settings workspace are saved to the web.config file for Form Designer.

Name	Value
HostedPageUrl	URL that identifies the location of the payment processor's page: https://pilot-payflowlink.paypal.com ⚠ The prefix **pilot-** indicates that this URL should be used during testing. When you move to a live environment, remove "pilot-" from the URL. Make sure to use the right type of Payflow account (test/live) in your environments (test/live). Do not use a live account in the test environment and vice versa. The URLs should be with/without "pilot-" as appropriate.
Partner	Partner for your PayPal Manager account. It is the same field you use when you log in to https://manager.paypal.com/.
Password	Specify the Password (case sensitive) used to log in to https://manager.paypal.com/.
PaymentUrl	URL that identifies the payment page in live or test mode: ◦ Test mode: https://pilot-payflowpro.paypal.com ◦ Live mode: https://payflowpro.paypal.com
Provider	Name of the credit card payment provider: PayPal.
User	Specify the User (case sensitive) used to log in to https://manager.paypal.com/. It can be the same as the Vendor. The User is optional, but PayPal highly recommends using it. Create a User against which API calls can be made. Create a User from the Account Administration > Manage Users > Add User page. Refer to PayPal Manager documentation for more information.

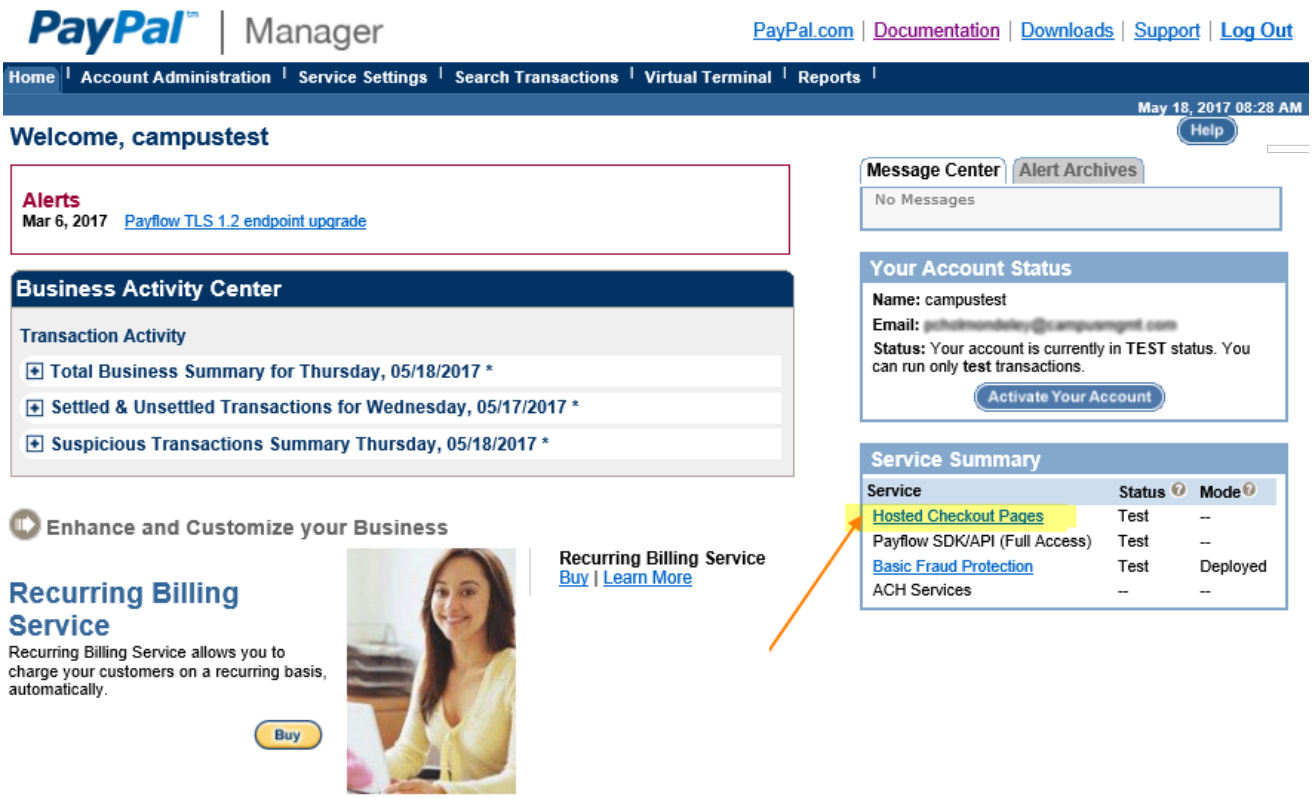
Name	Value
uPaySiteId	Indicates the URL to the TouchNet uPay page. TouchNet gateway provider URLs are not populated initially. You need to populate the URLs for TouchNet manually since the URLs are unique for each client's account. This field (numeric value) is MANDATORY when you select TouchNet as the payment provider. For other payment providers such as PayPal, ACI and IATS, uPaySiteId is not mandatory. The system will not consider the uPaySiteId value, even if provided.
Vendor	Specify the Vendor (case sensitive) used to log in to https://manager.paypal.com/.

Whenever you edit a field in a Settings panel with hidden fields (i.e., sensitive data that are removed from the client-side view like passwords, keys, and secrets), you must re-enter the hidden values, otherwise they will get cleared upon update and resave of all the Settings.

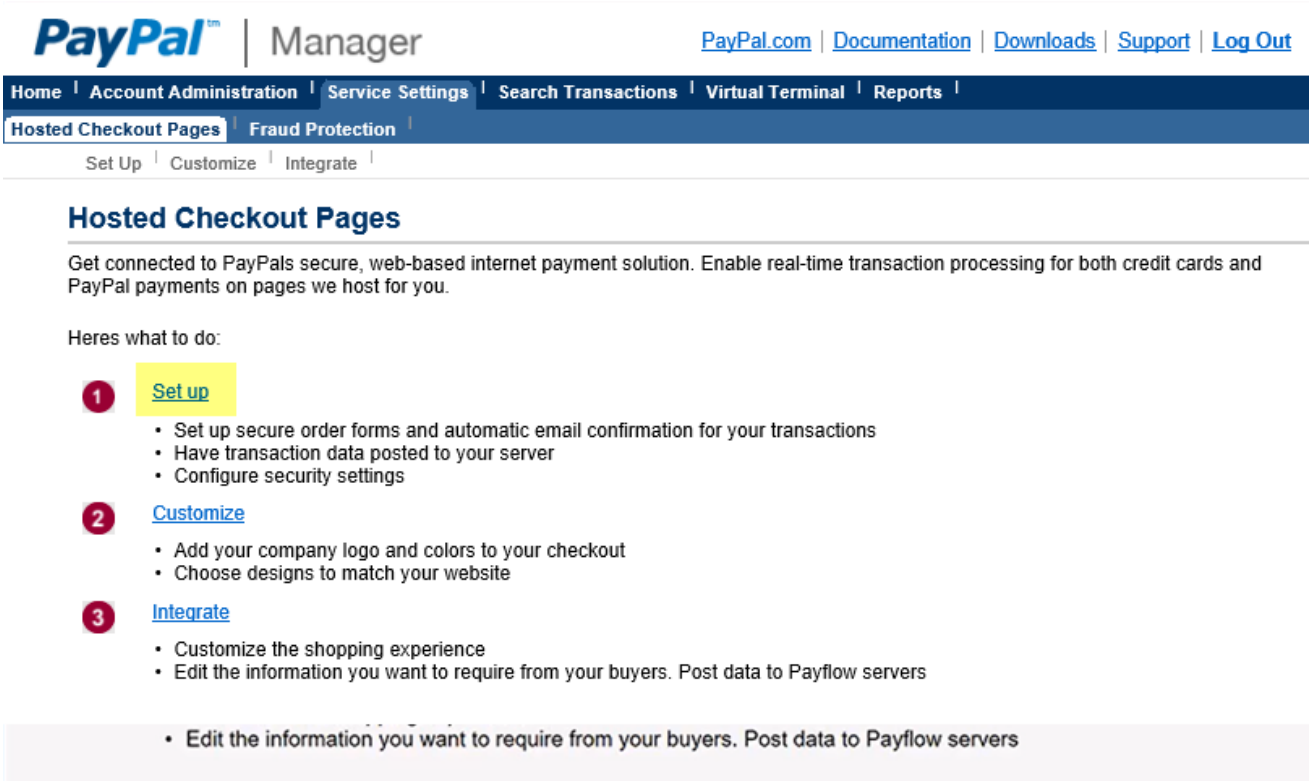
Configure the Hosted Payment Page

The fields displayed on the hosted payment page are configurable. To configure the fields, perform the following steps:

1. Log in to your account at https://manager.paypal.com/.
2. On the Home page, select the **Hosted Checkout Pages** link.

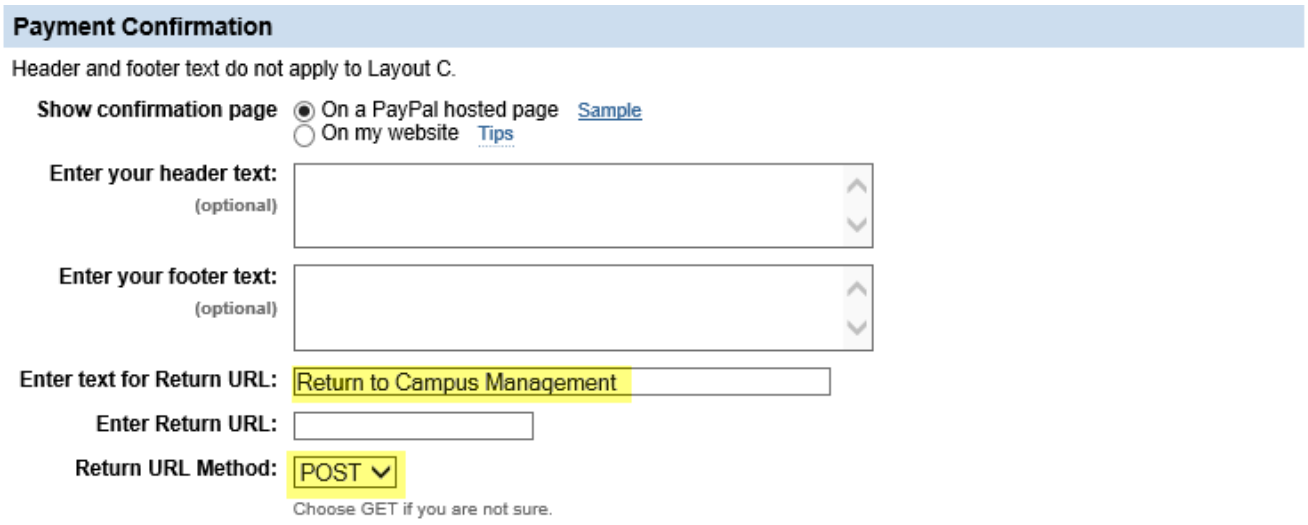


3. On the Server Settings page, select **Setup**.

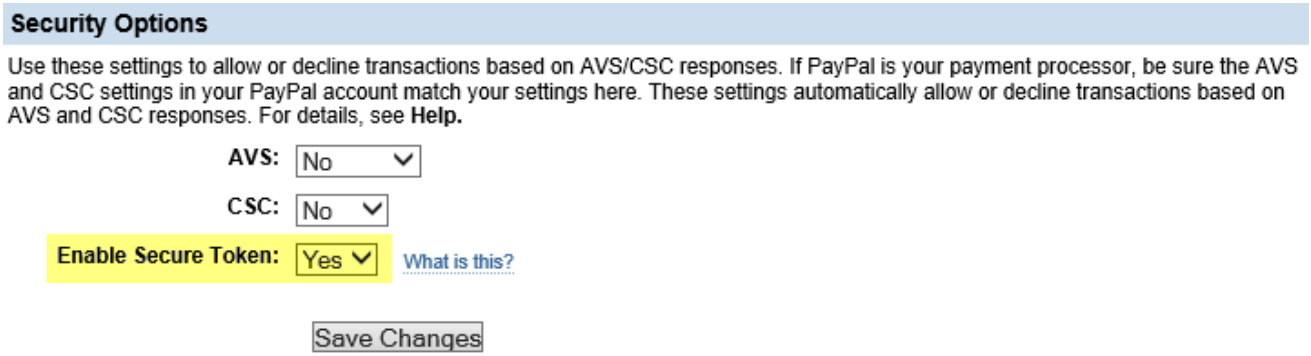


4. On the Service Settings page > Set Up tab, select the billing information settings that are appropriate for your institution. Specify which fields are required or optional, which fields are editable, and so on. Refer to the PayPal Manager documentation for more details about these settings.

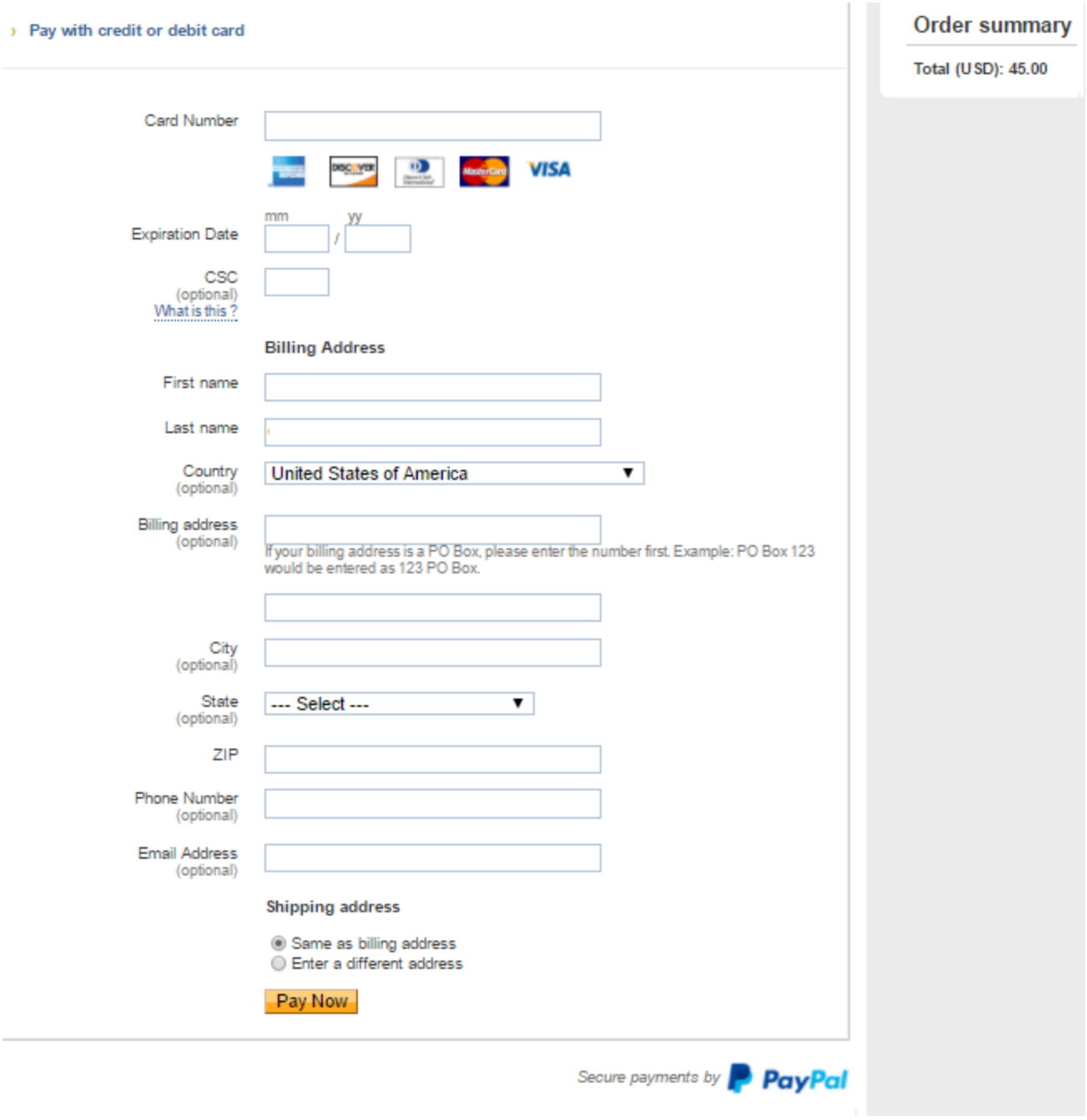
> ⚠ In the Payment Information section of the Set Up tab, specify the text for the **Return URL** link and ensure that the **Return URL Method** is set to **POST** (default: GET). Without this setting, the payment process with the Credit Card Payment component and VerifyCardPayment activity will not work.



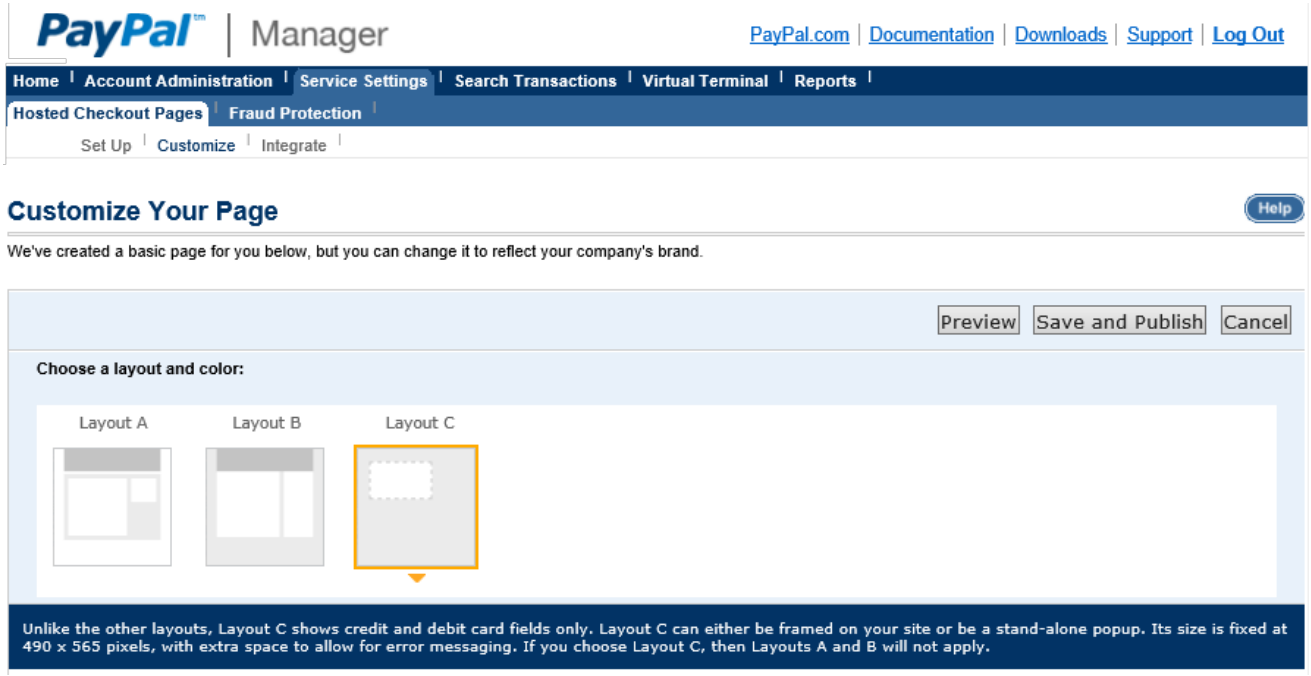
5. In the Security Options section, select **Yes** to **Enable Secure Token**.



6. On the Service Settings > Customize tab, select either **Layout A** or **Layout B** for the hosted payment page.
   Example: Layout A

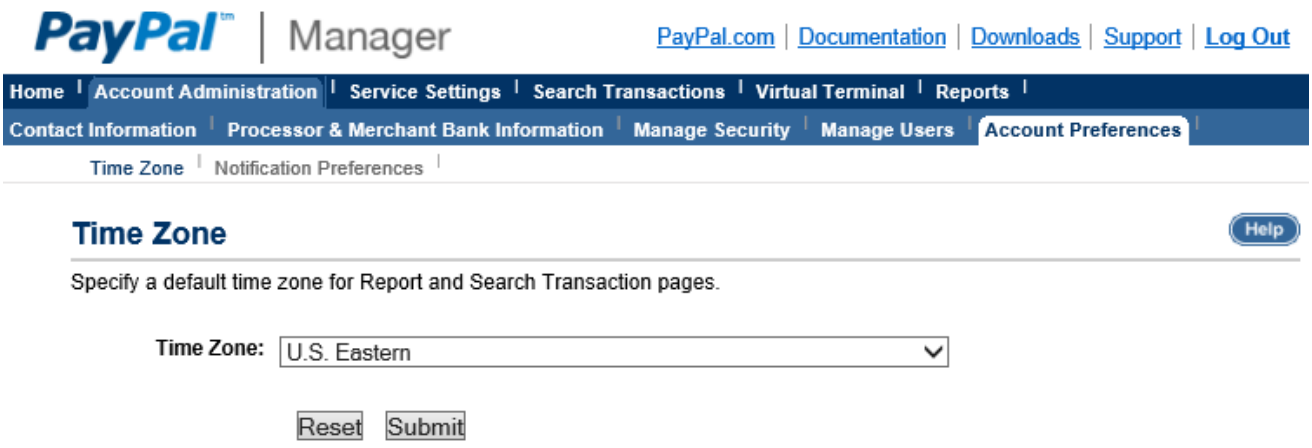


Layout C cannot be used with Forms Builder sequences because it does not provide a Return URL link to return to the Forms Builder site after the payment is made.



Refer to the PayPal Manager documentation for more details: https://developer-.paypal.com/docs/classic/payflow/gs_ppa_hosted_pages/.

7. On the Account Administration > Account Preferences tab, select the **Time Zone** of the Forms Builder server. The VerifyCardPayment activity uses the time stamp returned by the PayPal server and compares it to the current time stamps in the workflow. If the time stamps don't match, the activity returns a negative result and the payment transaction is not verified.



Card Numbers for Testing

To test your form, refer to the following website to obtain credit card numbers for testing:

https://developer.paypal.com/docs/classic/payflow/integration-guide/?mark=test%20card%20numbers#credit-card-numbers-for-testing

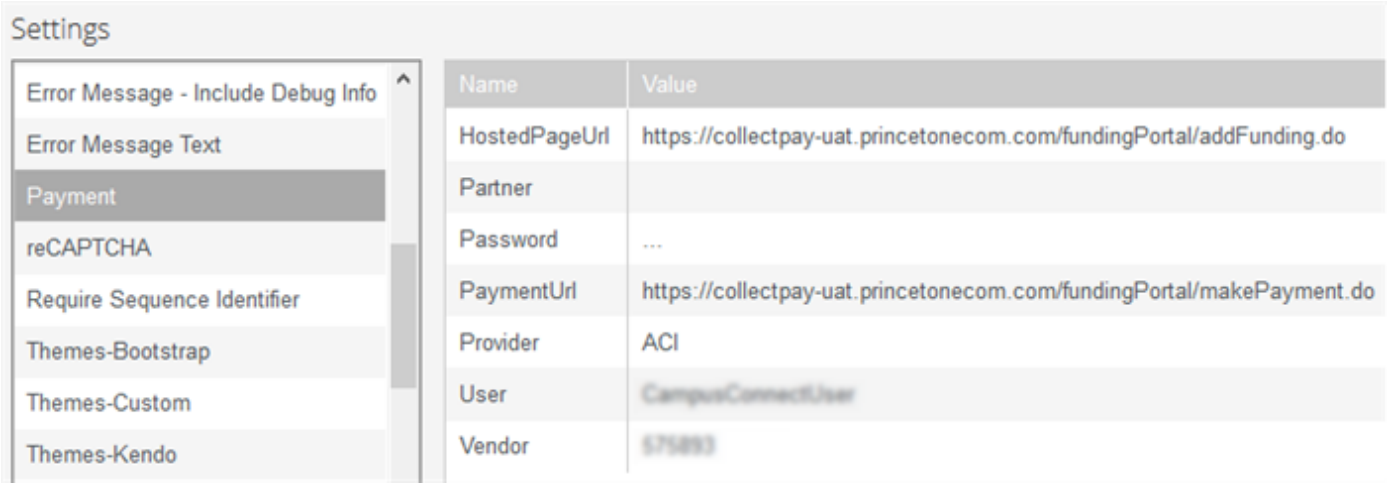
Payment Processing with ACI

The Forms Builder settings below are required to use ACI as the vendor for credit card payment processing. For additional setup requirements, please contact your Anthology Inc. account representative.

Anthology Inc. products support only the “Funding Portal” product from ACI.

Populate Forms Builder Settings

In the Forms Builder Settings workspace, populate the **Payment** setting fields.



Provide the **Password**, **User**, and **Vendor** values. The remaining fields are preset. If you change the credit card payment provider later, you must update the URLs, remove the Partner entry, and update the Password, User, and Vendor. The values specified in the Settings workspace are saved to the web.config file for Form Designer.

Name	Value
HostedPageUrl	URL that identifies the location of the payment processor's page: https://collectpay-uat.princetonecom.com/fundingPortal/addFunding.do
Partner	Partner for your ACI Manager account.
Password	Specify the Password (case sensitive) used to log in to your ACI account.
PaymentUrl	Url that identifies the payment page: https://collectpay-uat.princetonecom.com/fundingPortal/makePayment.do
Provider	Name of the credit card payment provider: ACI.
User	Specify the User (case sensitive) used to log in to your ACI account. It can be the same as the Vendor.

Name	Value
uPaySiteId	Indicates the URL to the TouchNet uPay page. TouchNet gateway provider URLs are not populated initially. You need to populate the URLs for TouchNet manually since the URLs are unique for each client's account. This field (numeric value) is MANDATORY when you select TouchNet as the payment provider. For other payment providers such as PayPal, ACI and IATS, uPaySiteId is not mandatory. The system will not consider the uPaySiteId value, even if provided.
Vendor	Specify the Vendor (case sensitive) used to log in to your ACI account.

⚠ Whenever you edit a field in a Settings panel with hidden fields (i.e., sensitive data that are removed from the client-side view like passwords, keys, and secrets), you must re-enter the hidden values, otherwise they will get cleared upon update and resave of all the Settings.

Configure the Hosted Payment Page

The default ACI payment page can be customized as needed. Please contact ACI (www.aciworldwide.com) to request changes to the payment page.

The user information (e.g., name and address) is not carried forward to the ACI payment site. Users will have to re-enter this data on the payment site.

Debug Information

Credit card payments using ACI generate a random 6-digit Correlation Id for each Transaction Id. The Transaction Id and Correlation Id are written to the Forms Builder Renderer log when using debug mode, for example:

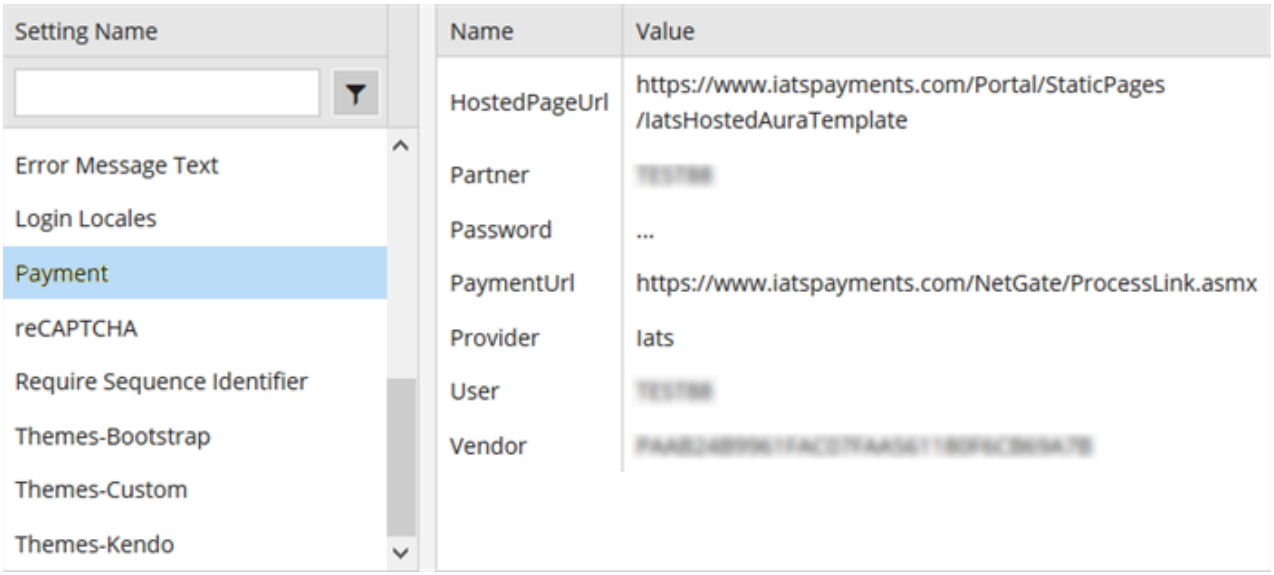
```
2018-11-26 13:54:56.4739  61 Debug          Cmc.Nexus.FormsBuilder.EventHandlers.PaymentEventHandlers
verifyPaymentRequest:
{
  "PaymentProviderInfo": {
    "Partner": "",
    "MerchantCode": "......",
    "UserName": ".........",
    "Password": ".........",
    "PaymentGatewayUrl": "https://collectpay-uat.princetonecom.com/fundingPortal/makePayment.do",
    "HostedPageUrl": "https://collectpay-uat.princetonecom.com/fundingPortal/addFunding.do "
  },
  "TransactionId": "9786473",
  "CorrelationId": "750009"
}
```

Payment Processing with IATS

The Forms Builder settings below are required to use IATS as a vendor for credit card payment processing. For additional setup requirements, please contact your Anthology Inc. account representative.

Populate Forms Builder Settings

In the Forms Builder Settings workspace, populate the **Payment** setting fields.



Provide the **Password**, **User**, and **Vendor** values. The remaining fields are preset. If you change the credit card payment provider later, you must update the URLs, remove the Partner entry, and update the Password, User, and Vendor. The values specified in the Settings workspace are saved to the web.config file for Form Designer.

Name	Value
HostedPageUrl	URL that identifies the location of the payment processor's page: https://www.iatspayments.com/Portal/StaticPages/IatsHostedAuraTemplate
Partner	Partner for your IATS account.
Password	Specify the Password (case sensitive) used to log in to your IATS account.
PaymentUrl	Url that identifies the payment page:
Provider	Name of the credit card payment provider: Iats.
User	Specify the User (case sensitive) used to log in to your IATS account. It can be the same as the Vendor.

Name	Value
uPaySiteId	Indicates the URL to the TouchNet uPay page. TouchNet gateway provider URLs are not populated initially. You need to populate the URLs for TouchNet manually since the URLs are unique for each client's account. This field (numeric value) is MANDATORY when you select TouchNet as the payment provider. For other payment providers such as PayPal, ACI and IATS, uPaySiteId is not mandatory. The system will not consider the uPaySiteId value, even if provided.
Vendor	Specify the Vendor (case sensitive) used to log in to your IATS account.

Whenever you edit a field in a Settings panel with hidden fields (i.e., sensitive data that are removed from the client-side view like passwords, keys, and secrets), you must re-enter the hidden values, otherwise they will get cleared upon update and resave of all the Settings.

Configure the Hosted Payment Page

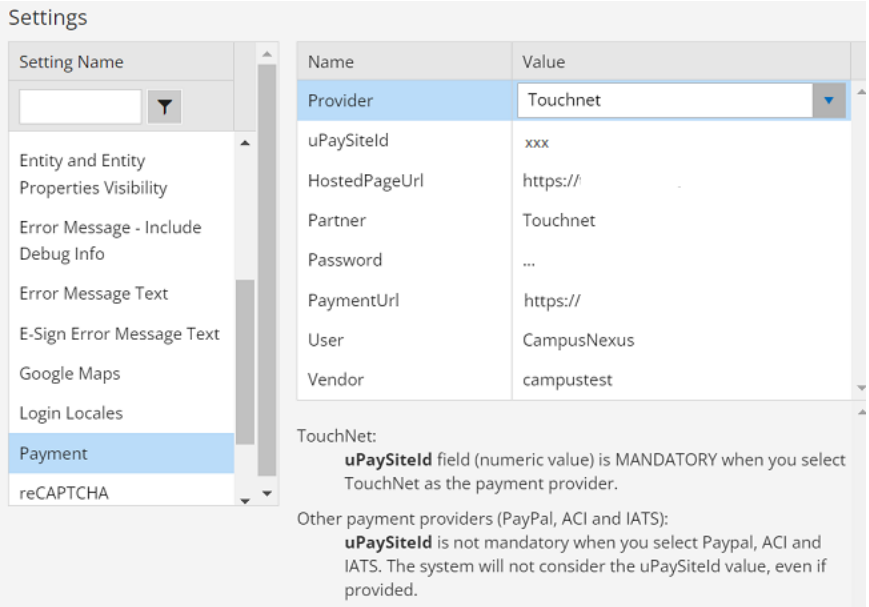
The default IATS payment page can be customized as needed. To do so, log in to the IATS Customer Portal at https://www.iatspayments.com and navigate to Aurora Form Setup.

Payment Processing with TouchNet

The Forms Builder settings below are required to use TouchNet as a vendor for credit card payment processing. For additional setup requirements, please contact your Anthology Inc. account representative.

Populate Forms Builder Settings

In the Forms Builder Settings workspace, populate the **Payment** setting fields.



Provide the **Password**, **User**, and **Vendor** values. The remaining fields are preset. If you change the credit card payment provider later, you must update the URLs, remove the Partner entry, and update the Password, User, and Vendor. The values specified in the Settings workspace are saved to the web.config file for Form Designer.

Name	Value
HostedPageUrl	URL that identifies the location of the payment processor's page.
Partner	Partner for your TouchNet account.
Password	Specify the Password (case sensitive) used to log in to your TouchNet account.
PaymentUrl	URL that identifies the payment page.
Provider	Name of the credit card payment provider: TouchNet
User	Specify the User (case sensitive) used to log in to your TouchNet account. It can be the same as the Vendor.
uPaySiteId	Indicates the URL to the TouchNet uPay page. TouchNet gateway provider URLs are not populated initially. You need to populate the URLs for TouchNet manually since the URLs are unique for each client's account. This field (numeric value) is MANDATORY when you select TouchNet as the payment provider. For other payment providers such as PayPal, ACI and IATS, uPaySiteId is not mandatory. The system will not consider the uPaySiteId value, even if provided.
Vendor	This field is not required for TouchNet account.



Whenever you edit a field in a Settings panel with hidden fields (i.e., sensitive data that are removed from the client-side view like passwords, keys, and secrets), you must re-enter the hidden values, otherwise they will get cleared upon update and resave of all the Settings.

Configure the Hosted Payment Page

The default TouchNet payment page can be customized as needed.

Date Picker

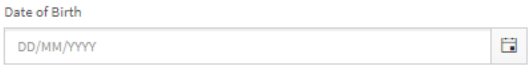
You can use the Date Picker component to quickly select a date and prevent the entry of an invalid value.

Properties

Control Property Settings

Control Type	Date Picker
Class	
Disable Input Text	false
Disabled	false
Format	dd/MM/yyyy
Id	idae170308-ac75-f35c-8173-7b7527d7af26
Ignore Time	☑
Label	Date of Birth
Maximum Value	
Minimum Value	
Model	vm.models.myDate
Read-only	false
Required	false
Required Message	
Tab Index	
Tooltip	Date of Birth
Tooltip Duration	750
Visible	true

Rendered Component



Workflow Argument

Name	Direction	Argument type	Default value
myDate	In/Out	DateTime	*Default value not supported*

For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Disable Input Text** disallows typing in the field. When true, dates must be selected from the date picker popup.
    - If this property is bound, it must start with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)

    > ⚠ Do not specify the Disable Input Text property if tabbing through each item in a rendered form using only the keyboard (i.e., not using a mouse) is required.

- **Disabled** sets a control to disabled.
    - Must be true or false, or a binding beginning with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Format** property settings depend on the control type used:
    - **Date Picker**:

      Specifies the format of the date and, when text input is allowed (Disable Input Text = false), the parsing of typed text.

      Examples: yyyy-MM-dd, MM-dd-yy (case sensitive: d for day of month, M for month, y for year)
    - **Time Picker**:

Specifies the format of the time and, when text input is allowed (Disable Input Text = false), the parsing of typed text.

Examples: HH:mm, hh:mm:ss tt (case sensitive: H or h for hour, m for minute, s for second)

Refer to Kendo UI documentation for more information.

- **Date Time Picker**:

  Combines the two above with both date and time.

- **Numeric Textbox**:

  Specifies format of the numeric value. e.g., n2 - 2 decimal places, c - currency with cents

> ⚠ The date parsing and formats for these controls changed in Forms Builder 3.5. If you used date formats that are no longer supported, you need to update and re-save affected forms.

- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
  - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
  - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
  - Binding is not supported for this property.
- **Ignore Time** is an optional property that can be set to true or false (all lowercase). The default is false for backward compatibility, **but we strongly recommend that you set it to true when you use this control**.

  When set to true and a date is entered, the control will set a date only with no time. This prevents the server from changing the date when it is in a different time zone than the browser. This date will pair with a DateTime or DateTimeOffset type argument in the workflow and set the date and time as midnight no matter what time zone the Renderer server is in. Some entities use a Date object which does not have a time so this would not be required for them.

  If a DatePicker control is used as a minimum or maximum date dynamically by another control, IgnoreTime must be set true to avoid conversion errors in the browser developer tools (F12).

  The Ignore Time property is not dynamically bindable (cannot take an expression). Once set in a form, it cannot be changed.
- **Label** is the value displayed in the label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Maximum Value** specifies a maximum date value.

- To provide forward compatibility, the date string should be in ISO 8501 format, e.g., 2017-09-03 (which is a universal non-ambiguous format).
- If bound, it starts with {{vm.models. and ends with }}.

- **Minimum Value** specifies a minimum date value.
  - To provide forward compatibility, the date string should be in ISO 8501 format, e.g., 2017-09-03 (which is a universal non-ambiguous format).
  - If bound, it starts with {{vm.models. and ends with }}.
- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.
  - The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
  - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
  - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
  - The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
  - If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

  If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

  If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

  Construction of the model in the workflow is done by assigning data from a provider.
- **Read-only** makes the control read-only. It is set to false by default. If you want the component to be read-only, set the property value to true. It is typically used for an input box.
  - Can be bound to a workflow argument or another control's value.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).

- If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

- **Required** makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.
  - Can be bound to a workflow argument or another control's value.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
  - If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)
- **Required Message** is optional. It overrides the default "Required" message.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
  - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
  - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
  - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

    The tab index value should not match another control's tab index.
  - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.
- **Tooltip** is the value to display when hovering over the control's label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
    - When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

When binding controls, **String** and **Integer** properties require the Model value to be enclosed in **double curly braces**, for example, `{{vm.models.myTooltip}}` for Tooltip or `{{vm.models.myMinValue}}` for a Text Box of Type Number.

Boolean properties do not need the curly braces, for example, `vm.models.myRequired`.

This control has the capability to output an ISO 8601 String value which is converted to a DateTime or DateTimeOffset object (depending on the type of the workflow argument). For more information, refer to Date & Time Values and Offsets.

Troubleshoot a Date Picker

When a Date Picker is populated with values from a database, the Date Picker's Control Property Settings determine whether correct date values are displayed on the form if the client browser is in a different time zone than the database server.

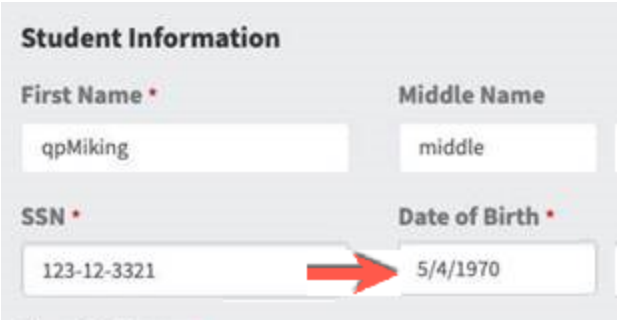
In this example, a form with Date of Birth field was bound to the Anthology Student database. However, the Date of Birth value on the form did not match the value in the database — it was off by one day. To correct this issue, the Date Picker's Control Property Settings were configured as follows:

- Format: MM/dd/yyyy
- Ignore Time: true

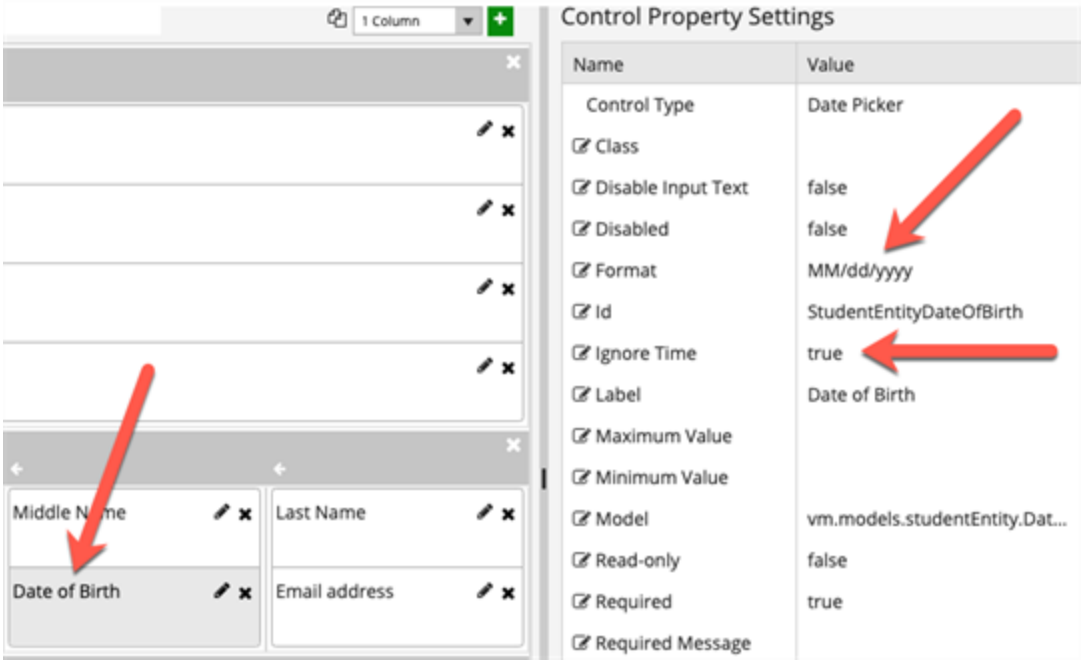
Date Value in Anthology Student



Incorrect Date Value on the Rendered Form



Corrected Date Picker Component



Date Time Picker

You can use the Date Time Picker component to quickly select a date and time. This component helps prevent the entry of invalid date and time values.

Properties

Control Property Settings

Control Type	Date Time Picker
Class	
Disable Input Text	false
Disabled	false
Format	
Id	id2be1d386-c58a-da50-5b9e-0492be226294
Interval	30
Label	Date and Time
Maximum Value	2020-12-31 12:00
Minimum Value	2020-12-1 12:00
Model	vm.models.myDateTime
Read-only	false
Required	false
Required Message	
Tab Index	
Tooltip	Select the Date and Time
Tooltip Duration	750
Visible	true

Specify the maximum date and time and minimum date and time if you want to validate a value range.

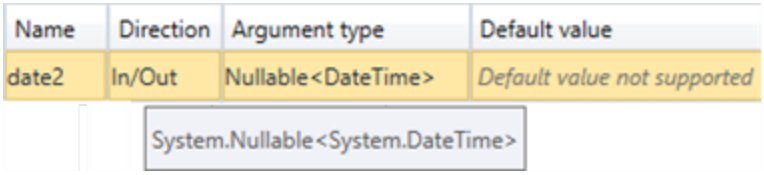
Rendered Component



Workflow Argument

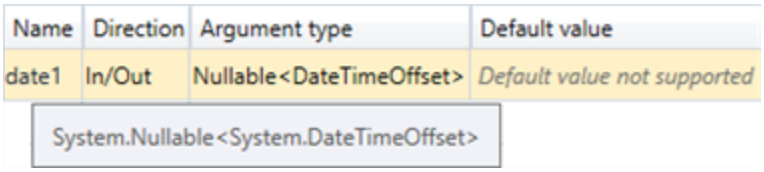
The argument type must be *System.Nullable<System.DateTime* if you intend to store a value to the database. Do not use an argument type of *String*.

When vm.models.myDate is “2018-02-22T09:00:00” (which is the DateTime object serialized for that date and time), it is treated as a local time in the component. No matter what time zone the component is in.



If the value is used only for display and print purposes, you may use an argument type of *System.Nullable<System.DateTimeOffset*. This represents a date and time value together with an offset that indicates how much that value differs from UTC.

When vm.models.myDate is “2018-02-22T09:00:00Z” or “2018-02-22T09:00:00-08:00”, it will be different depending on which time zone the component is in.



For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Disable Input Text** disallows typing in the field. When true, dates and times must be selected from the date time picker popups.
    - If this property is bound, it must start with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)

  If Input Text is allowed, the date and time value entry must be complete including AM/PM (e.g., 02/01/2018 4:00 PM), otherwise the date and time value on the rendered form will be an empty string.
- **Disabled** sets a control to disabled.

- Must be true or false, or a binding beginning with "vm.models.".
- A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
- If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

- **Format** property settings depend on the control type used:
  - **Date Picker**:

    Specifies the format of the date and, when text input is allowed (Disable Input Text = false), the parsing of typed text.

    Examples: yyyy-MM-dd, MM-dd-yy (case sensitive: d for day of month, M for month, y for year)
  - **Time Picker**:

    Specifies the format of the time and, when text input is allowed (Disable Input Text = false), the parsing of typed text.

    Examples: HH:mm, hh:mm:ss tt (case sensitive: H or h for hour, m for minute, s for second)

    Refer to Kendo UI documentation for more information.
  - **Date Time Picker**:

    Combines the two above with both date and time.
  - **Numeric Textbox**:

    Specifies format of the numeric value. e.g., n2 - 2 decimal places, c - currency with cents

  ⚠ The date parsing and formats for these controls changed in Forms Builder 3.5. If you used date formats that are no longer supported, you need to update and re-save affected forms.

- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
  - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
  - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
  - Binding is not supported for this property.
- **Interval** for the time picker. The default is 30 minutes.

- If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myInterval}}.
- Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Label** is the value displayed in the label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Maximum Value** specifies a maximum date AND time value.
  - To provide forward compatibility, the date string should be in ISO 8501 format., e.g., 2017-09-03 (which is a universal non-ambiguous format).
  - If bound, it starts with {{vm.models. and ends with }}.
- **Minimum Value** specifies a minimum date AND time value.
  - To provide forward compatibility, the date string should be in ISO 8501 format., e.g., 2017-09-03 (which is a universal non-ambiguous format).
  - If bound, it starts with {{vm.models. and ends with }}.
- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.
  - The Model value must always start with "vm.models.".
  - The argument type must be Nullable.
  - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
  - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
  - The casing of an argument used in the workflow must match the "vm.models." suffix casing.
- **Read-only** makes the control read-only. It is set to false by default. If you want the component to be read-only, set the property value to true. It is typically used for an input box.
  - Can be bound to a workflow argument or another control's value.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).

- If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

- **Required** makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.
  - Can be bound to a workflow argument or another control's value.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
  - If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)
- **Required Message** is optional. It overrides the default "Required" message.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
  - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
  - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
  - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

    The tab index value should not match another control's tab index.
  - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.
- **Tooltip** is the value to display when hovering over the control's label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
    - When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

When binding controls, **String** and **Integer** properties require the Model value to be enclosed in **double curly braces**, for example, `{{vm.models.myTooltip}}` for Tooltip or `{{vm.models.myMinValue}}` for a Text Box of Type Number.

Boolean properties do not need the curly braces, for example, `vm.models.myRequired`.

This control has the capability to output an ISO 8601 String value which is converted to a DateTime or DateTimeOffset object (depending on the type of the workflow argument). For more information, refer to Date & Time Values and Offsets.

DocuSign

You can use the DocuSign component to integrate DocuSign fields into a form. Forms Builder supports a several types of DocuSign fields, e.g., Approve, Decline, Sign, Initial, and others. The fields are selected in the **Type** property of the Control Property Settings.

Before using the DocuSign component, you must set up and account with DocuSign and configure the DocuSign Settings in Forms Builder.

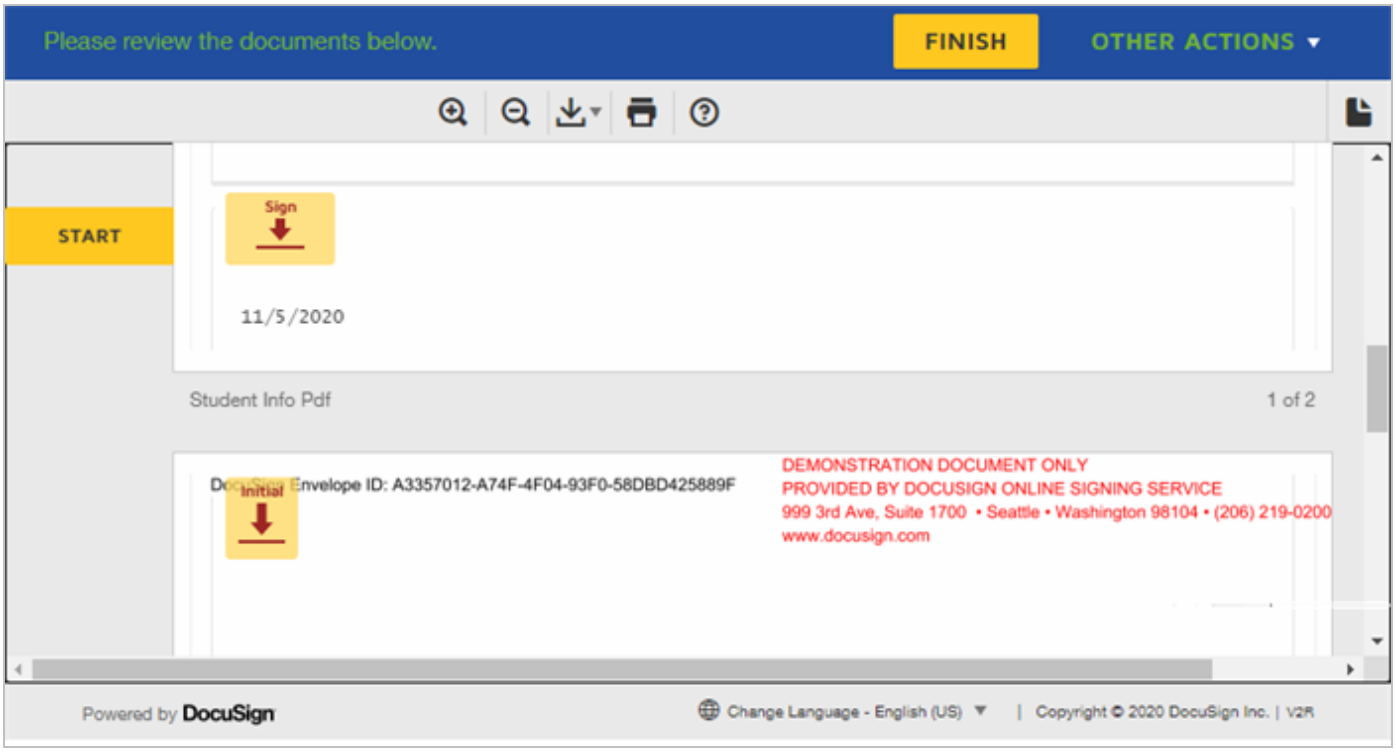
Properties

Control Property Settings

Control Type	DocuSign
Class	
Signer	Self - Signer 1
Tag Name	
Type	Signature
Visible	true

Control Type	DocuSign
Class	
Signer	Self - Signer 1
Tag Name	
Type	Initial
Visible	true

Rendered Components (Type=Signature and Type=Initial)

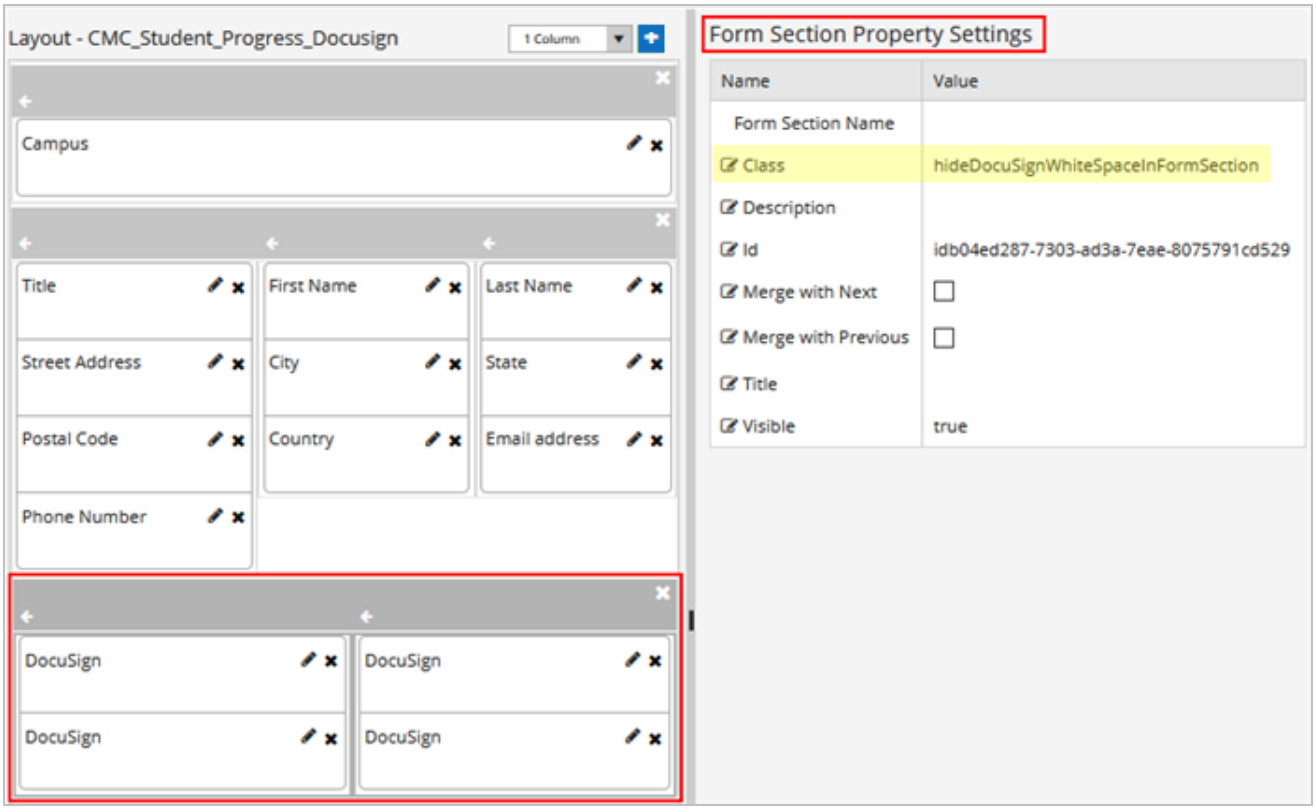


- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer

to Custom Styles.

When you design a form with hidden fields or Form Sections, you may need to hide the empty space for e-sign components to ensure that the form is rendered as intended. To hide the space for e-sign components:

1. Place the e-sign components in their own **Form Section** (usually at the bottom of a form).
2. In the Form Section Property Settings, specify the Class name **hideDocuSignWhiteSpaceInFormSection**.
3. Save the form. Forms Builder will not render the Form Section on the screen but will allow it to render when it is printed to a PDF.



- **Signer** provides the following options:
  - Self - Signer 1
  - Signer 2
  - Signer 3
  - Signer 4
  - Signer 5

- **Type** provides the following DocuSign fields:

Field	Description	Required/ Optional	User Input
Approve	Allows the recipient to approve documents without placing a signature or initials on the document.	Required	Yes
Attachment	Allows the recipient to attach supporting documents to an envelope.	Required	Yes
Checkbox	Allows the recipient to select a yes/no (on/off) option.	Optional	Yes
Company	Allows the recipient to specify the company name.	Required	Yes
Date	Allows the recipient to enter a date.	Required	Yes
Date Signed	Allows the recipient to specify the date the document was signed.	Required	Yes
Decline	Allows the recipient the option of declining an envelope.	Optional	Yes
Email	Allows the recipient to enter an email address.	Required	Yes
Email Address	Allows the recipient to specify an email address.	Required	Yes
Envelope Id	Displays (read-only) the envelope ID. The Envelope object is the overall container for a transaction. The envelope contains the documents for the eSignature transaction. It also contains information about the recipients and timestamps that indicate delivery progress.	Required	No
Full Name	Allows the recipient to specify his/her full name.	Required	Yes
Initial	Allows the recipient to initial the document.	Required	Yes
Number	Allows the recipient to enter numbers and decimal (.) points.	Required	Yes
Signature	Allows the recipient to sign a document. May be optional.	Required	Yes
Ssn	Allows the recipient to enter a Social Security Number.	Required	Yes
Text	Allows the recipient to enter any type of text.	Required	Yes

- **Tag Name** applies to the following fields in the Type control for a specific signer:
    - Checkbox
    - Date
    - Email
    - Numbers
    - Ssn
    - Text

  When any of the above fields is inserted multiple times in a form, typing a **Tag Name** for each field prevents the recipient's typed value from being replicated to remaining fields of the same type.

  Alternatively, if tag names are same or blank in a document or envelope with fields of an identical type, data population scope will be turned on and field data will be linked. The recipient's typed value in a field will be

replicated to all fields of the same type. For more information about this feature, refer to Using Data Population Scope.

Across all the forms in a sequence, to enable a unique value to be typed in each field of a specific type, assign a unique Tag Name to the field.

Example to turn off data population scope

Type a unique value for each Tag Name to delink fields of the same type from each other. In the form, the recipient's typed value will then be unique, i.e., the values will not be copied to all fields of the same type.

A form with text fields is inserted twice as follows:

- With the value **First Name** in the **Tag Name** field.
- With the value **Last Name** in the **Tag Name** field.

A value typed in the first field will not be replicated to the second field, and vice versa.

Example to turn on data population scope

Type identical tag names or leave the **Tag Name** blank. This will link all such fields in the form, and the recipient's typed value in any field will be copied to all fields of the same type.

Notes:

- Values typed in the **Tag Name** apply only to the 6 fields listed above. Tag Name values typed for other fields will not affect the behavior of the field.
- In an upgrade scenario to version 4.1, the **Tag Name** will be blank. Type a value to enable the behavior described above.

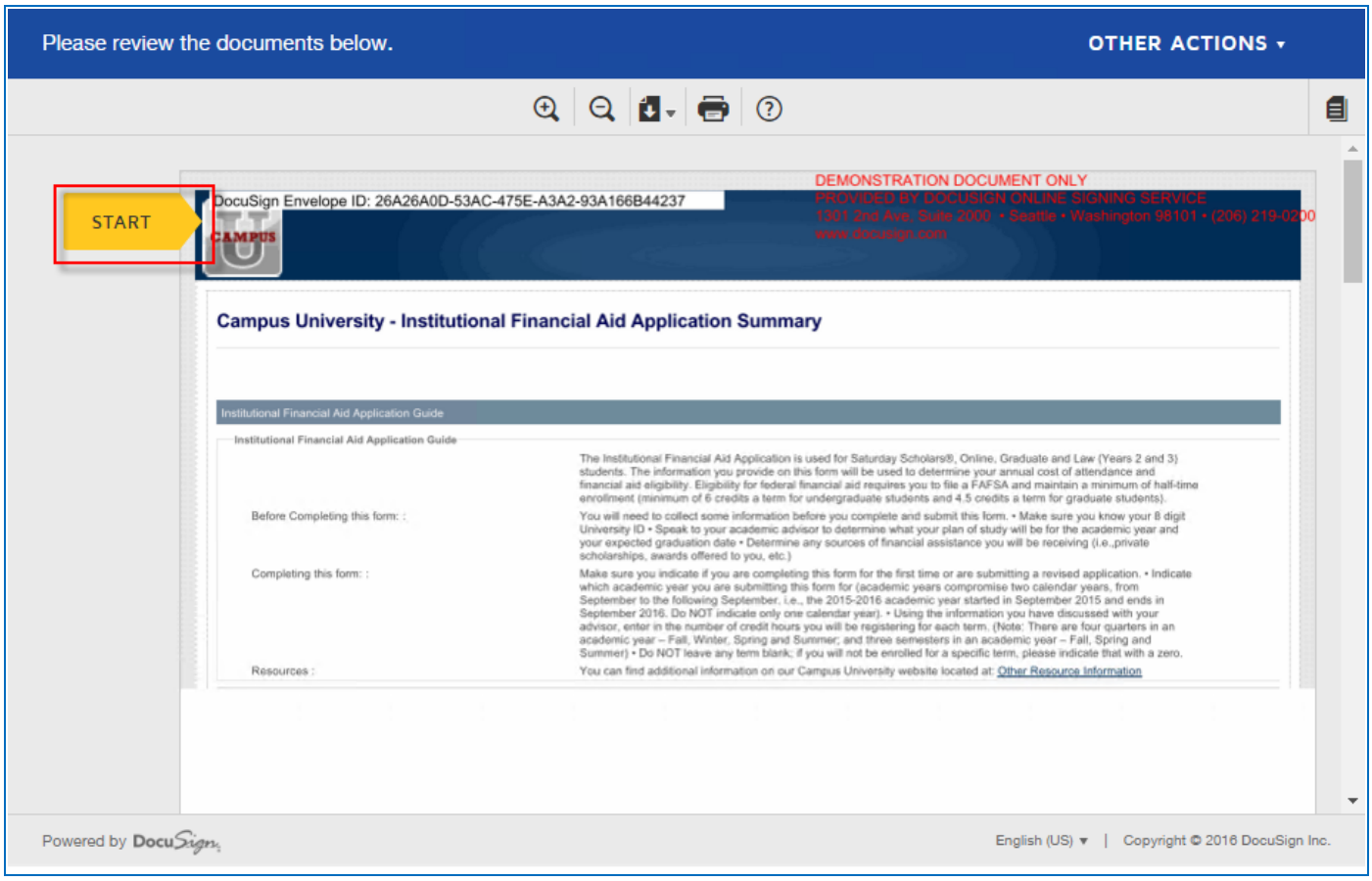
- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)

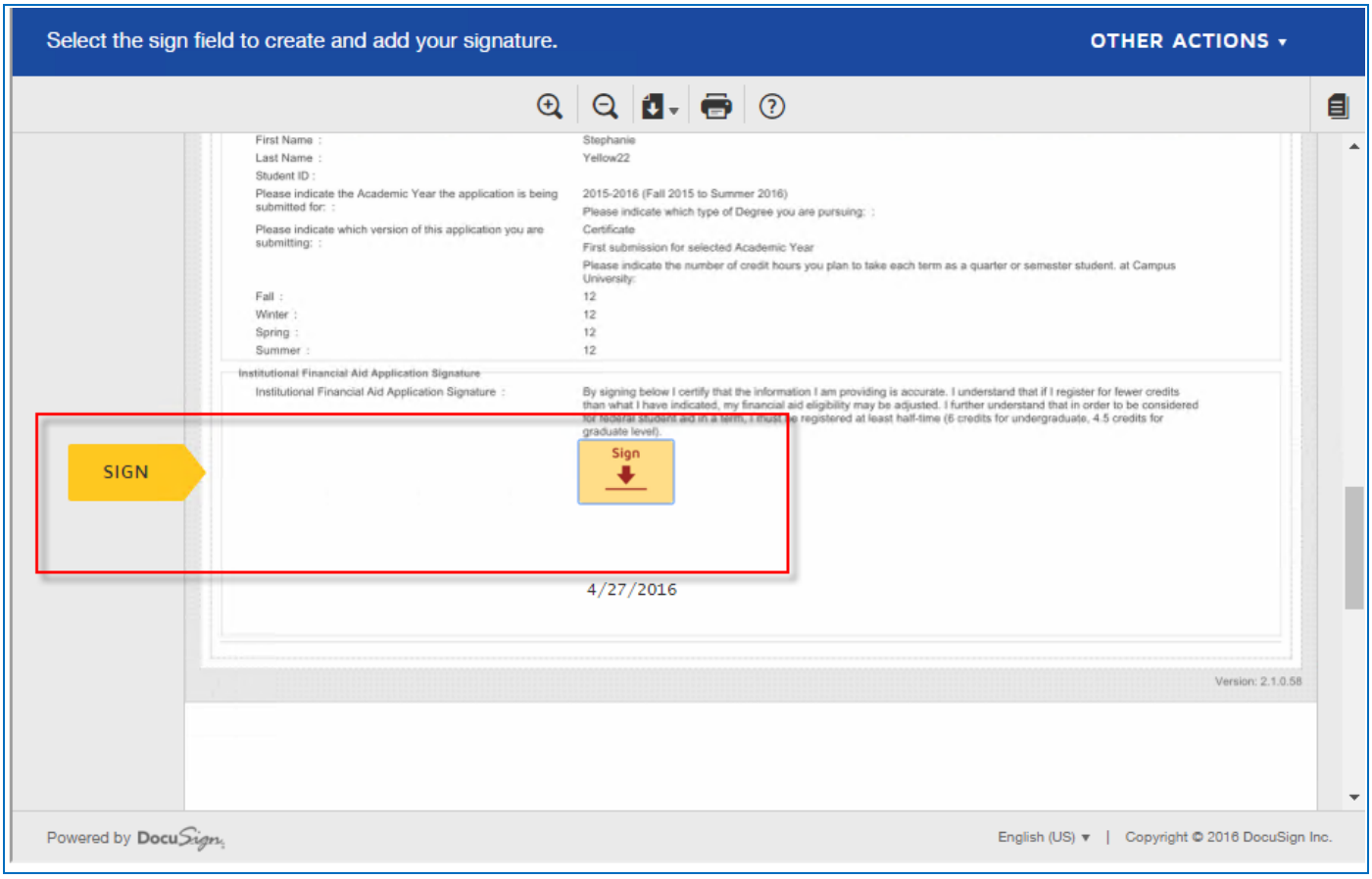
Working with the DocuSign Component

You are designing a form in which you want the person completing the form to sign and date the form electronically.

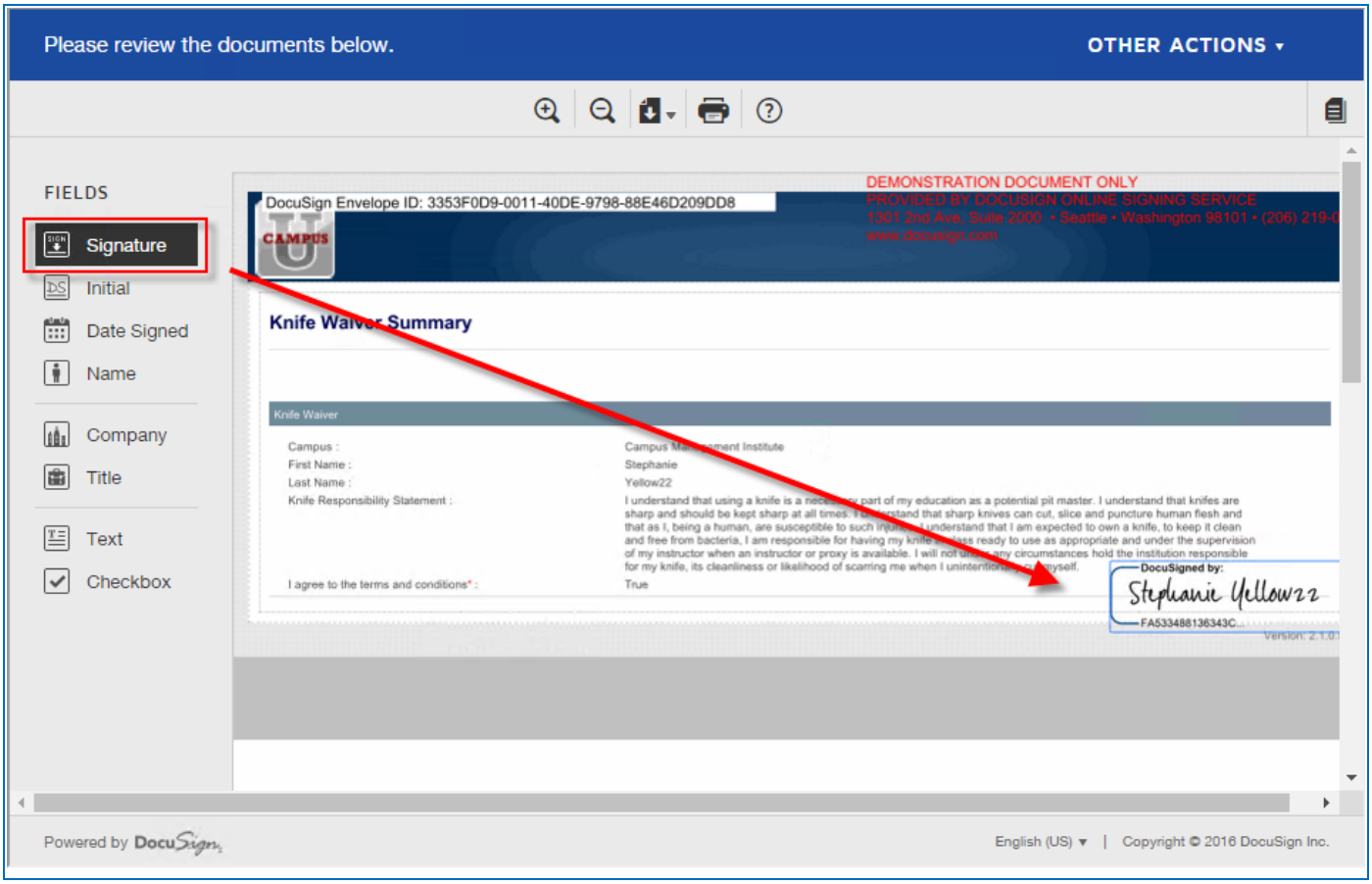
1. Drag two DocuSign components into the Layout pane of your form in Form Designer.
2. On the first DocuSign component, set the Signer to **Self - Signer 1** and set the Type to **Signature**.
3. On the second DocuSign component, set the Signer to **Self - Signer 1** and set the Type to**Date Signed**.

When Anthology Student fields are inserted into a form, the student is automatically directed to the section that needs an e-signature.


Please review the documents below.
OTHER ACTIONS
START
DocuSign Envelope ID: 26A26A0D-53AC-475E-A3A2-93A166B44237
DEMONSTRATION DOCUMENT ONLY
PROVIDED BY DOCUSIGN ONLINE SIGNING SERVICE
1301 2nd Ave. Suite 2000 • Seattle • Washington 98101 • (206) 219-0200
www.docusign.com
Campus University - Institutional Financial Aid Application Summary
Institutional Financial Aid Application Guide
Institutional Financial Aid Application Guide
The Institutional Financial Aid Application is used for Saturday Scholars®, Online, Graduate and Law (Years 2 and 3) students. The information you provide on this form will be used to determine your annual cost of attendance and financial aid eligibility. Eligibility for federal financial aid requires you to file a FAFSA and maintain a minimum of half-time enrollment (minimum of 6 credits a term for undergraduate students and 4.5 credits a term for graduate students).
Before Completing this form: :
You will need to collect some information before you complete and submit this form. • Make sure you know your 8 digit University ID • Speak to your academic advisor to determine what your plan of study will be for the academic year and your expected graduation date • Determine any sources of financial assistance you will be receiving (i.e.,private scholarships, awards offered to you, etc.)
Completing this form: :
Make sure you indicate if you are completing this form for the first time or are submitting a revised application. • Indicate which academic year you are submitting this form for (academic years compromise two calendar years, from September to the following September, i.e., the 2015-2016 academic year started in September 2015 and ends in September 2016. Do NOT indicate only one calendar year). • Using the information you have discussed with your advisor, enter in the number of credit hours you will be registering for each term. (Note: There are four quarters in an academic year – Fall, Winter, Spring and Summer; and three semesters in an academic year – Fall, Spring and Summer) • Do NOT leave any term blank; if you will not be enrolled for a specific term, please indicate that with a zero.
Resources :
You can find additional information on our Campus University website located at: Other Resource Information
Powered by DocuSign
English (US)
Copyright © 2016 DocuSign Inc.



If the fields are not inserted into a form but e-signature is enabled, the student is directed to electronically sign the document and can drag out the signature anywhere on the document.



In Forms Builder 3.6 and later, the DocuSign component allows for a DocuSign sequence to be embedded in an IFrame. Previously, users received a "Waiting for data" message after successfully completing an embedded DocuSign sequence. Now, users will be redirected to the parent sequence with a "DocuSign complete" form and/or confirmation form.

The DocuSign component provides an automatic transition (auto-redirect) from the Default-Frame form to the confirmation form after a successful DocuSign session. The auto-redirect obsoletes the "DocuSign Confirmation Message Text" setting.

The auto-redirect depends on a forward direction in the WaitForFormBookmark activity in the transition after the DocuSign redirect state (typically Default-Frame), in particular if DisplayName has been modified.

- If there is only a single button and DisplayName has been customized but Transition Type was left as "Default", the auto-redirect moves forward to next form state.
- If there are two buttons and DisplayName(s) have been customized but Transition Type was left as "Default", the auto-redirect will assume the rightmost button (alphabetically last) is the transition for next state.

Best Practice is always to specify Display Order and Transition Type (“MoveForward” or “MoveBack”) when button Display Name(s) have been customized so behavior is known. The Transition Type of “Default” was kept for compatibility for forms built prior to Transition Type being available on WaitForFormBookmark with default Display Names "Next" and "Back”.

Localization of DocuSign Sequences

To localize sequences with DocuSign components, follow the procedure described here: Steps to Localize Sequences. There are no additional steps to be completed for DocuSign sequences. DocuSign does not provide a method for Forms Builder to pass in a locale.

The DocuSign portion of the sequences (Start, Sign here, Finish, and Completed email) displays drop-down control with a limited list of locales that are supported by DocuSign. In single signer sequences, the language selection control is in the IFrame of the sequence. In multiple signer sequences, the language control is on the DocuSign site.



The language selection for the DocuSign portion of the sequences is cached in the user's browser. In multiple signer sequences, the primary signer and other signers need to set the languages themselves. For example, one secondary signer may select French, while another secondary signer may select Italian. However, the PDF translation within the form itself is based on primary signer's locale selection on the form prior to the redirection to the DocuSign site for the secondary signers.

Allow Sequential Signing

Forms Builder 3.6 and later supports RoutingOrder as an optional property on the DocuSignRecipient entity. If your DocuSign form sequences use this property, ensure that you have permissions to create a routing order.

1. Log in to your account at https://admin.docusign.com.
2. Go to **DocuSign Admin.** and navigate to **Permission Sets > Action: Edit > User Permissions**.
3. Select **Allow sequential signing**.
4. **Save** your changes.

If the RoutingOrder is not explicitly specified in the workflow, the RoutingOrder will be initialized with SignerId in the activity (Signer 1 gets 1, Signer 2 gets 2, etc.). When the first recipient has a routing order of 1 and the second recipient has a routing order of 2, the second recipient will not receive the request until the first signer has finished signing.

If the third and fourth recipients have a routing order of 3, they will simultaneously receive a copy of the completed envelope once the second signer completes their actions.

For more information, refer to Get the DocuSign Configuration and Pass the Recipient Information in *DocuSign Workflow Sample - Multiple Signers*.

Drop-down List

You can use the Drop-down List component to select a single value from a list of values.

Forms Builder supports the following types of drop-down controls:

- **Default** drop-down controls for fields in the CampusNexus entity model. Built-in **Lookup Queries** retrieve the list values for default drop-down controls.
    - Example: Campus Drop-down List
    - Example: Program Field with Area of Study Drop-down List
- **Custom** drop-down controls with a **Value List** defined using the Edit button on the Value List property. Refer to Drop-down List with Value List.
- **Custom** drop-down controls with a **Workflow Initialized List** defined using the Edit button on the Value List property.
    - Drop-down List with Workflow Initialized List
    - Drop-down List with Workflow Initialized List via ExecuteQuery
    - Drop-down List with Workflow Initialized List and Template

Properties

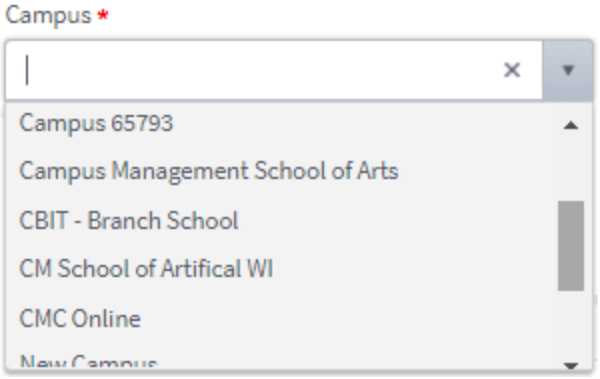
Example: Campus Drop-down List

This example shows a Drop-down List component for the selection of a Campus.

Control Property Settings

Control Type	Drop-down List
Class	
Default Value	string
Disabled	false
Filter Type	contains
Id	ide75fd16a-a782-9745-6cd9-d083283d8ec7
Label	Campus
Lookup Display Member	Name
Lookup Query	Campuses?$select=Code, Name, Id&$filter=IsActive eq true&$orderby=Name
Lookup Sort Member	Name
Lookup Translation Members	
Lookup Value Member	Id
Model	vm.models.myCampus
Option Label	<Select>
Page Size	100
Product	Student
Read-only	false
Required	true
Required Message	
Server Filtering	☐
Tab Index	
Template	
Tooltip	
Tooltip Duration	750
Value List	Edit...
Visible	true

Rendered Component



Workflow Argument

Name	Direction	Argument type	Default value
myCampus	In/Out	Int32	*Default value not supported*

Example: Program Field with Area of Study Drop-down List

This example shows a default drop-down list control for the Program field paired with a Drop-down List component for Areas of Study associated with a selected Program value.

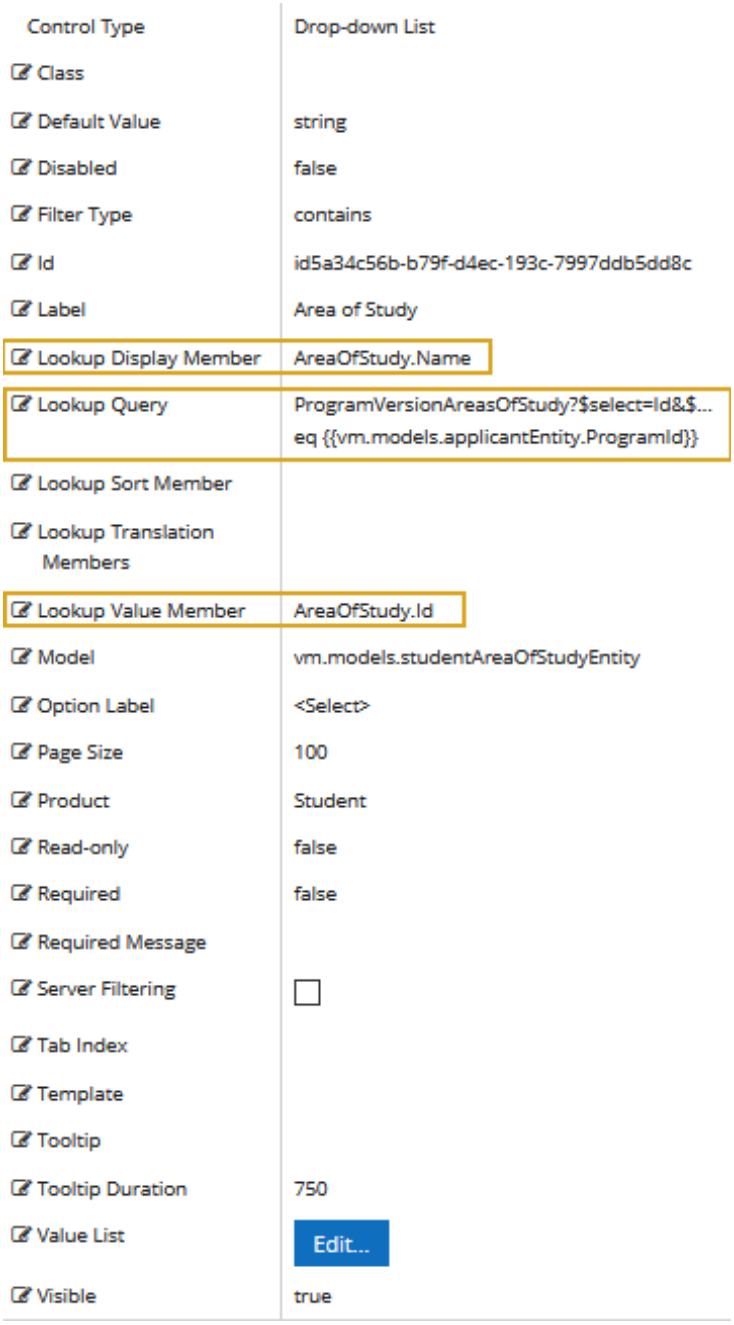
Control Property Settings for the Program Field

The Program field in the Applicant entity provides the following properties and is bound to the ProgramId value.

Control Type	Drop-down List
Class	
Default Value	string
Disabled	false
Filter Type	contains
Id	ApplicantEntityProgramId
Label	Program
Lookup Display Member	Name
Lookup Query	Programs?$select=Code,Name,Id&$filter=IsActive eq true
Lookup Sort Member	Name
Lookup Translation Members	
Lookup Value Member	Id
Model	vm.models.applicantEntity.ProgramId
Option Label	<Select>
Page Size	100
Product	Student
Read-only	false
Required	false
Required Message	
Server Filtering	☐
Tab Index	
Template	
Tooltip	
Tooltip Duration	750
Value List	Edit...
Visible	true

Control Property Settings for the Area of Study Component

The Area of Study Drop-down List component has an OData Query that retrieves the Area of Study values for a selected ProgramId.

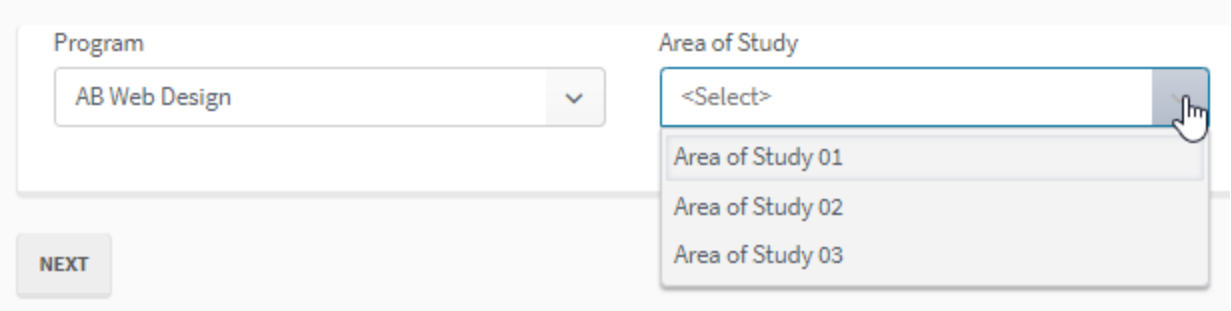


The OData Query is as follows:

ProgramVersionAreasOfStudy?$select=Id&$expand=AreaOfStudy($se-
lect=Name)&$filter=ProgramVersion/Program/Id eq {{vm.models.applicantEntity.ProgramId}}

For more information, refer to Build Queries Using the Data Model.

Rendered Program Field and Associated Area of Study Drop-down List Component



- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Disabled** sets a control to disabled.
    - Must be true or false, or a binding beginning with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Filter Type** defines how search values will be filtered. Select from the following filter types: *contains* (default), *endswith*, and *startswith*. Values typed using the keyboard will be used to filter the list according to the selection.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Label** is the value displayed in the label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Lookup Display Member** is the name of the property in the OData query string to use for the display.

For example, if the query string for a list of Campuses contains the Code, Name, and ID fields, the Lookup Display Member value can be set to Code, Name, or ID.

- **Lookup Query** is the OData query string to retrieve values for the control.

  The following is an example of an OData query string that retrieves the Code, Name, and ID values from the Campus table, where `isActive` equals true and the returned values are sorted by Name.

  ```
  Campuses?$select=Code, Name, Id&$filter=IsActive eq true&$orderby=Name
  ```

  Lookup Query is **not** used if a custom Value List is specified. Refer to Drop-down List with Value List.

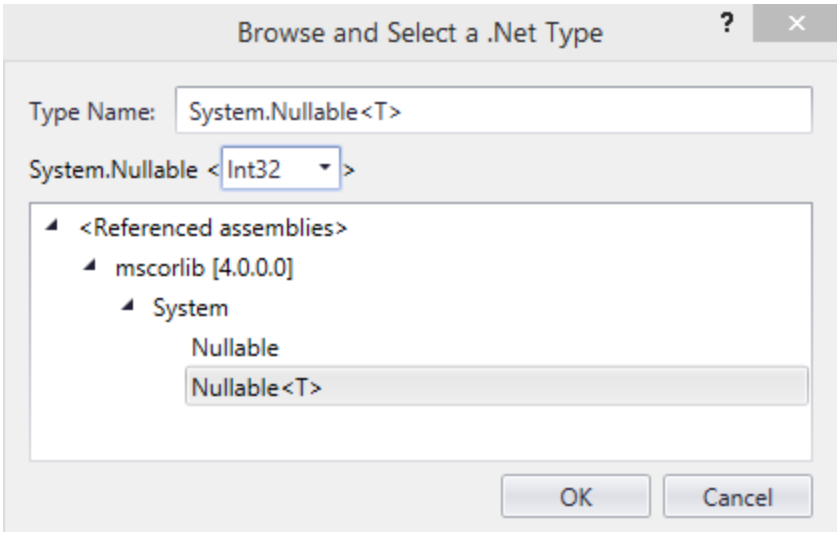
- **Lookup Sort Member** is the name of a property in a Lookup query string to sort on. By default, the Lookup query sort order is used.
- **Lookup Translation Members** is a comma separated list of property names in an OData query string to be translated. You should always validate the query will work in a browser. Only basic errors can be detected in Form Designer.
- **Lookup Value Member** is the name of the property in the OData query string to use as the value.
  - If the Lookup Value Member is an Id, the associated data type in Workflow Composer is `Int32`.
  - If the Lookup Value Member is a Code or Name, the associated data type in Workflow Composer is `String`.
- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.
  - The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
  - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
  - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
  - The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
  - If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

  If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

  If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

  Construction of the model in the workflow is done by assigning data from a provider.

When the Drop-down List is used to retrieve integer values and the field is optional (can be empty or null), this must be accounted for when defining the variable for the model binding in the workflow. Instead of defining the variable as an `Int32`, it must be defined as `Nullable<Int32>`. To do so: In the "Browse and Select a .Net Type" window, browse to Type **System.Nullable<T>**and select **Int32** in the System.Nullable field.



- **Option Label** is the label in a drop-down list when no list value is selected. The default value is <Select>.

  In forms created prior to Forms Builder 3.3, the default Option Label is rendered blank. Re-save these forms in Forms Builder 3.3 to render the default Option Label as <Select>.

- **Page Size** is the size of the page when paging for returned values is true. The default value is 100. Example: If Page Size is set to 10 and the number of returned values exceeds 10, paging occurs.

- **Product** indicates the product from which OData query results are returned. Specify one of the following values:

  - Student
  - CRM
  - Occupation Insight

  The specified product must be configured in the <products> section of the Renderer web.config file.

  The default Product value will be "Student" if "Student" is selected in the <Select Provider> list on the Fields tab.

  The default Product value will be "CRM" if "CRM" is selected in the <Select Provider> list on the Fields tab.

  Specify "Occupation Insight" in the Product property if the source of the query will come from a different data source other than Student/CRM. For more information, refer to Build Queries for Occupation Insight.

A form can have multiple controls that retrieve data from different providers. For example, a form can have a control that is populated by a query to the Student database. The same form can have another control that retrieves data from Occupation Insight.

Specify the query to retrieve data from the selected provider using the Lookup Query or ODataQuery property (as applicable for the control). The query contains only the URL specific part of an OData URI. The Base URL and Product will be supplied by the configuration.

- **Read-only** makes the control read-only. It is set to false by default. If you want the component to be read-only, set the property value to true. It is typically used for an input box.
    - Can be bound to a workflow argument or another control's value.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Required** makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.
    - Can be bound to a workflow argument or another control's value.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Required Message** is optional. It overrides the default "Required" message.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Server Filtering** turns on server filtering. An input box is provided for text, and values containing the typed text will be displayed up to the page size. This property is not bindable.

  Impact of Server Filtering on large drop-down lists:

- If Server Filtering is set to false, all list items will be displayed in the drop-down list as specified by the page size value.
- If Server Filtering is set to true, only the first list items will be displayed in the drop-down list. Any list items in positions higher than the page size value will not be visible. The user must enter search text to find matching list items. The server filter returns any occurrence of the search text in the list items.

  If Server Filtering is set to true, we suggest using the **Tooltip** property on the Drop-down List component to instruct the user to enter search text to view all matching list values.

- **Tab Index** allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
  - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
  - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
  - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

    The tab index value should not match another control's tab index.
  - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.

- **Template** is optional. You can use this property to define a custom template to format data in a rendered drop-down list. For an example, refer to Drop-down List with Workflow Initialized List and Template.

  For information on how to use the Kendo embedded JavaScript to write a conditional template, refer to https://docs.telerik.com/kendo-ui/framework/templates/overview.

- **Tooltip** is the value to display when hovering over the control's label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Value List** is optional. Select the **Edit** button to specify the source of the values to be displayed in the Drop-down List.

  For examples of custom drop-down controls with Value Lists, refer to Drop-down List with Value List and Drop-down List with Workflow Initialized List.

- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)
  - When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

Lookup Queries for CampusNexus CRM Metadata

For any drop-down or search controls that will be populated via a lookup query, the CampusNexus CRM user needs to enter values for the **Lookup Display Member** and **Lookup Sort Member** attributes. The **Lookup Query** and **Lookup Value Member** property settings should have default values (if applicable for the selected property) as these are currently specified in the metadata.

Drop-down List with Value List

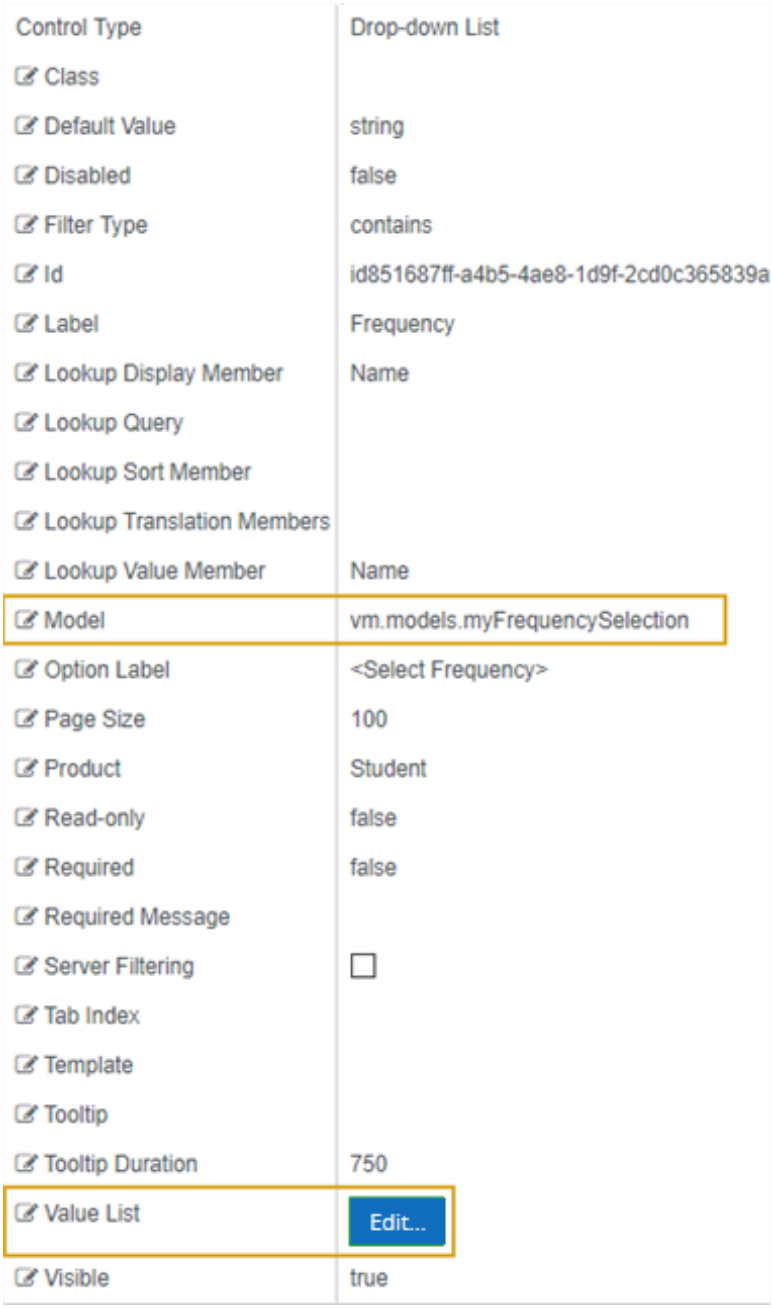
You can use the Drop-down List component to create a custom list of values for a drop-down control. In our example, the custom list contains values describing the frequency at which something occurs.

The Model property needs to define the Model binding for the selected value (e.g., vm.-models.myFrequencySelection), and the argument type in the workflow needs to be set properly based on the list values (e.g., String). The selection itself (in Control Property Settings) will NOT be of type SerializableDynamicObject[].

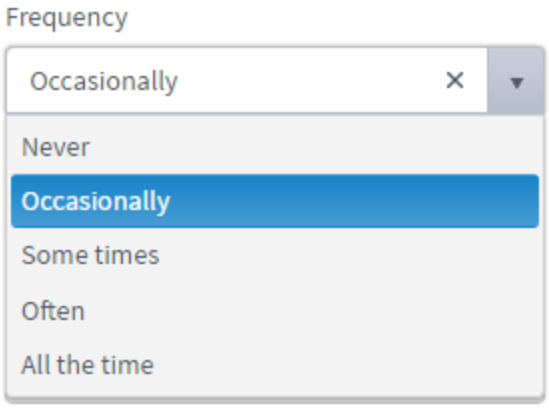
A custom value list applies only to a specific component and cannot be reused for multiple drop-down components on the same page. For example, if "Frequency" is a drop-down for multiple sections on the same page, each drop-down must define the value list with unique bindings.

This topic describes only the Value List property of the Drop-down List component. Refer to the Drop-down List topic for property settings other than Value List.

Control Property Settings



Rendered Component



Workflow Arguments

Name	Direction	Argument type	Default value
myFrequencySelection	In/Out	String	*Default value not supported*

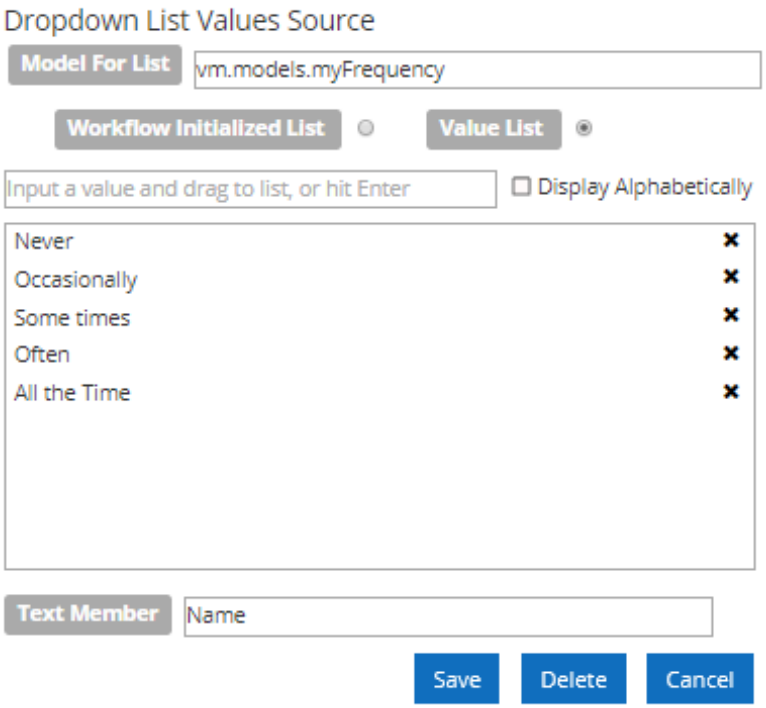
Use a workflow argument of type String to capture the selections on the form.

Name	Direction	Argument type	Default value
myFrequency	In/Out	SerializableDynamicObject[]	*Default value not supported*

Create a matching argument of type SerializableDynamicObject[] to make the Value List available in a workflow.

For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

Value List is an optional property. Select the **Edit** button to specify the source of the values to be displayed in the drop-down list.



- In the **Model For Value List** field, specify the Model property to which the drop-down list will be bound. The value must start with "`vm.models.`".
- Select **Value List** to create a custom list. Use the Model property to bind the Value List. The Value List overrides an OData Lookup Query.

  The Value List is available in a workflow if a matching argument of type SerializableDynamicObject[] is created.

  To create the list, type each value in the input field and drag it to the list area. Select ✖ in the list area to delete a value.

  In Forms Builder 3.7 and later, the **Display Alphabetically** option is available and selected by default. Clear this check box if you want to prevent the sorting of value list items on the rendered form.
- In the **Text Member** field, specify the value that will be used as the DataTextField. This is a required field. In a custom Value List, the Text Member value can be any string, e.g., `Name`.

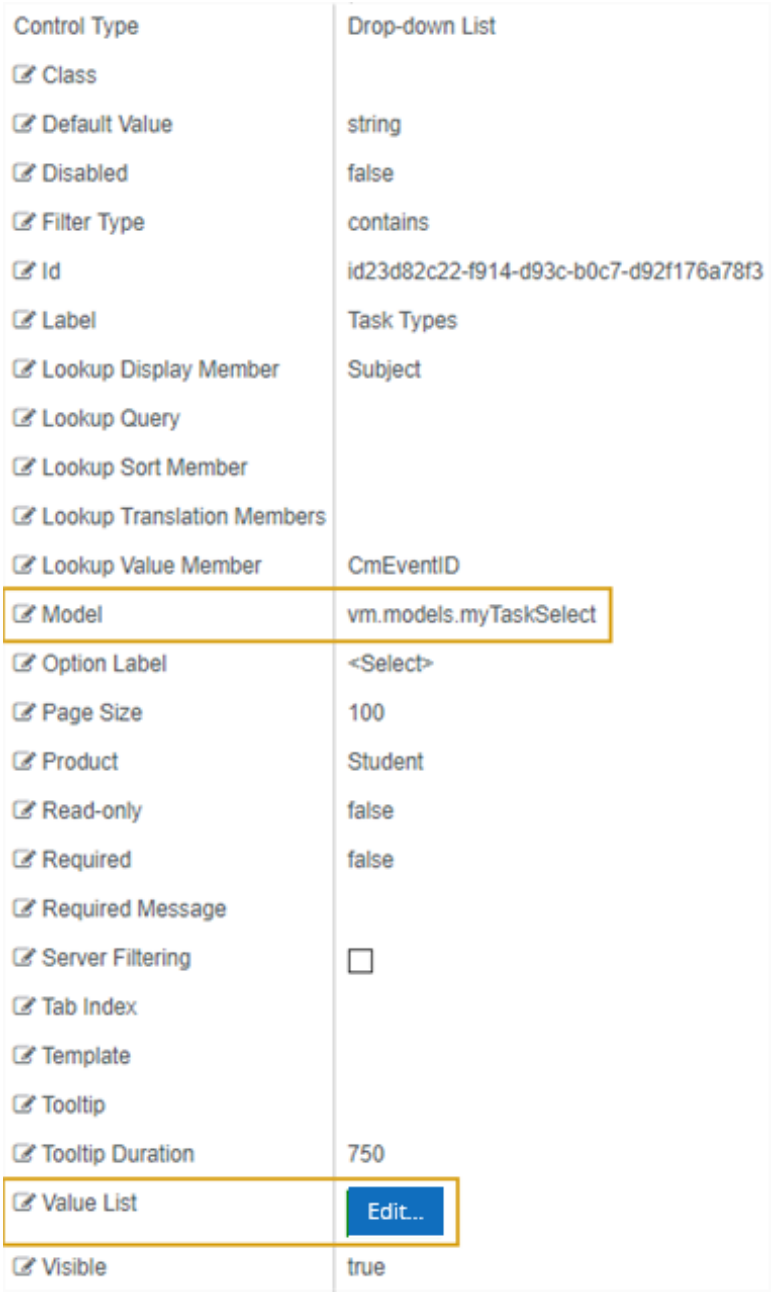
Select **Save** to save the data source settings for the drop-down list values.

Drop-down List with Workflow Initialized List

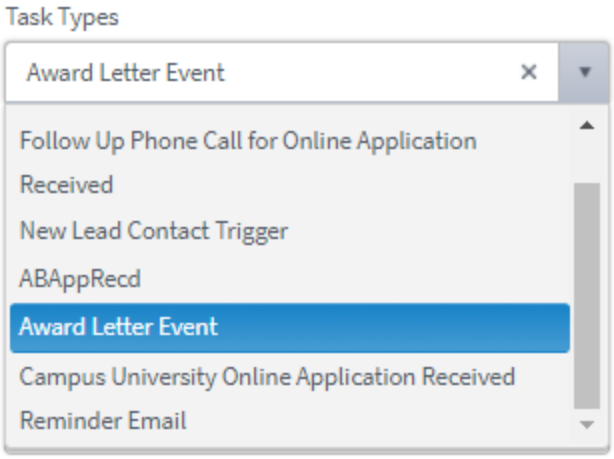
You can use the Drop-down List component to create a Workflow Initialized List of values for a drop-down control. Our example contains a list of task types retrieved from the database using a workflow.

This topic describes only the **Value List** property of the drop-down control. Refer to the Drop-down List topic for the remaining properties.

Control Property Settings



Rendered Component



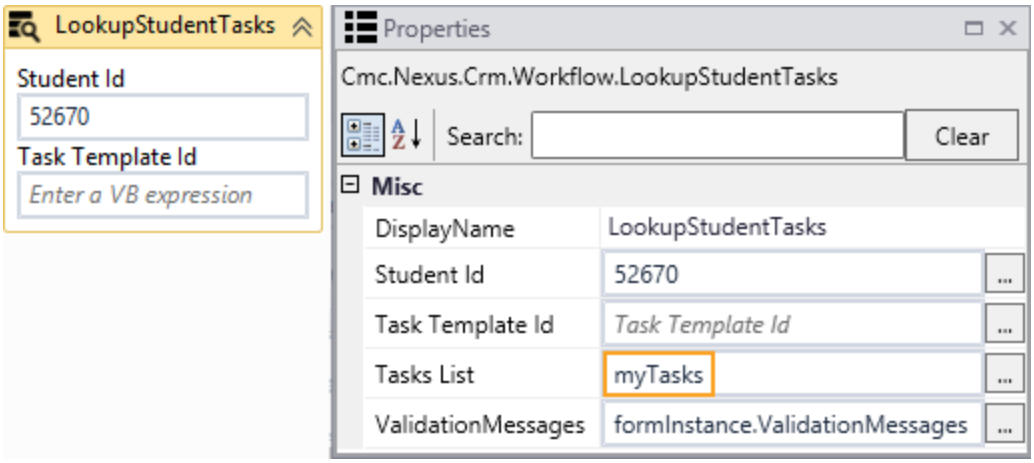
Workflow Arguments

Name	Direction	Argument type	Default value
myTaskSelect	In/Out	String	*Default value not supported*
myTasks	In/Out	TaskEntity[]	*Default value not supported*

Use an argument of type String to capture the selections on the form.
Use another argument to capture the values of the entity Lookup activity.

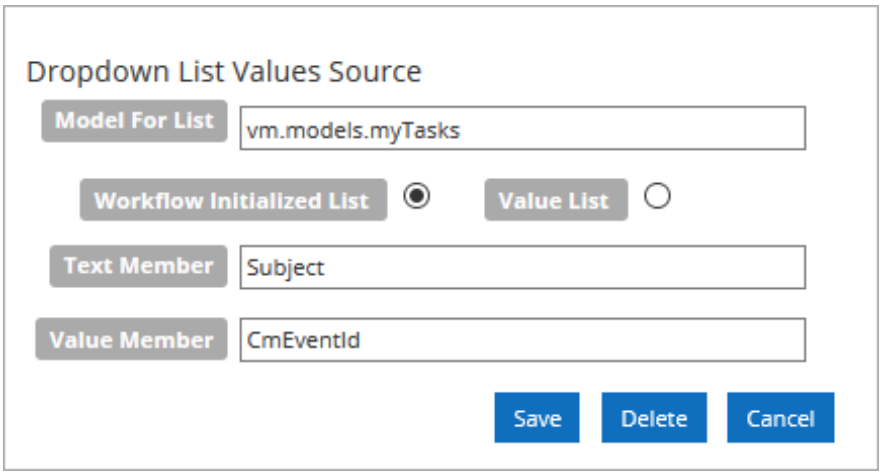
For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

Workflow Activity



Use a Lookup activity to initialize the drop-down list. The out-argument of the activity holds the entity values. The Lookup activity must be placed on a form that precedes the form with the Drop-down list component.

Value List is an optional property. Select the **Edit** button to specify the source of the values to be displayed in the drop-down list.



- In the **Model For List** field, specify the Model property to which the drop-down list will be bound. The Model For List is required for a Workflow Initialized List. The value must start with "`vm.models.`".
- Select **Workflow Initialized List** to bind the values in the list to a Model property value. With this option, the drop-down list values are set in the workflow. The Workflow Initialized List overrides an OData Lookup Query.

  An easy way to create a Workflow Initialized List is to use one of the Lookup activities (LookupStudentTasks, LookupStudentAdvisors, etc.) in Workflow Composer.

  When you are creating a Workflow Initialized List, the simplest object to use is a **NameIdObject**. With an array of these, the Text Member will be Name and the Value Member will be Id, and they will be of type string and integer respectively. If you don't need the Id, it is optional to set it.

  In the workflow, create a variable (myList in this case). DO NOT use an argument or this will not work.

  The type will be NameIdObject[] (array of NameIdObject). You can initialize the object with assign statements, but since variables allow a Default value, use the following example.

  In this example we want to create a list of 2 elements, where Yes is value 1 and No is value 2. Set Default to:

  ```
  new NameIdObject(1){new NameIdObject With { .Name="Yes", .Id=1}, new NameIdObject With { .Name="No", .Id=2}}
  ```

  Note some significant syntax here: the 1 for the array size is VB syntax for an array of 2 elements, with index 0 and 1. There is a dot before each property name in the With sections.

  If you were doing this in assign statements, you could break the statements down as follows:

  ```
  myList = new NameIdObject(1){}  - creates a 2-element array that is empty.
  ```

myList(0) = new NameIdObject - initialize the first array element with a new object, “With” could have been used here instead of the following two assigns.

myList(0).Name = “Yes”

myList(0).Id = 1

etc.

The Default initialization above, while looking more complex, is less wieldy than a few assign statements in the workflow.

To use this, you must expose this as an Out argument of type NameIdObject[]. After you create this argument, you do this with a final assign statement.

```
myArgList = myList
```

The result is that all drop-down list controls that have vm.models.myArgList as the Model For Value List binding (in the popup), will have a Yes/No list. Their Text Member must be Name, and if you use the Value Member, it must be Id.

If any of the following is true, then a SerializableDynamicObject can be used in the same way. It has none of the following limitations.

- You need more than two properties
- The property names cannot be Name and Id
- The types of the property names cannot be string and int respectively.

However, the initialization for the SerializableDynamicObject is considerably more complex to understand to do the same thing as above (with only 2 elements). Here it is:

new SerializableDynamicObject(1){new SerializableDynamicObject With { .DataDictionary = new Dictionary (Of String, Object) From { { "Name", "Yes"}, { "Id", 1} } }, new SerializableDynamicObject With { .DataDictionary = new Dictionary (Of String, Object) From { { "Name", "No"}, { "Id", 2} } } }

You must do this with a variable, and then you must assign it to an argument which is bound to the control.

- In the **Text Member** field, specify the value that will be used as the DataTextField. This is a required field. In a Workflow Initialized List, the Text Member value must match a property in the workflow object collection used to populate it, e.g., `Id`.
- In the **Value Member Field**, specify the value returned when item is selected in the dropdown. It may be the same as Text Member.

Select **Save** to save the data source settings for the list values.

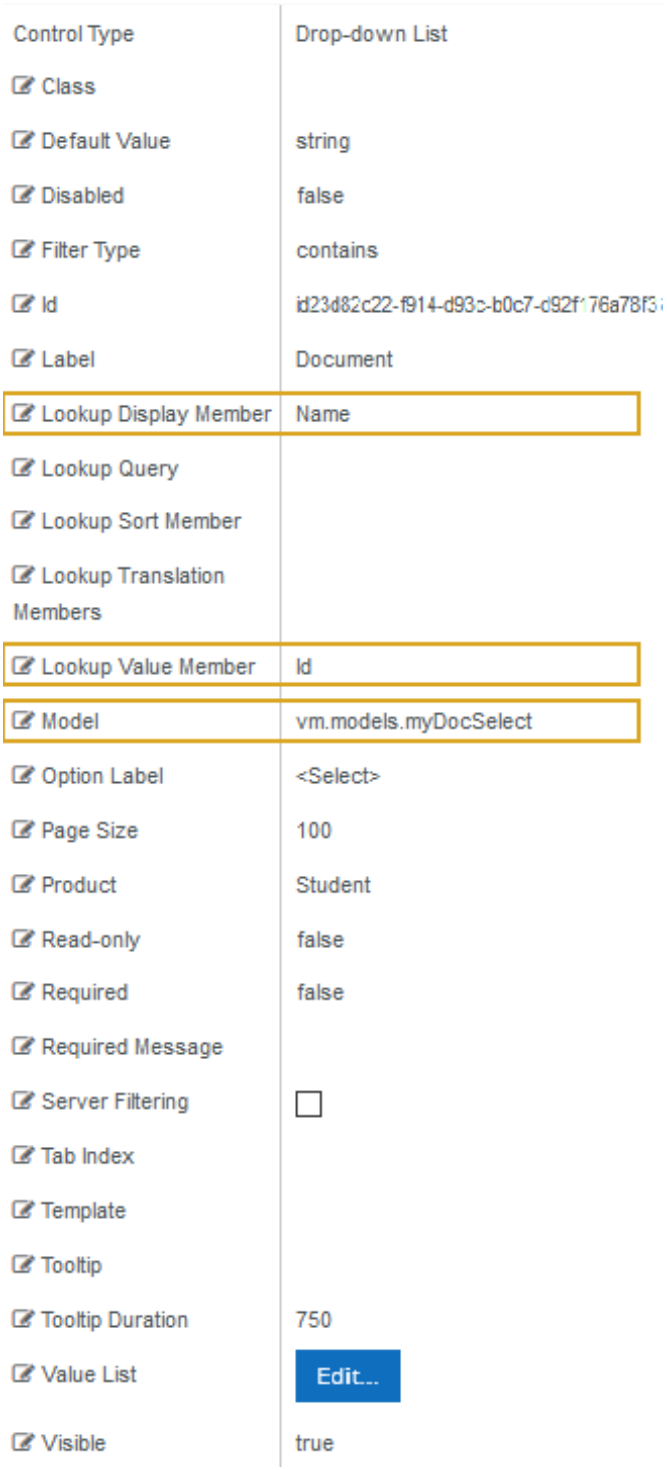
Drop-down List with Workflow Initialized List via ExecuteQuery

You can use the Drop-down List component to create a Workflow Initialized List of values for a drop-down control. The workflow activity ExecuteQuery enables you to retrieve data from any database for display in a drop-down control.

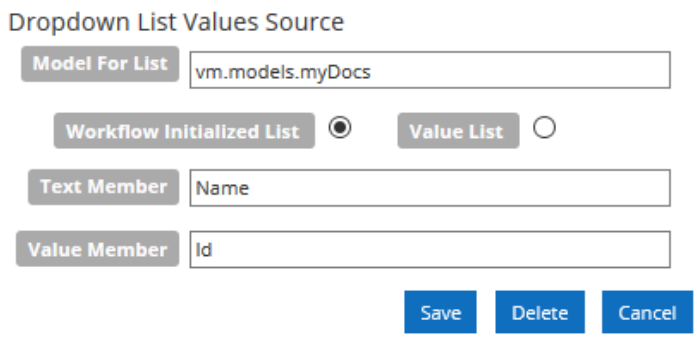
Our example contains a list of documents retrieved from an Anthology Student database using an ExecuteQuery activity followed by a ForEach<> activity, Assign activities, and an AddToCollection<> activity. Details of the attributes used with each activity are provided below.

Refer to the Drop-down List topic for details about the properties of the Drop-down List control.

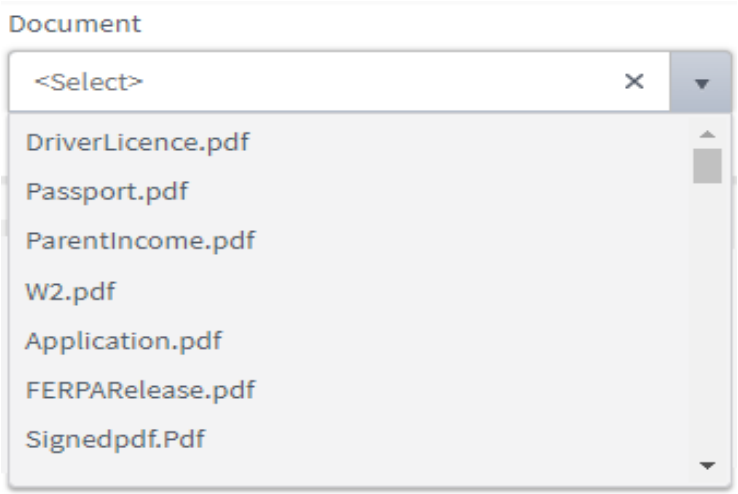
Control Property Settings



Value List is an optional property. Select the **Edit** button to specify the source of the values to be displayed in the drop-down list.



Rendered Component



Workflow Arguments

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event	In/Out	ConstructedEvent	*Default value not supported*
prospectInquiryEntity	In/Out	ProspectInquiryEntity	*Default value not supported*
myDocs	In/Out	NameIdObject[]	*Default value not supported*
myDocSelect	In/Out	Int32	*Default value not supported*

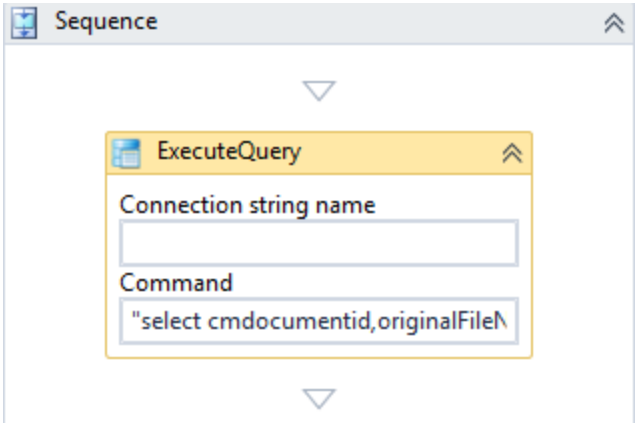
Use an argument of type Int32 to capture the selections on the form.
Use another argument to capture the full list of selections from the ExecuteQuery.

Workflow Variables

Name	Variable type	Scope	Default
renderedFormImage	String	StateMachine	*Enter a VB expression*
docSet	DataSet	StateMachine	*Enter a VB expression*
idValues	List<NameIdObject>	StateMachine	new List(Of NameIdObject)
idList	NameIdObject	StateMachine	*Enter a VB expression*

Workflow Activities

Use an **ExecuteQuery** activity to initialize the drop-down list. The activity must be placed on a form that precedes the form with the Drop-down List component.



The activity executes the following command:

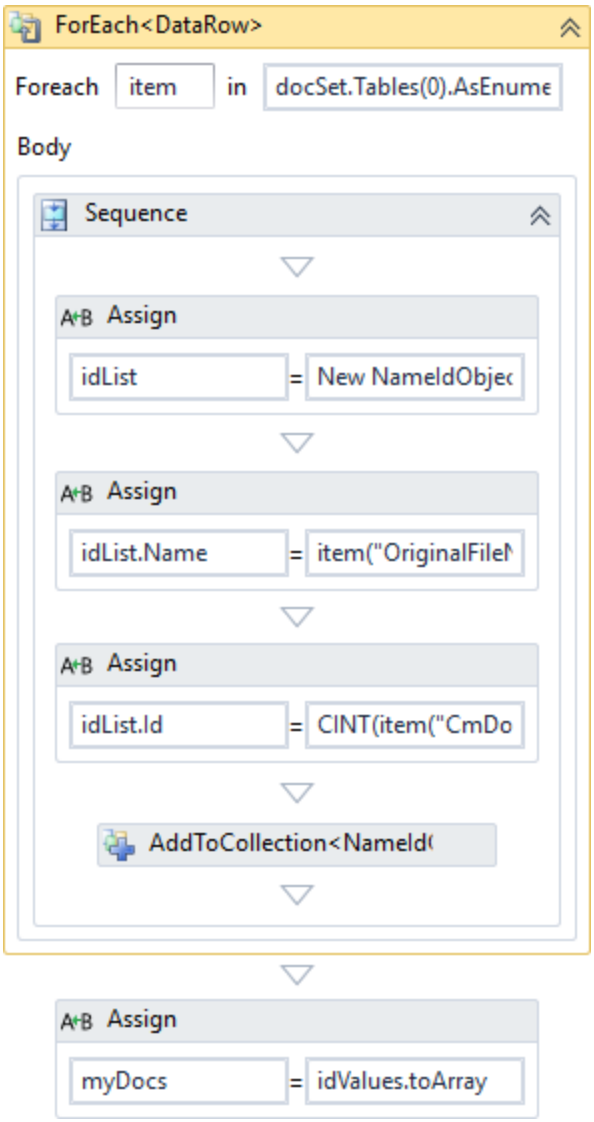
"select cmdocumentid,originalFileName from cmdocument where systudentid = 51850 order by datelstmod desc"

The systudentid value is hard-coded. Specify an appropriate value/variable for your environment.

The connection string name does not need to be specified when the query is executed on the Anthology Student database. When you query a different database, specify the connection string name.

The out-argument of the query holds the docSet variable, i.e., the data set for the drop-down list.

Place the following activities below the ExecuteQuery activity:



The **ForEach<DataRow>** activity uses the TypeArgument `System.Data.DataRow` and the Values property `docSet.Tables(0).AsEnumerable`

The three **Assign** activities have the following attributes:

To	Value
idList	New NameIdObject
idList.Name	item("OriginalFileName").ToString
idList.Id	CINT(item("CmDocumentID"))

The **AddToCollection<NameIdObject>** activity uses the following properties:

Property	Value
Collection	idValues
Item	idList
TypeArgument	Cmc.Nexus.FormsBuiler.Entities.NameIdObject

The final **Assign** activity assigns the document Id values to the myDocs argument.

To	Value
myDocs	idValues.toArray

Drop-down List with Workflow Initialized List and Template

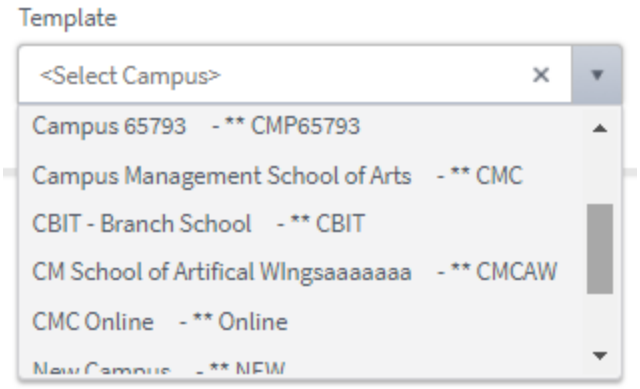
You can use the Drop-down List component to create a Workflow Initialized List of values for a drop-down control. Our example is a drop-down list for the Campus field with formatting using the Template property.

This topic describes only the **Template** property of the drop-down control. Refer to the Drop-down List topic for the remaining properties.

Control Property Settings

Control Type	Drop-down List
Class	
Disabled	false
Filter Type	startswith
Id	4d458df5-78c6-3f65-41d6-26d694a0da5e
Label	Template
Lookup Display Member	Name
Lookup Query	Campuses?$select=Code, Name, Id&$filter=IsActive eq true&$orderby=Name
Lookup Sort Member	Name
Lookup Translation Members	
Lookup Value Member	Id
Model	vm.models.myTemplateCampus
Option Label	<Select Campus>
Page Size	100
Product	Student
Read-only	false
Required	false
Required Message	
Server Filtering	☐
Tab Index	
Template	#: data.Name # - ** #: data.Code #
Tooltip	
Tooltip Duration	750
Value List	Edit...
Visible	true

Rendered Component



Workflow Argument

Name	Direction	Argument type	Default value
myTemplateCampus	In/Out	String	*Default value not supported*

Use a workflow argument of type String to capture the selections on the form.

Template

Template is an optional property. You can use this property to define a custom template to format data in a drop-down list.

In our example, we want the drop-down list for the Campus field to display both the Campus Name and Code instead of only the Name (default).

To retrieve the Name and Code values, we use the following OData query in the Lookup Query property:

```
Campuses?$select=Code, Name, Id&$filter=IsActive eq true&$orderby=Name
```

To display both the Name and Code in the drop-down list, we use the following Template property:

```
<span class='k-state-default'>#: data.Name #     - ** #: data.Code #</span
```

Note the custom span class where the Name and Ccode values are separated by   - ** (space space dash asterisk asterisk).

Drop-down List for Extended Properties

The **ExtendedPropertyDefinitions** entity holds the extended properties that can be configured in Anthology Student under Settings > System > Extended Properties. The ExtendedPropertyDefinitions entity does not provide a built-in drop-down control for the selection of property names. However, you can use the Drop-down List component in Forms Builder with a modified OData query to only return the names of the properties that are assigned to a certain entity.

For example, if you have configured properties for the **Document** entity in Anthology Student, in your OData query you can filter out that **EntityName** to only display the properties for that entity.

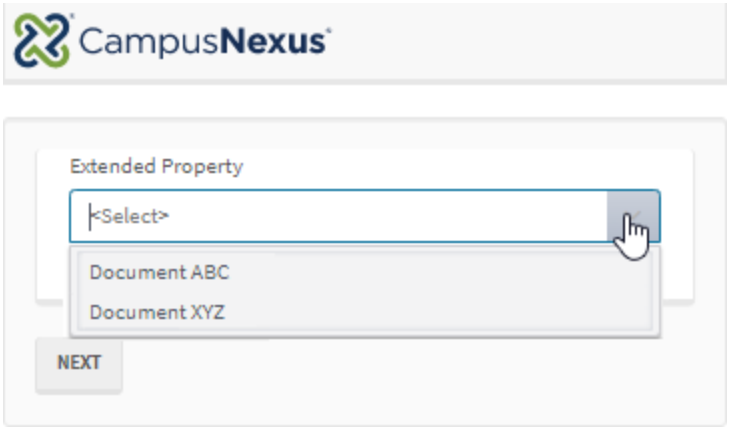
Example

```
ExtendedPropertyDefinitions?$filter=EntityName eq 'Docu-
ment'&$select=Id,Name,EntityName&$orderby=Name
```

The Control Property Settings

Control Type	Drop-down List
Class	
Default Value	string
Disabled	false
Filter Type	contains
Id	id36abc3ef-2e12-cdc7-c4f6-51b8ac7f3e8d
Label	Extended Property
Lookup Display Member	Name
Lookup Query	ExtendedPropertyDefinitions?$filter=EntityName eq 'Document'&$select=Id,Name,EntityName&$orderby=Name
Lookup Sort Member	Name
Lookup Translation Members	
Lookup Value Member	Id
Model	vm.models.extendedPropertyDefinitionEntity.Name
Option Label	<Select>
Page Size	100
Product	Student
Read-only	false
Required	false
Required Message	
Server Filtering	☐
Tab Index	
Template	
Tooltip	
Tooltip Duration	750
Value List	Edit...
Visible	true

Rendered Component



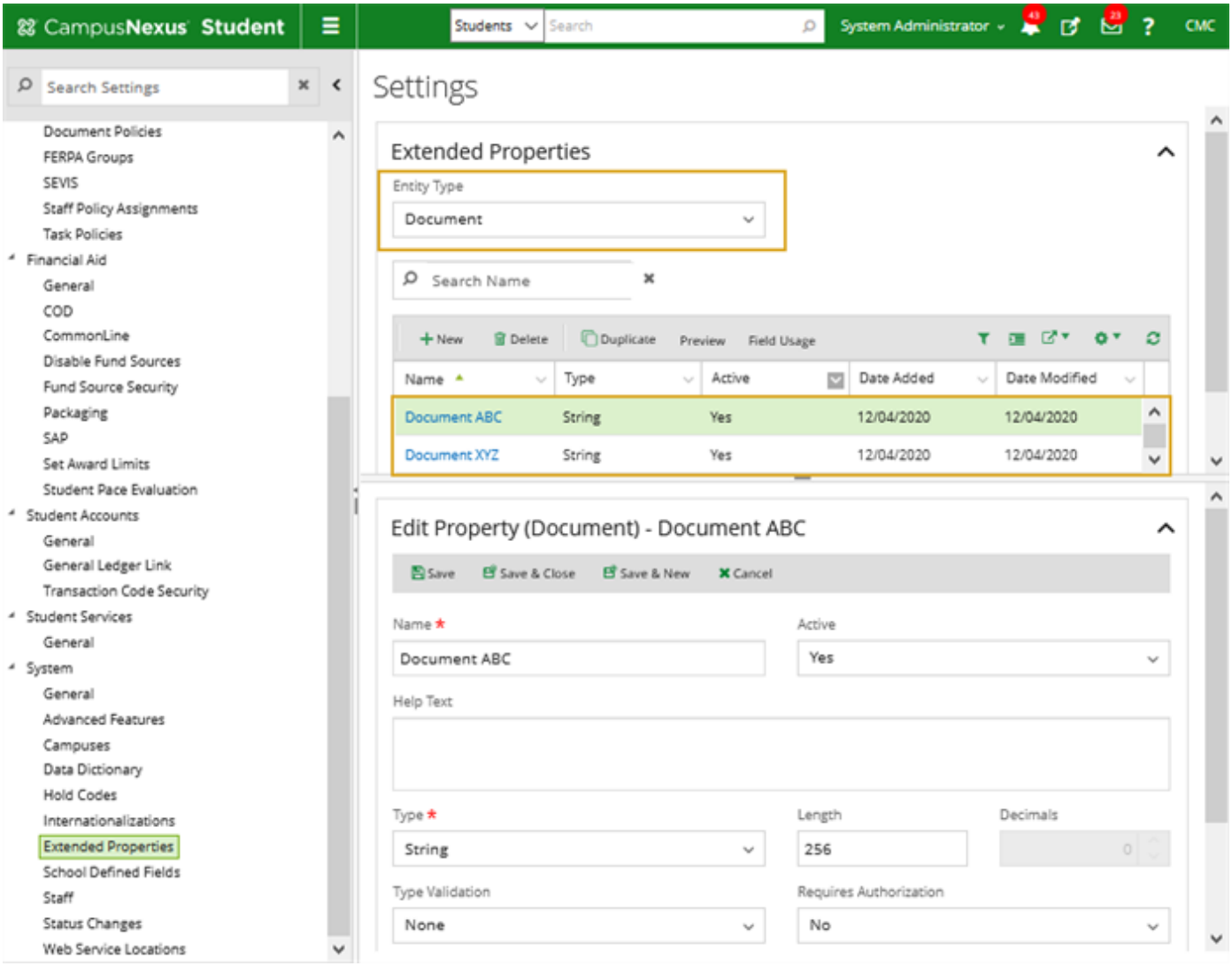
The rendered will display the properties for available for selection.

Workflow Argument

Name	Direction	Argument type	Default value
extendedPropertyDefinitionEntity	In/Out	Nullable<Int32>	*Default value not supported*

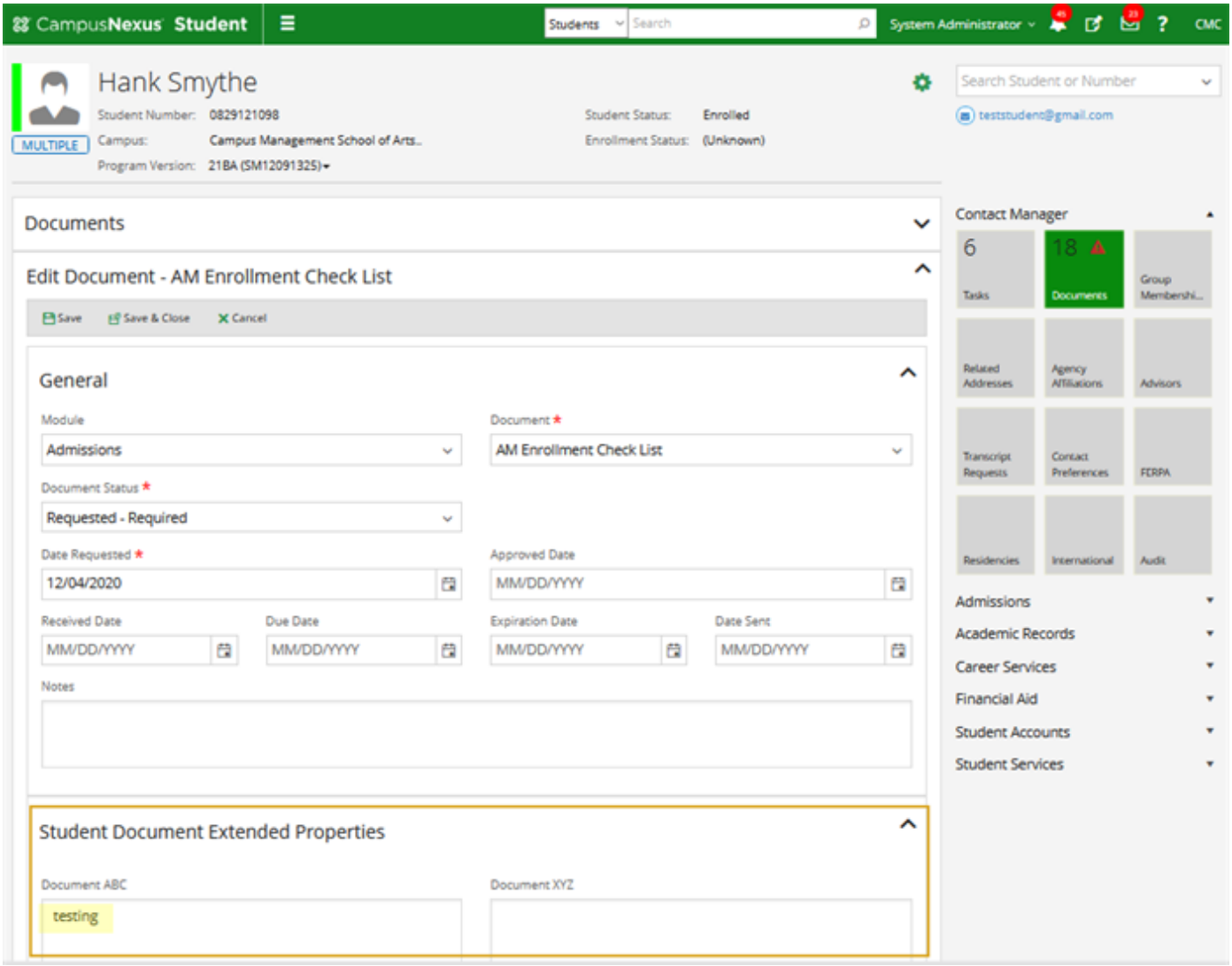
For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

In Anthology Student, the Extended Properties for Entity Type "Document" are configured as shown below.

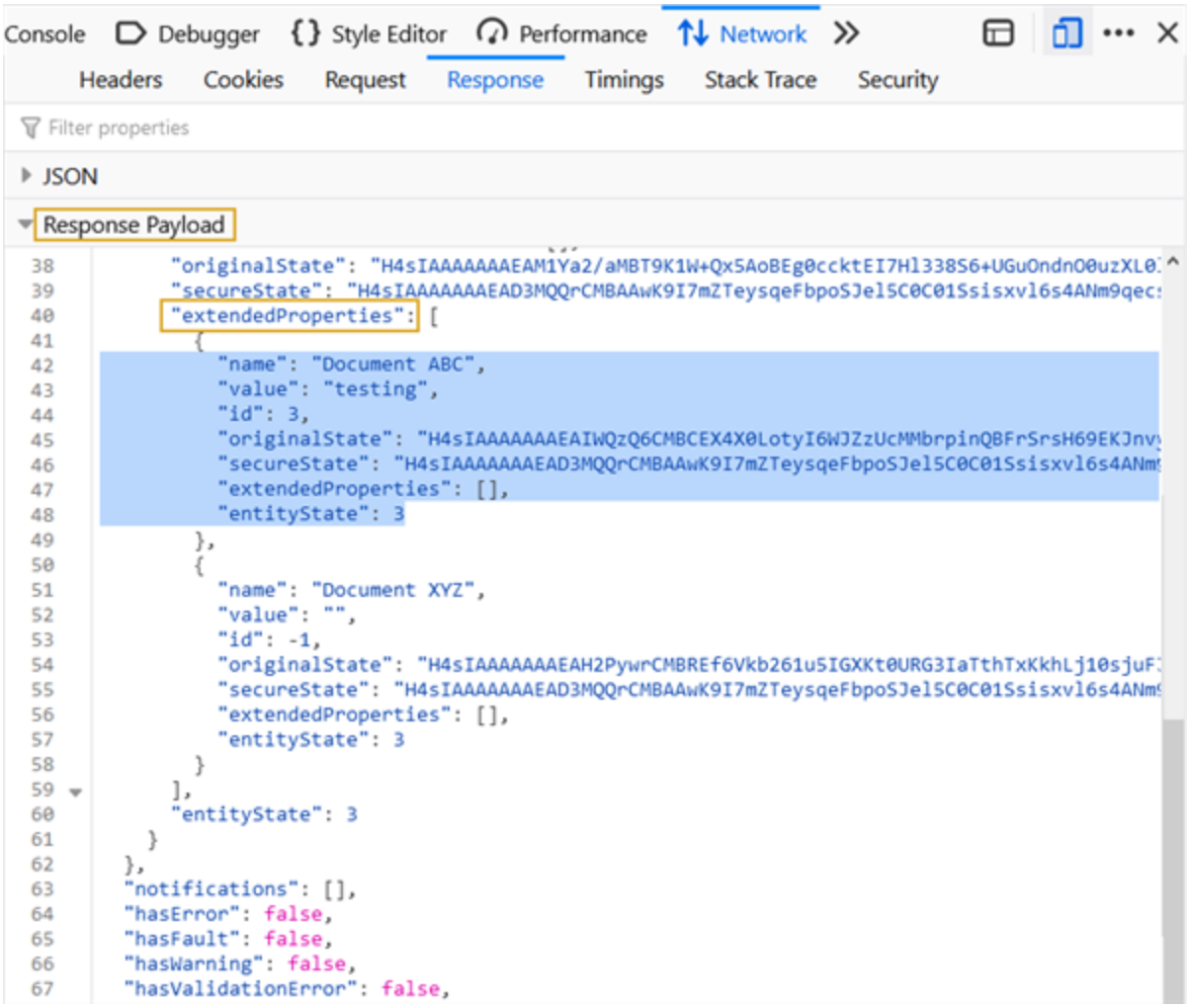


In workflow, you can get the value selected from that entity and then assign it to the student on the Document Entity (extended property/value).

The image below shows extended properties section on the Documents page for a student. In our example, the extended property "Document ABC" has the value "testing".



When the property updated, the Response Payload shows the extended properties/values for the Document entity.


Console
Debugger
Style Editor
Performance
Network
Headers
Cookies
Request
Response
Timings
Stack Trace
Security
Filter properties
JSON
Response Payload
"originalState": "H4sIAAAAAAAEAM1Ya2/aMBT9K1W+Qx5AoBEg0ccktEI7Hl338S6+UGuOndnO0uzXL0
"secureState": "H4sIAAAAAAAEAD3MQQrCMBAAwK9I7mZTeysqeFbpoSJel5C0C01Ssisxvl6s4ANm9qec
"extendedProperties": [
{
"name": "Document ABC",
"value": "testing",
"id": 3,
"originalState": "H4sIAAAAAAAEAIWQzQ6CMBCEX4X0LotyI6WJZzUcMMbrpinQBFrSrsH69EKJnv
"secureState": "H4sIAAAAAAAEAD3MQQrCMBAAwK9I7mZTeysqeFbpoSJel5C0C01Ssisxvl6s4ANm
"extendedProperties": [],
"entityState": 3
},
{
"name": "Document XYZ",
"value": "",
"id": -1,
"originalState": "H4sIAAAAAAAEAH2PywrCMBREf6Vkb261u5IGXKt0URG3IaTthTxKkhLj10sjuF
"secureState": "H4sIAAAAAAAEAD3MQQrCMBAAwK9I7mZTeysqeFbpoSJel5C0C01Ssisxvl6s4ANm
"extendedProperties": [],
"entityState": 3
}
],
"entityState": 3
}
},
"notifications": [],
"hasError": false,
"hasFault": false,
"hasWarning": false,
"hasValidationError": false,

Expand/Collapse Panel

You can use the Expand/Collapse Panel component to add an interactive control to a form that allows the user to expand or collapse content. You can style the control as a text link or as a plus/minus button that is center, left, or right aligned. Custom styling is also supported. You specify the expanded/collapsed content in the HTML property.

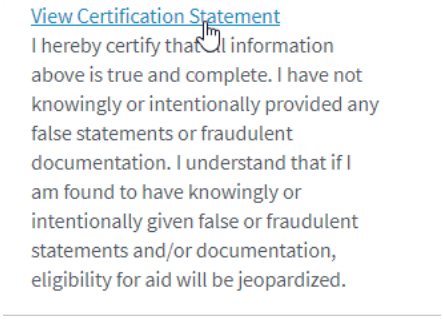
Properties

Control Property Settings

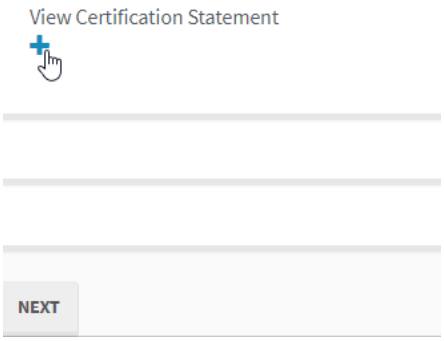
Control Type	Expand/Collapse Panel
Class	
Expand Label	View Certification Statement
HTML	<span translate>I hereby certify that all information above is true & complete. I have not knowingly or intentionally provided any false statements or fraudulent documentation. I understand that if I am found to have knowingly or intentionally given false or fraudulent statements and/or documentation, eligibility for aid will be jeopardized.
Id	id9d6aebc4-816d-948b-02fc-8b4044cb3bf9
Is Expanded	false
Label Alignment	left
Label Type	link
Tab Index	
Visible	true

Rendered Component

Label Type = link



Label Type = plus/minus



- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Expand Label** is the value displayed in the label for the clickable expand.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- The **HTML** property enables you to format rendered output using a subset of standard HTML markup as a fragment of an HTML page. That is to say, <!DOCTYPE html>, <html>, <head>, <body> and <form> tags are not appropriate in an HTML fragment. While they may not harm the page, they do have the potential to create silent Renderer errors or cause the page rendering to fail completely. The HTML validation parser will point out errors in the HTML fragment and mark them as a warning but will not attempt to enforce rules. Warnings should be corrected to avoid unexpected results.

  An example of an HTML fragment is:

  <h2 class=”myclass”>Campus View</h2>

  <img src="IMG_2671.jpg" alt="Campus View" style="width:304px;height:228px;">

Also possible are <script> and <style> fragments. This allows a great deal of customization. A model value can be addressed in JavaScript with “window.vmModelsRef”. If you had an argument in a workflow “myKey”, which would also be a model value “vm.models.myKey”, then in external HTML JavaScript this can be addressed with “window.vmModelsRef.myKey” or “window.vmModelsRef[‘myKey’]”.

Similarly, parameters for the Renderer URL that are addressed with formInstance.QueryParams.DataDictionary(“myKey”) or special case formInstance.QueryParams.DataDictionary (“addonQueryParams”) in the workflow (refer to Renderer URL Query Parameter), can be addressed in external HTML JavaScript as “window.vmQueryParamsRef.myKey” or “window.vmQueryParamsRef [‘myKey’]. This would allow you to pass information in the Renderer URL to your custom JavaScript (or even to a custom style via a binding). Of course, depending on how it is to be used, make sure your JavaScript and/or workflow validates the information passed, or this could be a security risk.

- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Is Expanded** sets the control to be collapsed or expanded.
    - Must be true or false, or a binding beginning with "vm.models.".
    - A property array string index requires single quotes, e.g. vm.models.xxx.CustomProperties['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g. vm.models.myvalue==7 (>,<,!-!=,==,>=,<=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)

  If you want the content of the Expand/Collapse Panel component to be displayed in View Summary and/or PDF, set the “Is Expanded” property to true.
- **Label Alignment** aligns the label. Select from the drop-down list. The options are center, left, and right.
- **Label Type** set a label type. Select from the drop-down list. The options are link and plus/minus.
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
    - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.

- A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
- A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

  The tab index value should not match another control's tab index.
- A blank value (default) will not add a tab index in the HTML.

For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.

- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)

File Upload

You can use the File Upload component to upload files in a form.

When files are uploaded to Forms Builder, they are stored temporarily in a table named *UploadStorage*. To retrieve the files and use them in a workflow, use the GetAttachments activity.

You can retry uploading a file using the same File Upload component when the webpage was reloaded, or the network connection was interrupted during the first try. In previous versions, an error was displayed.

Enhancements in Forms Builder 3.5

In Forms Builder 3.5, the code for the File Upload component was completely revised to provide new functionality such as the "Required", "Required Message", and "Tab Index" properties. When you drag the File Upload component into the Layout pane, the underlying functionality will be that of the new component. Existing forms that already include the File Upload component continue to execute the previous version of the code.

For existing forms, if you want to use the “Required” property, you need to:

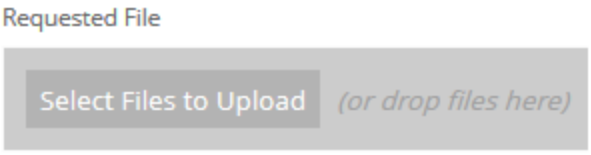
- Remove the File Upload component from the previously created form and then re-drag the component into the Layout pane.
- Update the GetAttachments activity in the workflow with the new ControlIdentifier value.

Properties

Control Property Settings

Control Type	File Upload
Allow Multiple Files	false
Class	
Extensions Allowed	pdf,png,jpg,docx,doc,gif,bmp
Id	ida7e9dba0-a82f-6d15-4e02-1c701a55f9c6
Label	
Max Size Allowed	0
Required	false
Required Message	At least one file must be uploaded
Select Message	Select Files to Upload
Tab Index	
Tooltip	
Visible	true

Rendered Component



- **Allow Multiple Files** allows upload of multiple files.
    - If this value is bound, it must start with "vm.models."
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.

- **Extensions Allowed** is the list of allowed extensions for upload, in comma separated format, such as pdf, docx, png. This must be a comma separated list of extensions (no spaces and no periods).

  If you use dynamic binding for the Extensions Allowed property, for example, `{{vm.models.myFileTypes}}`, no server-side validation can be performed for potential security risks (such as uploading an .exe file). It is therefore your responsibility to perform these checks in the workflow.

- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
  - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
  - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
  - Binding is not supported for this property.

- **Label** is the value displayed in the label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Max Size Allowed** is the maximum allowable size of an uploaded file in bytes. 0 (default) means unlimited.
  - If this value is bound, it must begin with {{vm.models. and end with }}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

  Notes
  - If this control is bound to a model, depending on the product, the Max Size allowed may be limited by the service used to persist the attachment.
  - If you use dynamic binding for the Max Size Allowed property, no server-side validation can be performed for potential security risks (such overloading the server with very large files causing a "Denial of Service Attack"). It is therefore your responsibility to perform these checks in the workflow.

- **Required** makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.
  - Can be bound to a workflow argument or another control's value.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].

- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
- If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

- **Required Message** is optional. It overrides the default "Required" message.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Select Message** is the text to display in the upload button.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
  - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
  - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
  - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

    The tab index value should not match another control's tab index.
  - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.

  When Tab Indexes are specified on a form with File Upload component, after the user uploads a file, the focus will shift to the default tabs (e.g., buttons) at the end of the form. The Shift+Tab key combination can be used to go backwards in the tab order to fill in any other items after the file upload or any time an item on the form is missed.
- **Tooltip** is the value to display when hovering over the control's label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Visible** makes the control visible or hidden.

- Can be bound to a workflow argument or another control's value. This property is dynamic.
- A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)
- When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

Grid

You can use the Grid component to implement a grid or table within a rendered form. The property settings for the Grid component allow you to define the table structure and presentation, that is, the number of columns, column headings, data types, formatting, etc. Additional settings allow you to specify if the grid will be read-only or editable.

All contents of the grid should be bound via the Model property so that it will be displayed correctly when converted to PDF. This applies to a PDF file saved as attachment to a document tracking record or a PDF file displayed in e-sign processes. Use the PrintUrlToPdf activity to create the PDF.

To feed a grid with data, you can use the Model, Model Data, or OData Query properties:

- The Model property can be used to populate the grid with data from Anthology Student or CampusNexus CRM entities.
- The Model Data property can be used to define the initial grid data as a JSON string.
- The OData query property can retrieve data from the CampusNexus CRM or Anthology Student database. You can then configure the Grid to display specific columns and perform sorting, paging, and filtering operations via its built-in property settings.



To handle add, edit, and delete cases, Grids should be workflow initialized.
Added rows can be determined by their Id value.
ODdata queries will not return the entityState and are limited to read-only grids.

For details about all available grid properties, refer to Grid Property Settings and Grid Columns Properties.

For use case examples, refer to:

- Grid Initialized via OData Query
- Grid Bound to an Entity
- Grid Bound to Custom Model Data (non-Entity)
- Grid Bound to Results of ExecuteODataQuery
- Grid Populated by ExecuteQuery
- CRM Grid for One-to-Many Relationships

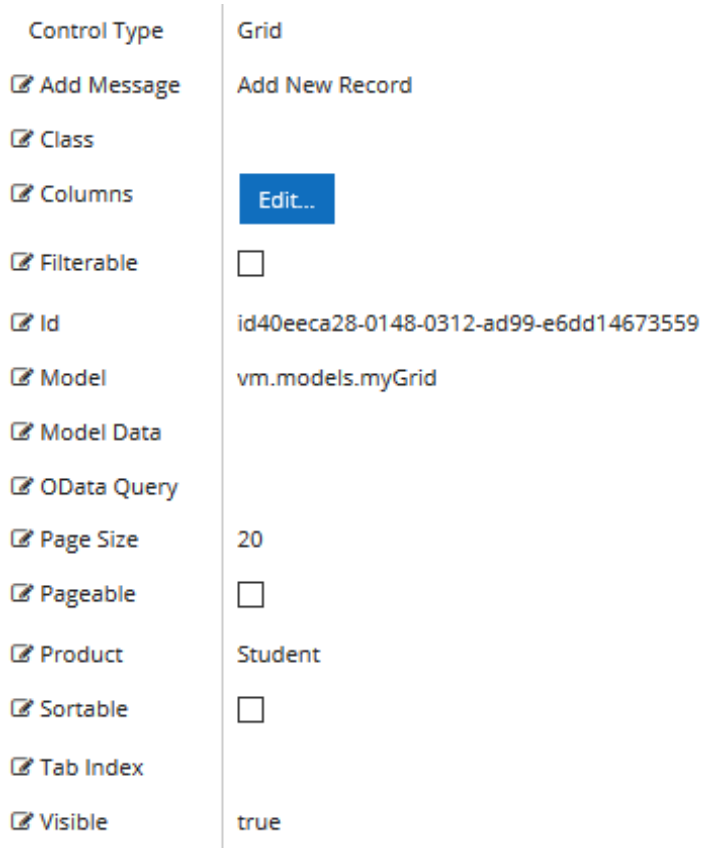
For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

Grid Property Settings

The Property Settings pane of the Grid component enables you to specify properties that apply to the entire grid and properties that apply to specific columns within the grid (refer to Grid Columns Properties).

The image below shows the default properties displayed when the Grid component is first added to a form. Note that the Model, Model Data, and OData Query properties are not prepopulated. Depending on the data source for the grid and depending on whether the grid data need to be bound, you must specify either Model, Model Data, or OData Query, or a combination of these properties.

Control Property Settings



Properties

- **Add Message** is the text used to instruct the user to add a new record to the grid on the form.
    - If this value is bound, it must begin with {{vm.models. and end with }}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Columns** — Specifications of the columns that match the data properties to be bound in the grid. Refer to Grid Columns Properties.
- **Filterable** allows the data to be filtered. This property is not bindable. Default: false
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters). Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id. Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces. Binding is not supported for this property.
- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.
    - The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
    - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
    - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
    - The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
    - If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

  If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

  If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

  Construction of the model in the workflow is done by assigning data from a provider.

  The Model is passed to the workflow as an argument of type SerializableDynamicObject[]. This argument type will hold the data entered in the grid.

  Examples:

    - Grid Bound to an Entity
    - Grid Bound to Custom Model Data (non-Entity),
    - Grid Bound to Results of ExecuteODataQuery,
    - CRM Grid for One-to-Many Relationships.

- **Model Data** — Defines the initial data as a JSON string.

  Example: Grid Bound to Custom Model Data (non-Entity).

  When the Model Data property is used in conjunction with the Locale component and the user navigates back to select a different locale, the data displayed in the sequence will not be updated to reflect the new locale selection.

  The Model Data property is not applicable for a grid bound to an entity.

- **OData Query** — OData query that overrides Model or Model Data if supplied.

  Example

  Grid Initialized via OData Query

  OData Query is not used for editable grids.

- **Page Size** — If Pageable is true, Page Size will set the server-side paging size for the grid. Default: 20
- **Pageable** — If set to true, the grid will display a pager. This property is not bindable. Default: false.
- **Product** indicates the product from which OData query results are returned. Select from:
  - Student
  - CRM
  - Occupation Insight

  The selected product must be configured in the <products> section of the Renderer web.config file.

  The default Product value will be "Student" if "Student" is selected in the <Select Provider> list on the Fields tab.

  The default Product value will be "CRM" if "CRM" is selected in the <Select Provider> list on the Fields tab.

  Select "Occupation Insight" in the Product property if the source of the query will come from a different data source other than Student/CRM. For more information, refer to Build Queries for Occupation Insight.

  A form can have multiple controls that retrieve data from different providers. For example, a form can have a control that is populated by a query to the Student database. The same form can have another control that retrieves data from Occupation Insight.

  Specify the query to retrieve data from the selected provider using the Lookup Query or OData Query property (as applicable for the control). The query contains only the URL specific part of an OData URI. The Base URL and Product will be supplied by the configuration.

- **Sortable** — Allows data to be sorted. This property is not bindable. Default: false.

Note: When paging, sorting and add record are enabled in a Grid component, if a user has paged or sorted the grid, the add record operation will have inconsistent results because the grid display is being reorganized.

- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
    - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
    - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
    - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

      The tab index value should not match another control's tab index.
    - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.
- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
    - When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

For more information, refer to Telerik documentation:

- Grid / Basic usage
- kendo.ui.Grid - Configuration

Grid Validation

The Grid component does not support a "Required" validation on the client-side. However, this can still be achieved via workflow logic to verify a value has been set and if not, display a custom validation error.

To check if a numeric field within a grid has a value, use an If activity with the condition "Is Nothing".

Also use If activities with appropriate conditions to check if a numeric value is in a range since numeric values within a grid do not have range properties.

Grid Columns Properties

The Property Settings pane of the Grid component includes an Edit button that brings up an editor for properties associated with the columns and rows of a grid.

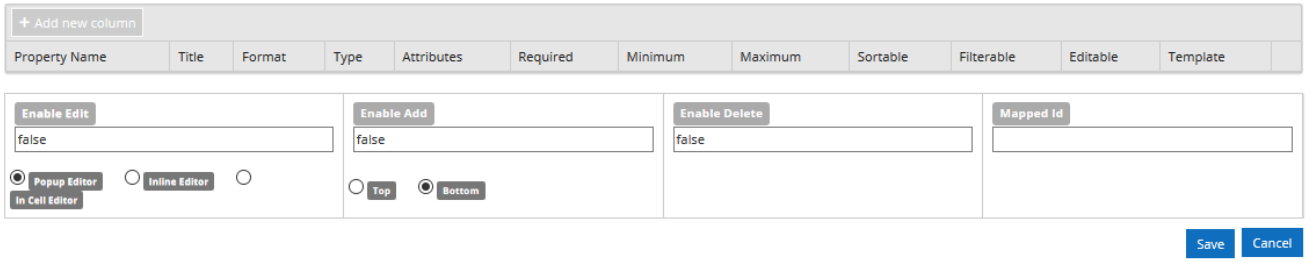
The properties available in the editor override the property settings for the Grid component. For example, when the *Sortable* property for the Grid component is set to true, you can set the *Sortable* property for a column X to false so that column X is not sortable while all other columns in the grid are sortable. To change the column title, add formats, attributes, or change the sortable and filterable properties, you need to specify this for each column.

When editing Columns property settings, Google Chrome and Microsoft Edge browsers will provide the best experience.

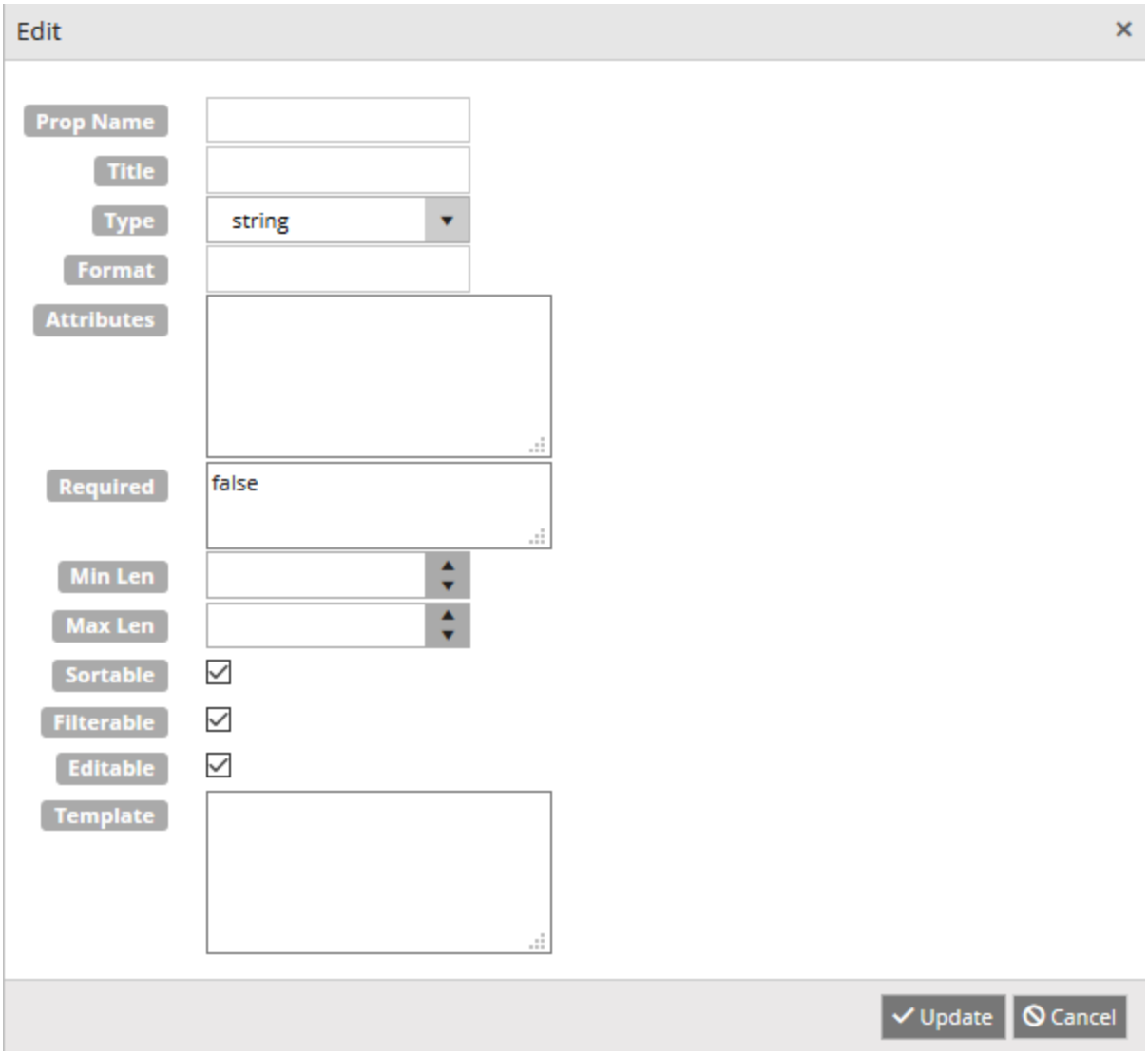
To specify properties using the Columns editor:

1. In the Property Settings pane of the Grid component, select the **Edit** button next to the Columns property setting.

   The editor is displayed in a popup window. It contains properties for the columns and rows in a grid. Place the cursor over the fields to view the tooltips.



2. To specify column properties, select **Add new column**. The Edit window is displayed.



3. Specify the properties for the newly added column in the Edit window.

Columns

a. **Property Name**

Required (Kendo "field" name). Property Name must be a simple case sensitive string starting with a letter, and followed by numbers, letters, or underscores. It must match a property in your bound data, i.e., in the OData query or Model.

Rows of properties can be dragged in the grid to reorder them.

The Property Name and the Mapped Id must be different. For more information, refer to Mapped Id.

b. **Title**

The title of the grid column. If no titles are added, the title row will not be displayed.

c. **Type**

If the data is a date or a number, and the formatting is specified for either, this must be set to the type the format acts on. The default is "string" and is not saved. The Type options are:

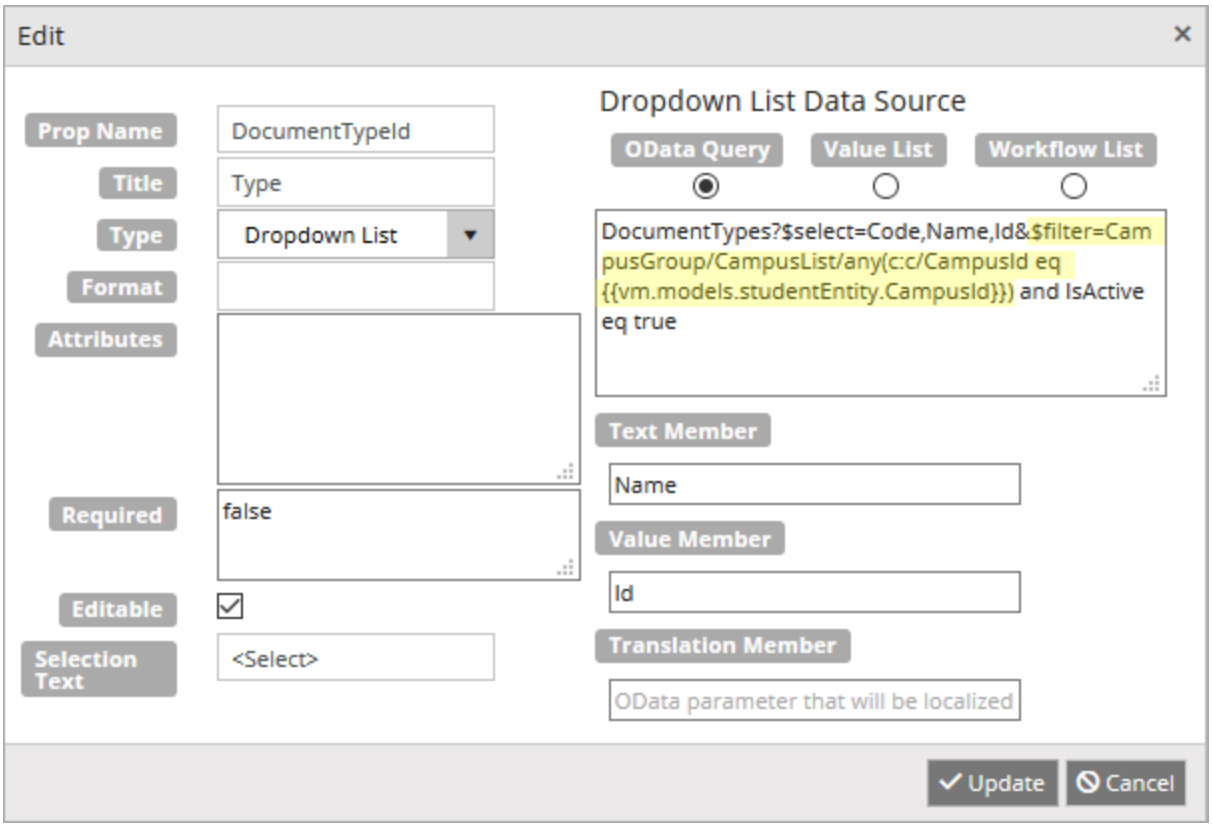
- **string**
- **number**
- **boolean**
- **date**
- **Drop-down List**
- **Text Area**

The data source for the possible values in a **Drop-down List** *Type* can be an **OData Query**, a **Value List**, or a **Workflow List**.

Example 1: OData Query as Data Source for a Drop-down

In this example, an **OData Query** for Document Types populates the Drop-down List. The values specified in the **Text Member** and **Value Member** fields match parameters in the OData Query.

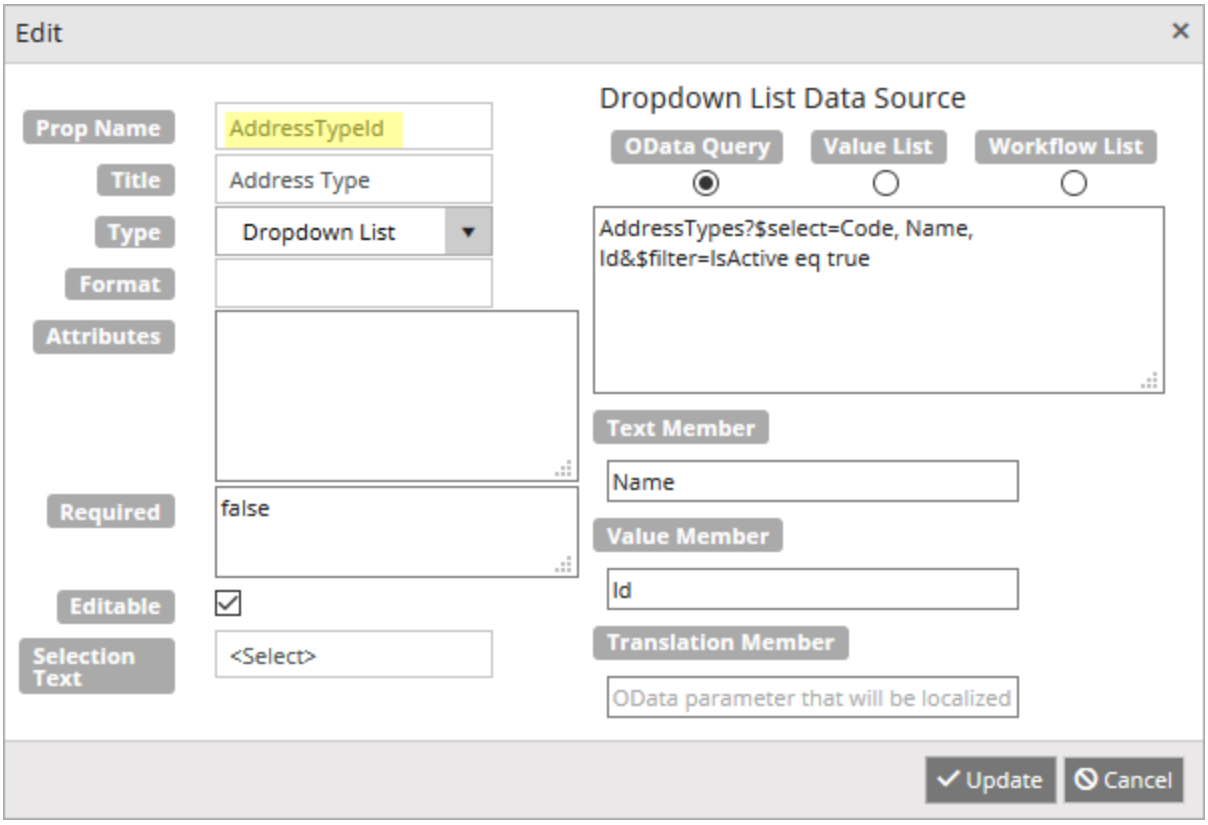
The query in this example uses a filter based on another value on the form (cascaded queries). Drop-down lists will show *Pending* until a Campus is selected.



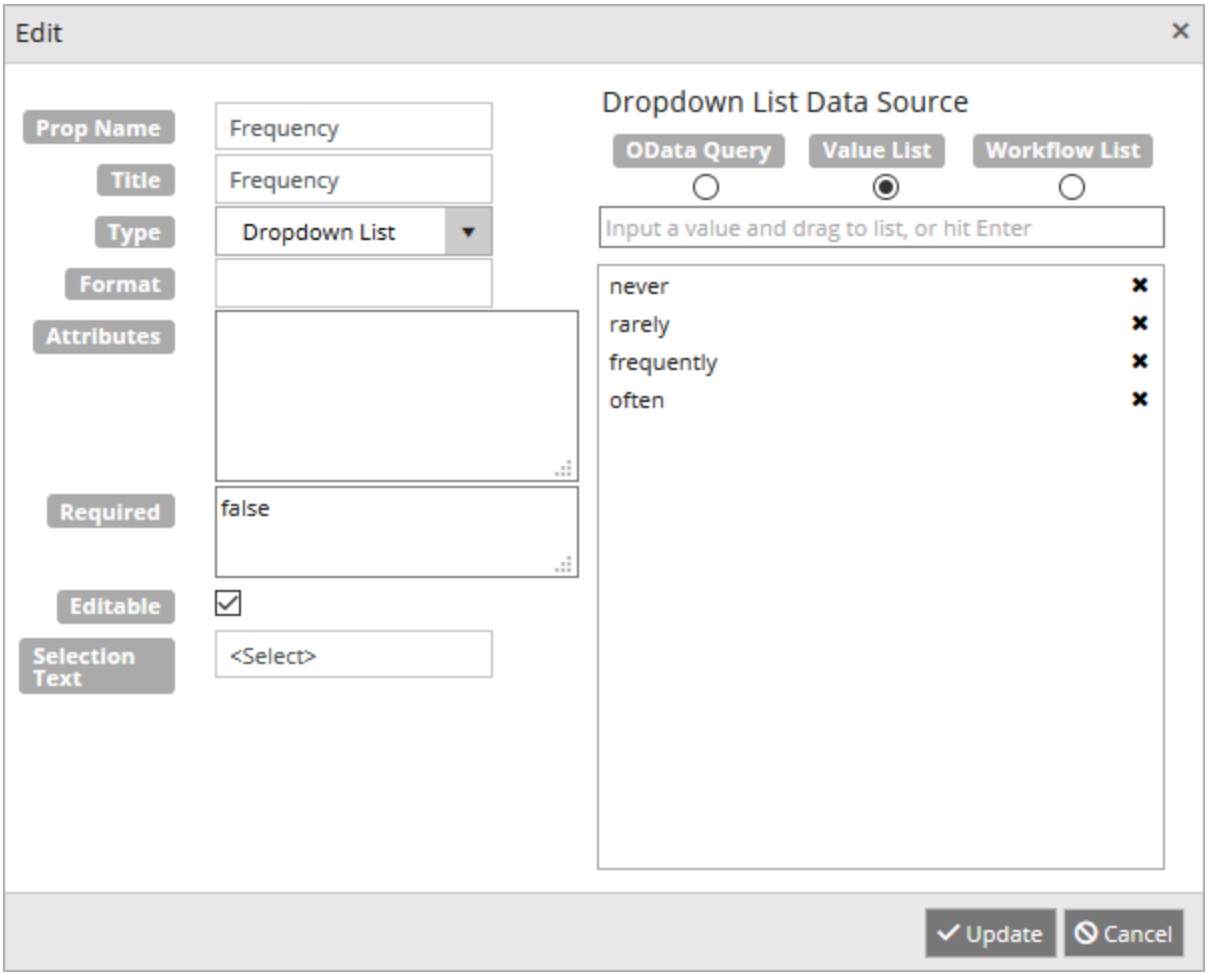
When the *Editable* check box is selected, the *Selection Text* field appears, and you can edit the selection text. When the *Editable* check box is cleared, the *Selection Text* field disappears, unless you are using Firefox. In Firefox, the *Selection Text* field does not toggle. You can still select or clear the *Editable* check box, then close the window, open it again, and set the *Selection Text* field as needed.

Example 2: OData Query for Id Value to Name Conversion in a Drop-down

A grid bound to an entity (e.g., StudentRelationshipAddressEntity) contains an *Id* value (e.g., AddressTypeId of 5), but in the grid you want to display the *Name* for that Id value (e.g., Parent). You can use the drop-down list to convert Id 5 to name Parent.

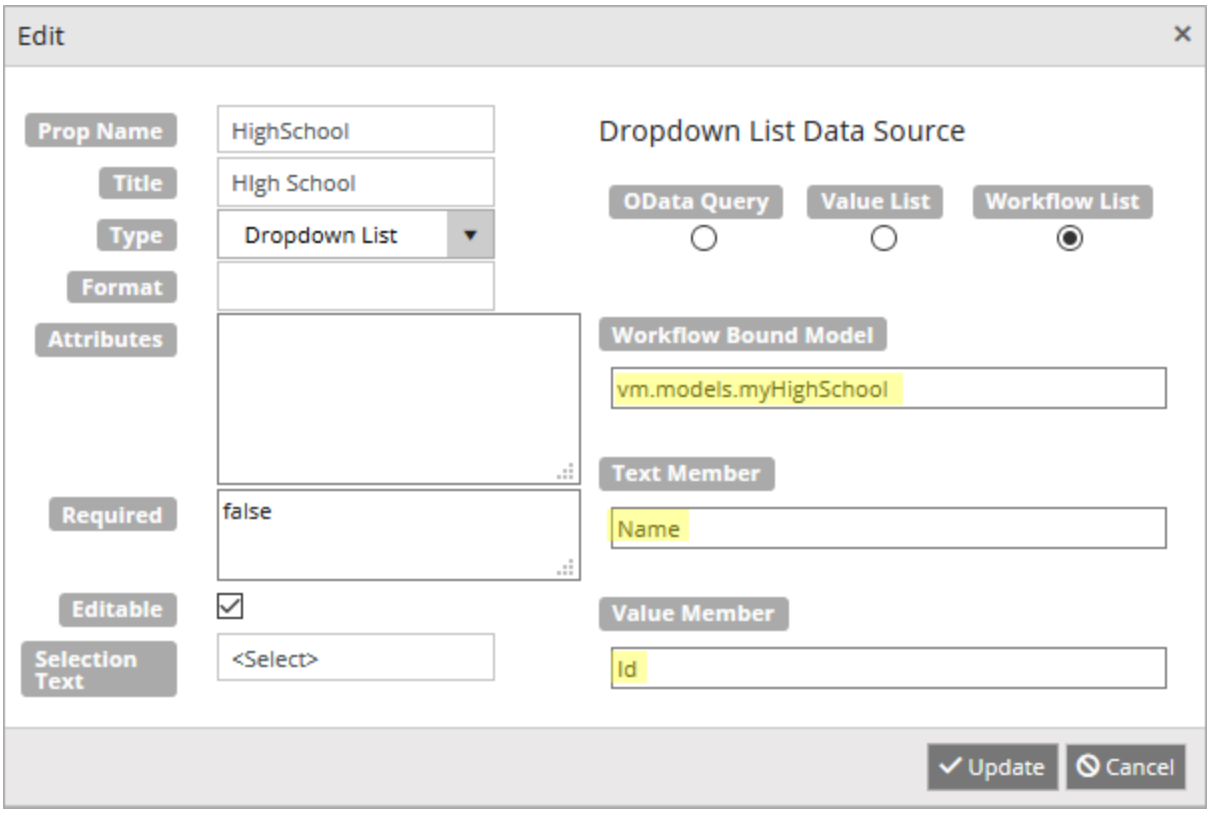


Example 3: Value List as Data Source for a Drop-down



When the *Editable* check box is selected, the *Selection Text* field appears, and you can edit the selection text. When the *Editable* check box is cleared, the *Selection Text* field disappears, unless you are using Firefox. In Firefox, the *Selection Text* field does not toggle. You can still select or clear the *Editable* check box, then close the window, open it again, and set the *Selection Text* field as needed.

Example 4: Workflow List as Data Source for a Drop-down



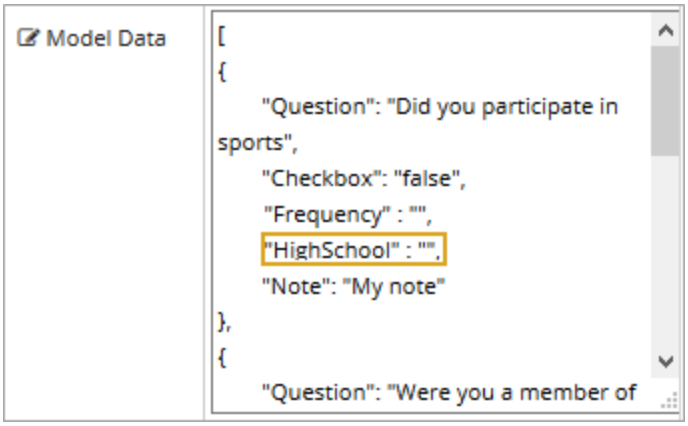
When you select Workflow List as the data source for a drop-down list, specify the values for:

- Workflow Bound Model

  By default, the model binding (e.g., vm.models.myHighSchool) is created as an argument of type string. You will have to update the argument type based on the type of object in your form.

- Text Member = Name
- Value Member = Id

In our example, the Model Data property initializes the Value Member (the Id property) of the drop-down list. Since the "HighSchool" field is left blank, the default <Select> will be displayed, otherwise, if the value is in the Workflow Bound Model list created in the workflow, the Text Member corresponding to that value will be selected.



In the workflow for this example, the Welcome form has a LookupHighSchools activity that returns an array from the High School field in the database. The out argument named "myHighSchool" is of type Cmc.Nexus.Models.Admissions.HighSchool[].

When a form with Workflow List as the data source for a drop-down list is localized, the TranslateText activity needs to be added to the workflow.

Example 5: Drop-down Selection Text Initialization

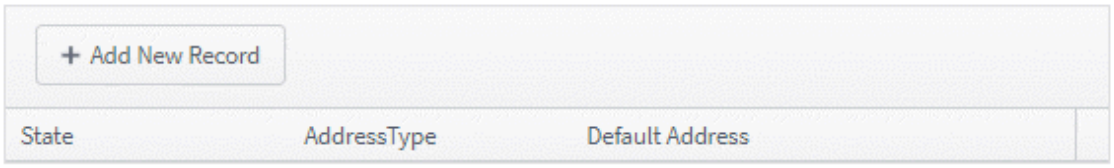
- If a form contains a grid **without Model Data**, Forms Renderer displays only the column titles without any rows.

  The example below shows a grid with three columns: drop-downs for State and Address Type and a Boolean (check box) for Default Address. The default Selection Text <Select> is used for the drop-downs. The Inline Editor is specified for the row.

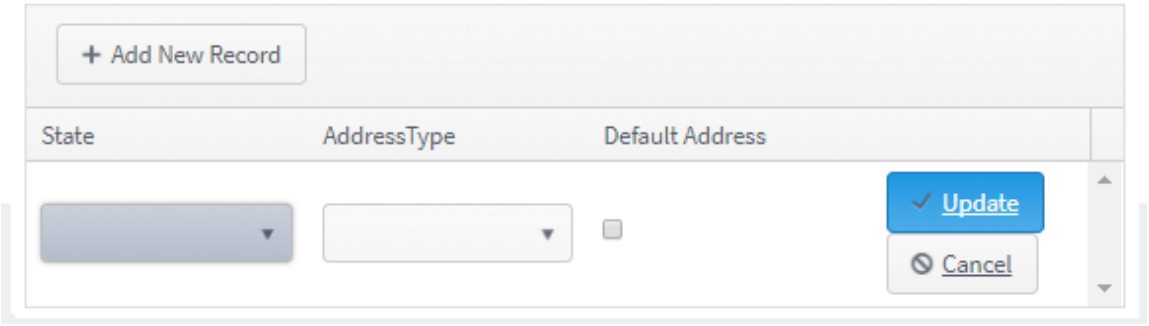
  The data sources for the drop-downs are OData Queries:

  UsaStates?$select=Code,Name,Id&$filter=IsActive eq true

  AddressTypes?$select=Code,Name,Id&$filter=IsActive eq true



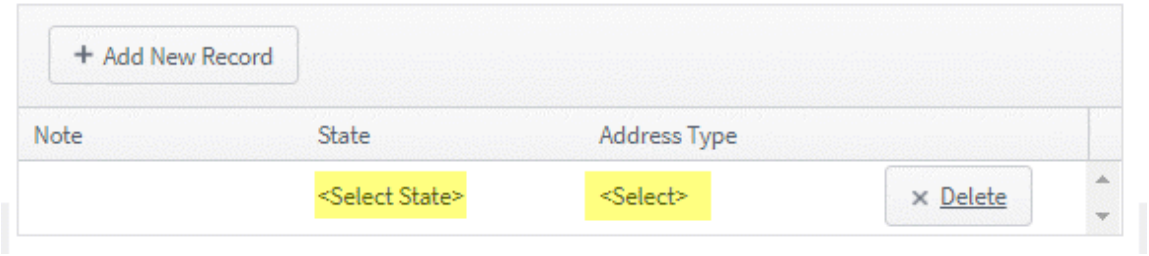
  When the user selects *Add New Record*, a row is added and the values for the drop-down lists are populated. The drop-down list Selection Text is displayed after the user selects a field.



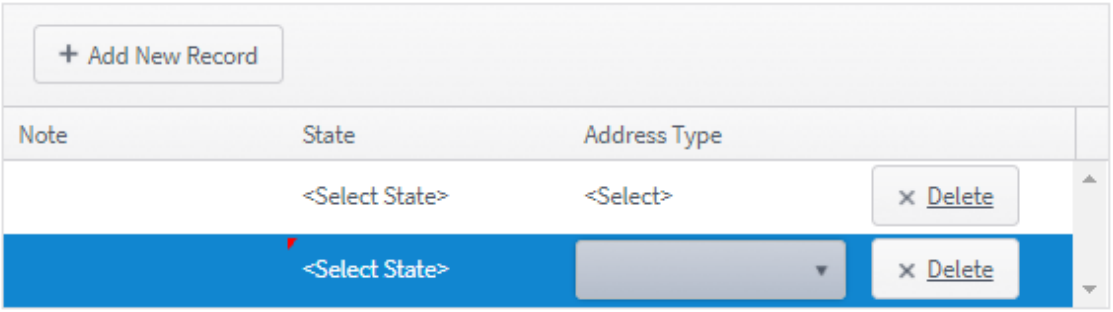
- If a form contains a grid **with Model Data** for any property (not just drop-down), Forms Renderer creates a blank row and initializes all drop-downs.

  The example below shows a grid with three columns: Note (string), State (drop-down), and Address Type (drop-down). The Selection Text property for the State drop-down is defined as <Select State>. The In Cell Editor is specified for the row. The data sources for the drop-downs are OData Queries.

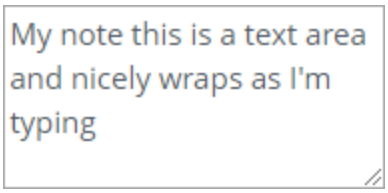
  The Model Data specified in the Grid control property settings is: `[{ "State" : "" }]`



  When the user selects *Add New Record*, a row with blank fields is added. The Selection Text labels for the drop-down lists are displayed after the user selects a new field in the row.



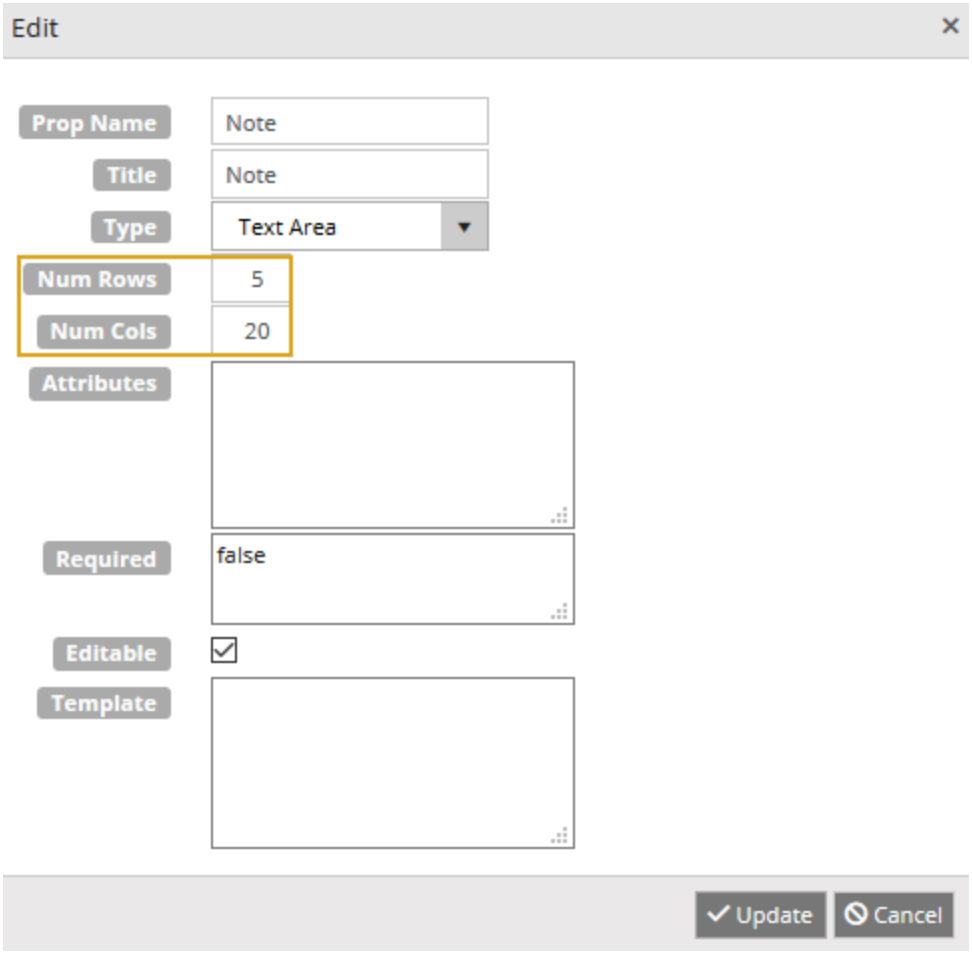
The Type option **Text Area** can be used to add a text box within a grid. The Text Area will expand to a predefined size as soon as the cell is put into edit mode.



The Text Area size is defined by the number of rows (height) and number of columns (width).

- **Num Rows** indicates the number of rows of text visible. Text that exceeds this will show a scroll bar for viewing the text. Not bindable, set a number from 2 to 999 or leave blank. If this is left blank, a default text area height will be created. Alternatively, width and height can be set with custom CSS. Use a CSS selector like "#<id of grid> .fb-grid-textarea" in a CSS file to set width and height. The cell height expands to show the textarea. Use of both techniques can produce unpredictable results.
- **Num Cols** indicates the number of columns of text (width). Text that exceeds this will wrap to the next line. Not bindable, set a number from 10 to 99 or leave blank. If this is left blank, a default text area width will be created. Alternatively, width and height can be set with custom CSS. Use a CSS selector like "#<id of grid> .fb-grid-textarea" in a CSS file to set width and height. Use of both techniques can produce unpredictable results.

  There must be enough width in the grid column to access the whole textarea in edit mode. It does not expand the grid column when it goes into edit. To enlarge the column width, both the header cell (TH) and grid cell (TD) of the column will need custom CSS. The grid cell width can be set in "Attributes" as a CSS style, but the header cell width must be set with custom CSS which matches the grid cell width.



d. **Format**

The format that is applied to the value before it is displayed. It takes the form "{0:format} where "format" is a standard number format. To format a date or number, the Data Type must be set to "date" or "number".

When a date is selected, if there is no format, a default is used. The default date format is `{0:d}`.

> The default date format applies to Forms Builder 3.5 and later. The `{0:d}` format allows for internationalization, whereas the previous default format `{0:MM/dd/yyyy}` caused problems when used with European locales.
>
> ⚠ Any forms with Format property created prior to Forms Builder 3.5 will need to be manually updated and resaved to use the new default date format.
>
> If internationalization is required, any forms created prior to Forms Builder 3.5 that use the Format property with a date value need to be manually updated and re-saved to use the new default date format .

Forms Builder 3.7 and later accepts the date format {0:dd/MM/yyyy}. It is no longer necessary to add a Locale component to force a sequence to accept dd/MM/yyyy dates.

For more information, refer to standard number and date formats in Kendo documentation.

e. **Attributes**

HTML attributes of the table cell (<td>) rendered for the column. HTML attributes which are JavaScript keyword (e.g., class) must be quoted, e.g., {"class":"myClass", style:text-align:right; font size: 14px}

The value in the *Attributes* field can be a comma separated list within a pair of braces. Use quotes around all properties and values. The *Attributes* value is added to the <td> element in the rendered table for the grid.

The attribute value ...

```
{"class":"myCustomFormat","style":"text-align: right; font-size: 14px"}
```

... results in the following <td> element:

```
<td class="myCustomFormat" style="text-align: right; font-size: 14px"><span ng-bind=”dataItem.MyPropName”>My text</span></td>
```

You can also use this class to color the text in the cell since the attribute cannot do it with a “style”. While background-color is a property of the cell, color is not because only the contents can be colored. In our case, text is embedded in a span element, so we need the class to define the color for the span contents.

```
/* Color the text red */
.myCustomFormat > span {
     color: rgb(255,0,0);    (or color:red; )
}
```

f. **Required**

Makes the column required and marks it with a red asterisk on the rendered form.

When a column is required, any row where the column does not have a value will trigger a validation message. The user will not be allowed to transition to the next page in the sequence until a value is provided in the required column for all rows in the grid.

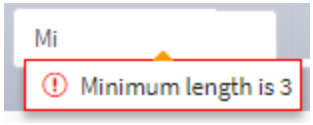
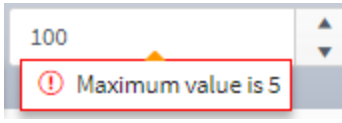
The Required property field can be set to false (default), true, or a Boolean expression.

If an expression is specified, the expression will be evaluated at run time to determine if the column is required or not. The expression is evaluated on page load of the page that contains the grid. It cannot be updated on the same page.

If the data type for the column is Boolean, the Required property will either be hidden or disabled.

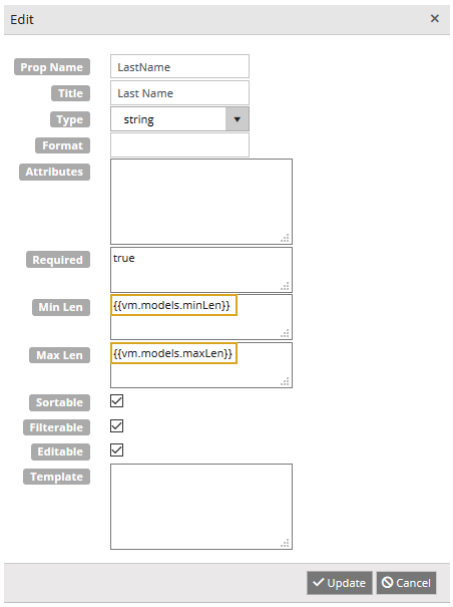
g. **Minimum and Maximum**

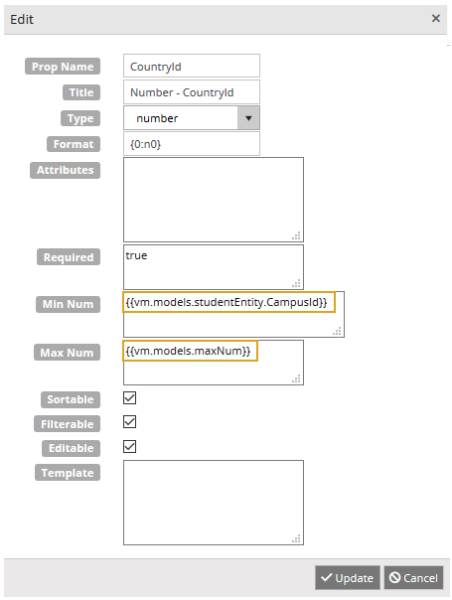
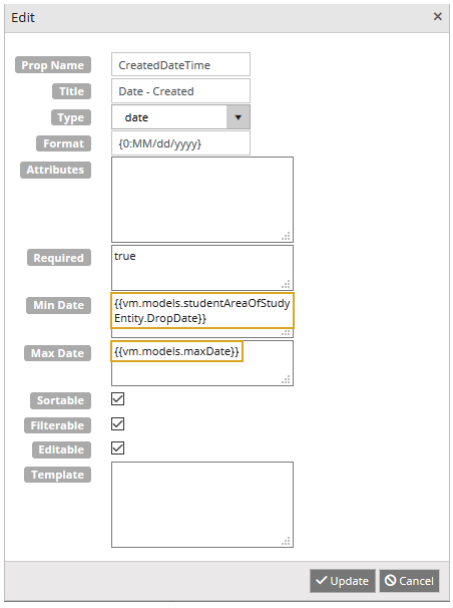
Minimum and maximum values can be assigned to the following data types:

String	Min Len and Max Len	Specify the number of characters allowed in a string. Be careful if you are using a query to populate grid because string values can have trailing blanks.	
Number	Min Num and Max Num	Specify the allowed numeric values. You can use decimal values.	
Date	Min Date and Max Date	Specify the allowed dates using the ISO 8601 format, e.g., 2018-10-28.	

If you do not want to enforce minimum and maximum values, leave the min, max, or both fields blank.

If you want to bind the minimum and maximum values, specify the model bindings as follows:

String	{{vm.models.minLen}} and {{vm.models.maxLen}}	

Number	{{vm.models.minNum}} and {{vm.models.maxNum}}	
Date	{{vm.models.minDate}}and {{vm.models.maxDate}}	

Notes:

- Model bindings within a Grid component will not be used for initial workflow argument creation. That means you need to create the corresponding arguments in the workflow unless the arguments were already created for fields in other forms.
- Model bindings for database fields can be used for min/max value bindings, e.g., studentAreaOfStudyEntity.DropDate can be used as a min value.

- Model bindings for School Defined Fields (SDFs) cannot be used because the ‘ ‘ character cannot be handled. However, you could assign an SDF to another argument in workflow and use that.
- Model bindings for min/max values are not “dynamically” bound, i.e., user cannot change values on the same form once the form is loaded. The values must be set on a preceding form in the sequence.

h. **Sortable**

Makes the column sortable. Default is true if the Sortable property is set. Set true or false.

i. **Filterable**

Adds the ability to filter a column. Set true or false.

j. **Editable**

Allows the column to be editable. When Enable Edit is set and this is turned off, the column is read only.

This has no effect if Edit is not selected or a non-editable OData query is specified.

k. **Template**

A template can be supplied for a column. This option is limited because Renderer must support the template. A Boolean data type does not use this property.

Example 6: Grid with Nullable Column

Grid OData query for entire previous education grid:

StudentPreviousEducation?$filter=Student/Id eq {{vm.models.studentEntity.Id}} and CollegeId gt 0&$select=Id,CollegeId,DegreeId,IsGraduated,GraduationDate**&$expand**=Student($select=Id),College($select=Id,Name),HighSchool($select=Id,Name),**Degree($select=Id,Name)**

Template for nullable(Degree) column:

if (data.Degree == null) { # NA #} else { # #:data.Degree.Name# #}#

For more information on how to use the Kendo embedded JavaScript to write a conditional template, refer to https://docs.telerik.com/kendo-ui/framework/templates/overview.

Example 7: Grid with Navigation Properties where the Navigation Property can be Null

If the OData query for a grid column returns navigation properties (e.g., NavigationProgram1) where the navigation property (Name) can be null and dotted notation is used (e.g., `NavigationProgram1.Name`), a JavaScript error occurs when binding the data to the column.

To prevent the JavaScript error, the property should either be named arbitrarily or just with `NavigationProgram1` instead of the dotted property name. Then, the Template can be used to access the data in the child row by checking for null on the navigation property, e.g.,

```
#= (NavigationProgram1 == null) ? ' ' : NavigationProgram1.Name #
```

The data displayed in the grid is controlled by the template and when null, `NavigationProgram1` will not cause JavaScript errors.

Property Name	Title	Format	Type	Attributes	Sortable	Filterable	Editable	Template	
ApplicationNumber	Application Number		string		Yes	Yes	Yes		Edit Delete
Program	Program		string		Yes	Yes	Yes	#= (NavigationProgram1 == null) ? ' ' : NavigationProgram1.Name #	Edit Delete
ApplicationCreatedDate	Applied On	{0:MM/dd/yyyy}	date		Yes	Yes	Yes		Edit Delete
Status	Status		string		Yes	Yes	Yes	#= (NavigationApplicationStatus == null) ? ' ' : NavigationApplicationStatus.DisplayValue #	Edit Delete

Resulting grid:

Application Number	Program	Applied On	Status
NB800021			Alternate Program
NB800022			Transfer Student
NB800023			Alternate Program
NB800044			Submitted
NB800072		5/8/2018 1:42 PM	Submitted
NB800082	AMERICAN BSC IN BUSINESS ADMINISTRATIONS	5/11/2018 12:45 PM	Application Inprogress

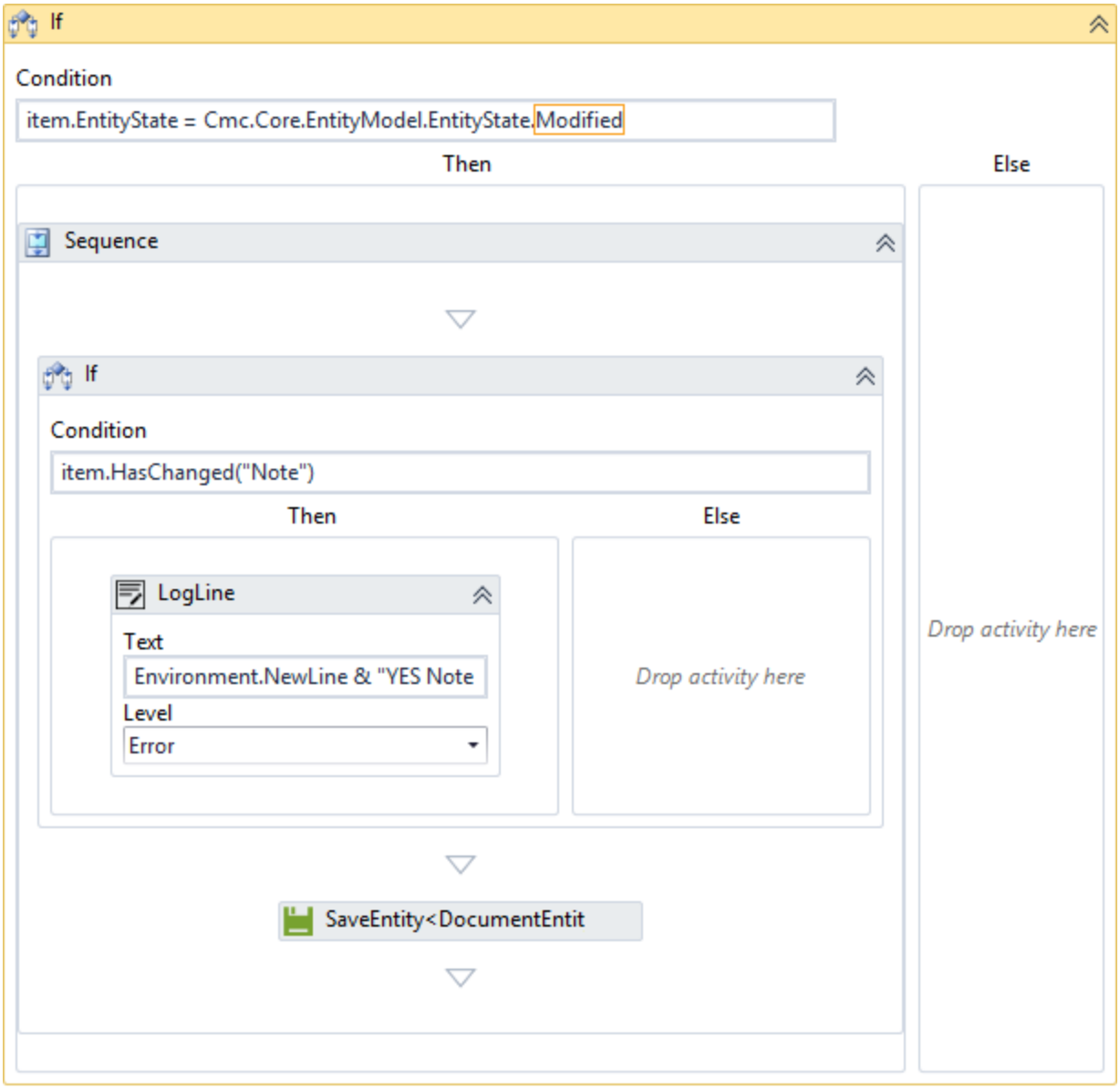
4. Select **Update**. The Edit window is closed, and the new record is added to the property editor.
5. Select **Add new record** again, **specify properties** for the next grid column, and select **Update** again. Repeat this step for each column in the grid.
6. When you have completed the properties for the columns, select appropriate properties for the rows in the lower section of the editor.

Rows

a. **Enable Edit**

Makes the row editable on the rendered form. The default setting for this property is false. You can set it to true or specify a binding (e.g., vm.models.myEdit or vm.models.studentEntity).

When users are allowed to edit rows in the grid, the final transition in the workflow for the associated sequence requires a SaveEntity activity for each entity item that is modified.



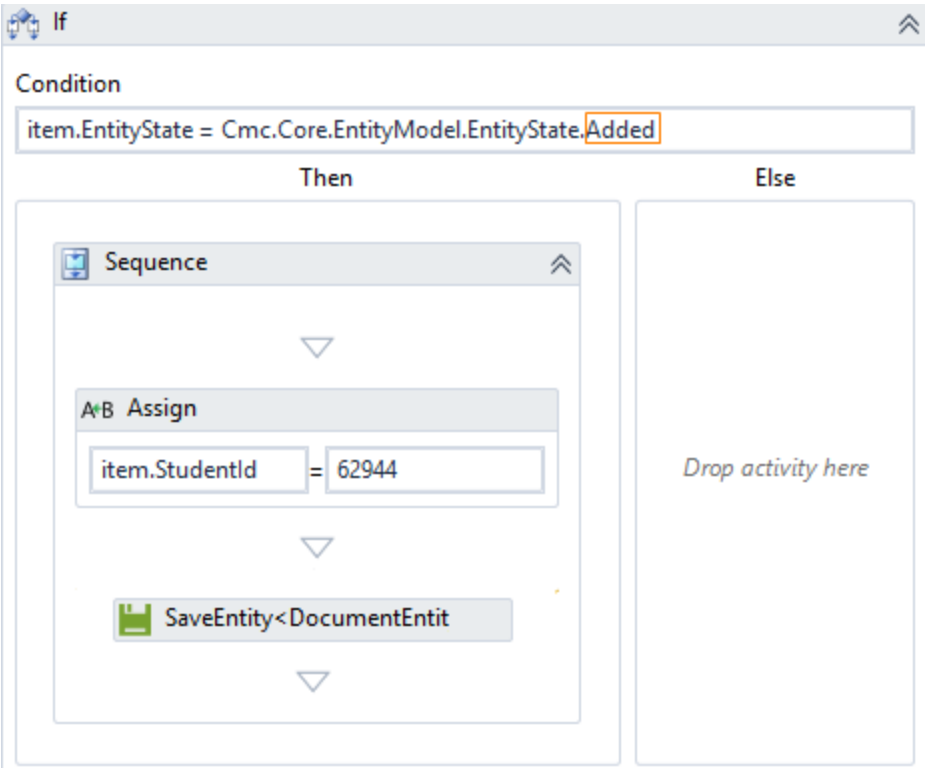
b. **Enable Add**

Allows the addition of new rows to the grid. The default setting for this property is false. You can set it to true or specify a binding (e.g., vm.models.myAdd).

Enable Add and Enable Edit should be used as coordinated pairs. If Enable Add=true and Enable Edit=false, the rendered form will display a 

button that does not have any functionality. To prevent the Confirm button from appearing on the form, set both Enable Edit and Enable Add to the same value (false or true).

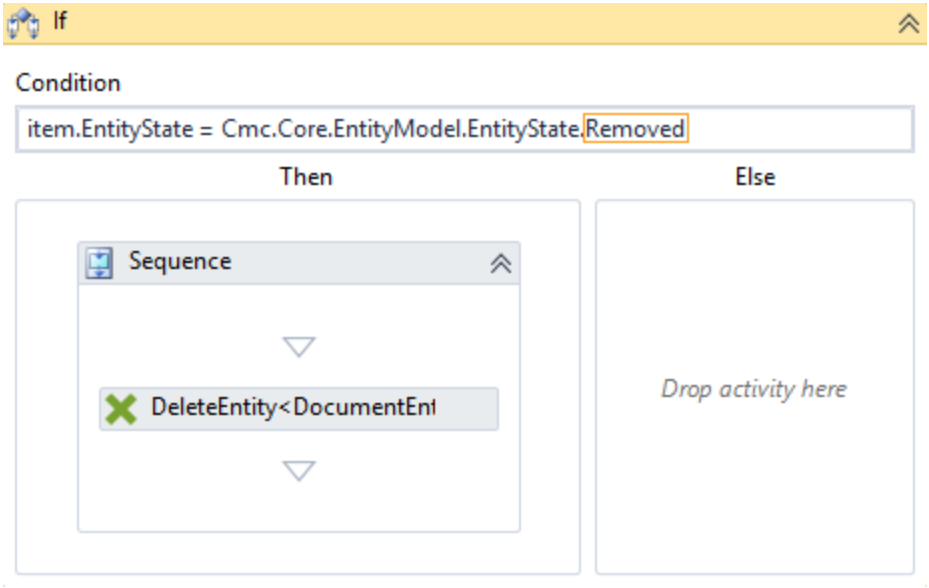
When users are allowed to add rows to the grid, the final transition in the workflow for the associated sequence requires a SaveEntity activity for each item that is added.



c. **Enable Delete**

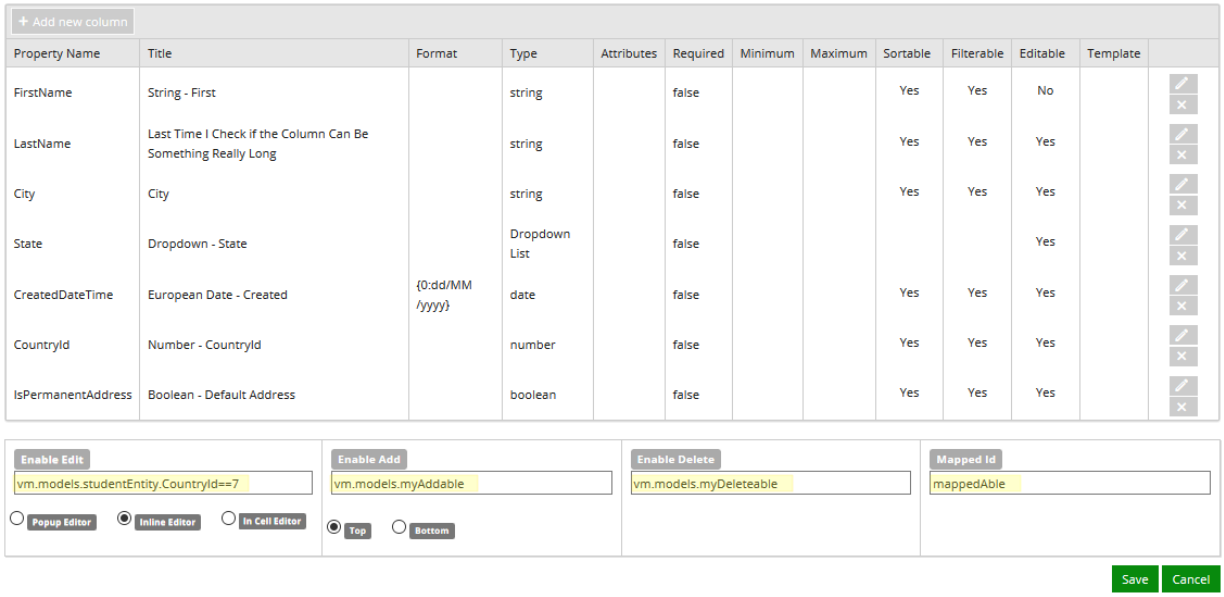
Allows deletion of rows in the grid. The default setting for this property is false. You can set it to true or specify a binding (e.g., vm.models.myDelete).

When users are allowed to delete rows from the grid, the final transition in the workflow for the associated sequence requires a DeleteEntity activity for the entity item that is deleted.



Enable Edit, Enable Add, and Enable Delete can be bound dynamically (i.e., based on another setting in the form or workflow). You may also use a Boolean value or expression.

> If **Enable Edit**, **Enable Add**, or **Enable Delete** are bound with a `vm.models.` binding, when they are changed from `true` to `false` or vice versa in a rendered form, the grid is completely refreshed with the original data. You will lose any additions, modifications, or deletions from the grid up to that point.



d. Mapped Id

This property is used to uniquely identify rows within the grid. The Ids are used internally by the grid to save records to the database. The Ids are exposed only in the JSON in debug mode.

Mapped Id defaults to “Id” if not specified. If not present in the bound data, it will be assigned values starting at 1. New rows will be given a value of -1.

An Id that is present in an entity argument in the workflow is used to find the row to update in the database on a SaveEntity.

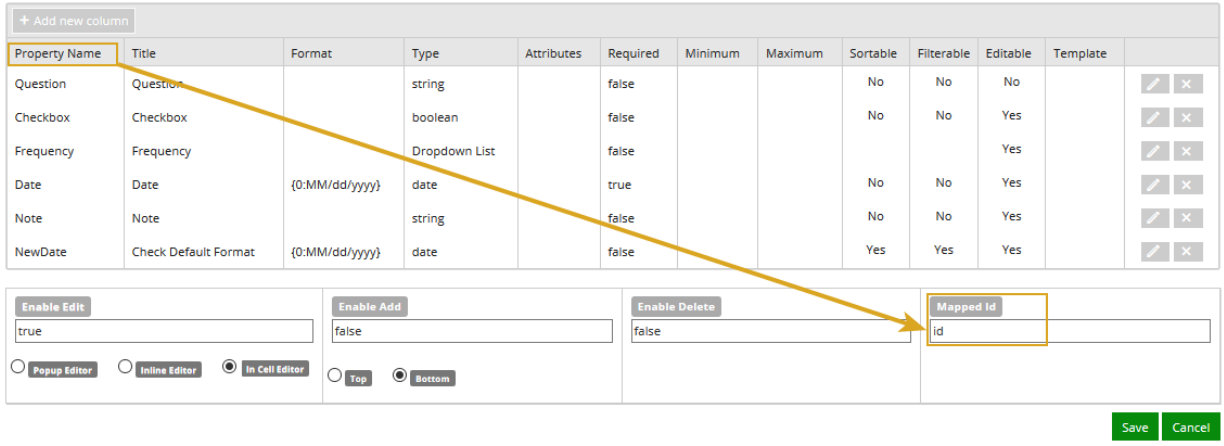
This property does not need to be mapped to a column. However, if the grid is initialized by an OData query, ensure that this field is included in the Select statement.

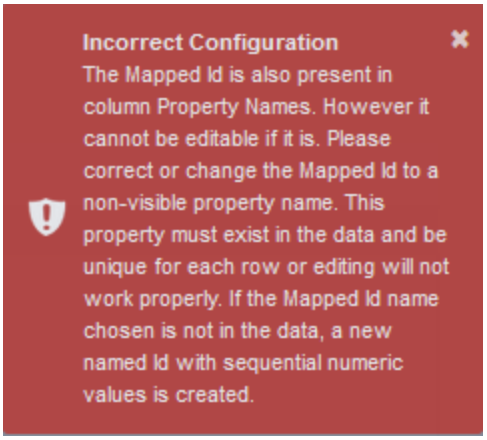
- For most Anthology Student entities, the **Id** field can be used.
- For CampusNexus CRM entities, the **KeyId** field can be used.

If using a SerializableDynamicObject not bound to a defined entity, the grid will automatically assign a unique Id to be used.

The Mapped Id is validated to ensure it is unique (case sensitive) for every row in the grid and based on the data being passed to the grid.

If the Mapped Id is the same name as a Property Name, the Property Name must be **not** be editable. If the Property Name is editable, an "Incorrect Configuration" error will be shown when you attempt to save.





e. **Popup Editor**

When edited, shows a popup editor containing the row being edited.

> ⚠ The **Popup Editor** may not display the entire cell content for a drop-down list. In those cases, the **In Cell Editor** can be more appropriate.

f. **Inline Editor**

When edited, puts the entire row into edit mode until saved or canceled.

g. **In Cell Editor**

No edit button will be displayed. Selecting a cell puts that cell into edit mode.

Use the In Cell Editor to ensure proper tabbing through each item in a row and each row in grid if data is entered using only the keyboard (i.e., not using mouse).

If you select Enable Add, adding and editing will both be enabled. Enable Edit=false will be over-ridden by Enable Add.

h. **Top**

Adds a new row at the top of the grid.

i. **Bottom**

Adds a new row at the bottom of the grid.

7. Select **Save**. The property editor is closed.

To modify column and row properties:

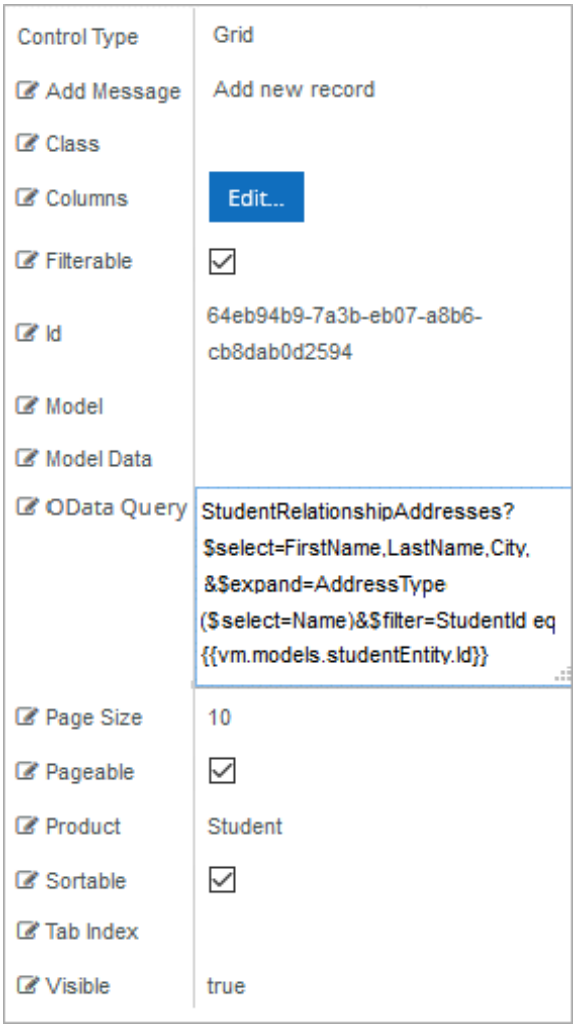
1. In the Property Settings pane of the Grid component, select the **Edit** button next to the Column property. The property editor is displayed.
2. Select **Edit** to modify the column properties.
3. Select **Delete** to remove the column properties.
4. Drag rows in the column properties grid to reorder them.
5. Edit the row properties as needed,
6. Select **Save** to close the property editor.

Grid Initialized via OData Query

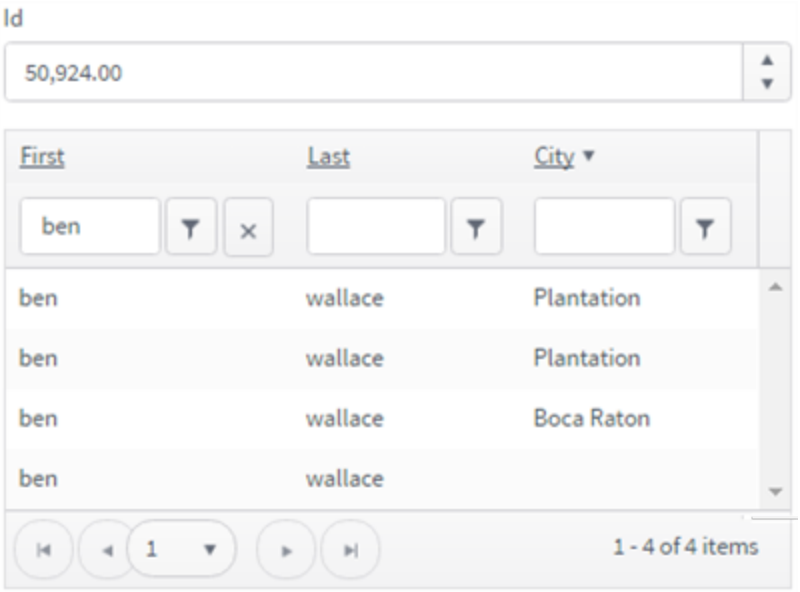
You use the Grid component for a read-only grid that displays data retrieved from the CampusNexus CRM or Anthology Student database. You will need to define an OData query to select the data to be retrieved. You then configure the grid to show specific columns and perform sorting, paging, and filtering operations via its built-in property settings.

> ⚠ The grid populated by an OData query must be a read-only grid. Do not use OData queries for editable grids.

Control Property Settings



Rendered Component

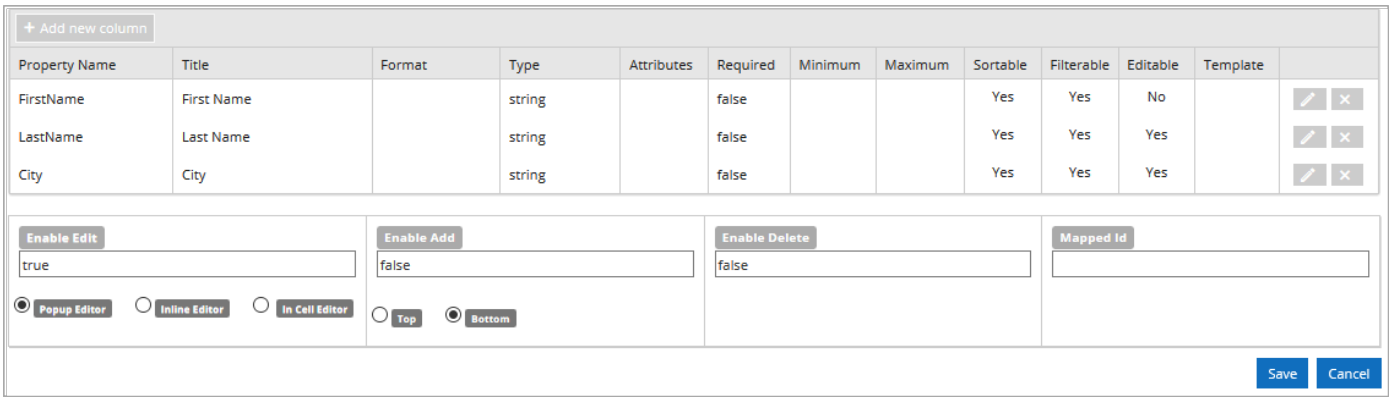


This example shows the settings for a Grid Initialized via OData Query that displays relationship addresses for a specific student. The LookupUser and GetEntity<StudentEntity> activities in the workflow supply the Student Id value. The OData query populates the rendered table.

Workflow Argument

Name	Direction	Argument type	Default value
studentRelationshipAddressEntity	In/Out	StudentRelationshipAddressEntity	*Default value not supported*

Column Specifications for a Grid with OData Query



> Unexpected results and additional columns may be appear when the grid is rendered with no Columns specified.

Grid Property: OData Query

The grid property OData Query supplies the data for a read-only grid. The Select statement for the OData Query must include all Property Names specified in Column popup editor. Note in example below how FirstName,

LastName, and City are included in the Select statement and in the Column settings. The Property Name fields must match the entity field names exactly and are case sensitive.

Example

StudentRelationshipAddresses?$select=FirstName,LastName,City,PostalCode,CreatedDateTime&$expand=AddressType($select=Name)&$filter=StudentId eq {{vm.models.studentEntity.Id}}

Important

The OData query must contain a **$select** option to retrieve the field names (FirstName, LastName, etc.). If column properties are specified, the *Property Name* field in the column editor must match the field name in the OData query.

When an OData query contains an **$expand** option (e.g., `$expand=AddressType($select=Name)` to retrieve properties within a node, the *Property Name* field in the column editor must use the dot notation, e.g., `AddressType.Name`.

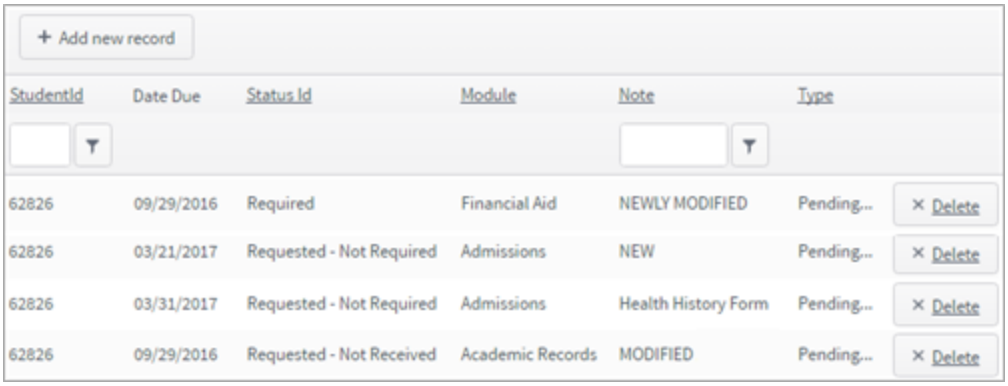
Grid Bound to an Entity

You can add the Grid component to a form that can be edited by the end-user. Any data entered or modified in the grid can be used to update an entity record in the CampusNexus CRM or Anthology Student database. For example, the grid may allow the end-user to attach documents that will then be stored in the database in the DocumentEntity array. You can configure the grid to display specific columns and perform sorting, paging, and filtering operations via its built-in property settings.

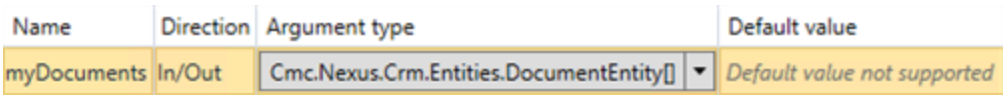
Control Property Settings

Control Type	Grid
Add Message	Add new record
Class	
Columns	Edit...
Filterable	☑
Id	6f7d6caa-3d36-6e5c-01ef-d084a3530761
Model	vm.models.myDocuments
Model Data	
OData Query	
Page Size	10
Pageable	☑
Product	Student
Sortable	☑
Tab Index	
Visible	true

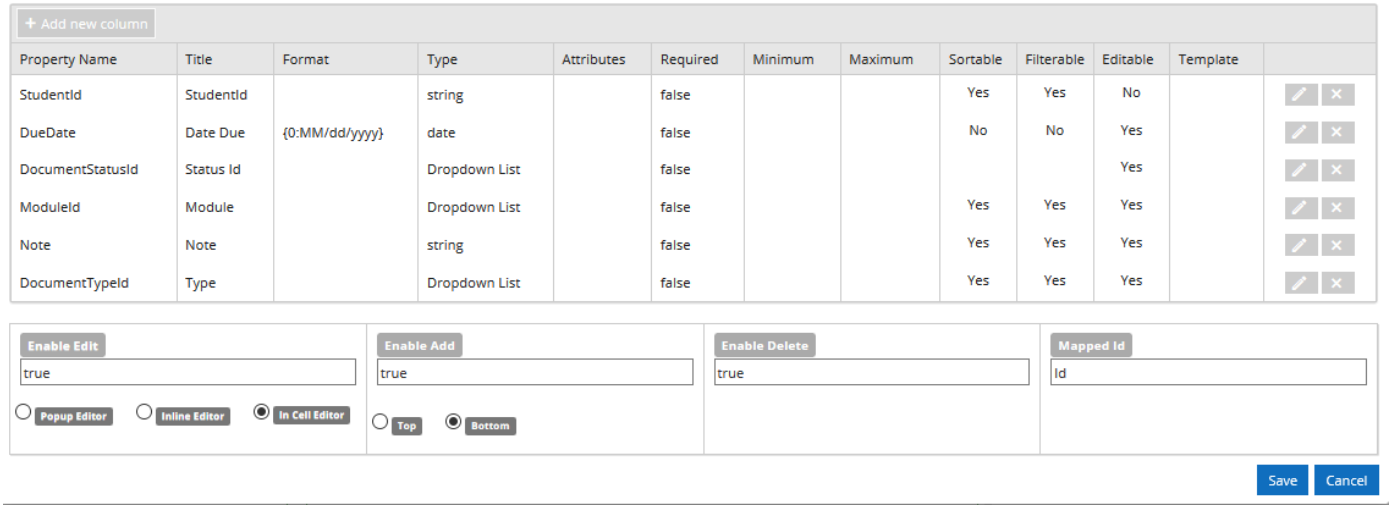
Rendered Component



Workflow Argument



Column Specifications for an Editable Grid bound to an Entity



Grid Bound to Custom Model Data (non-Entity)

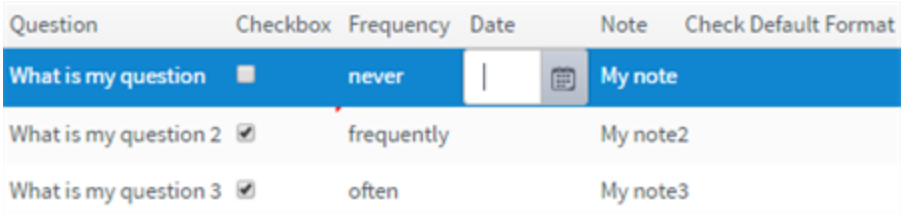
You can use the Grid component in a form that can be edited by the end-user. The grid can contain customized data rather than Anthology entity data.

The Model Data grid property is applicable for a grid that is bound to a non-entity. The Model Data property is not applicable for a grid that is bound to an entity.

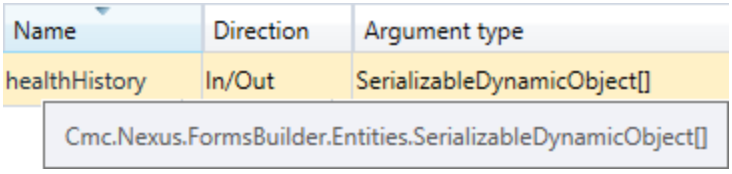
Control Property Settings

Control Type	Grid
Add Message	Add new record
Class	
Columns	Edit...
Filterable	☐
Id	66ff23f1-4b0d-3294-debb-abcfd2d0325a
Model	vm.models.healthHistory
Model Data	[{ "Question": "What is my question", "Checkbox": "false", "Frequency": "never", "Note": "My note" }, { "Question": "What is my question 2", "Checkbox": "false", "Frequency": "never", "Note": "My note2" }, { "Question": "What is my question 3", "Checkbox": "false", "Frequency": "often", "Note": "My note3" }]
OData Query	
Page Size	20
Pageable	☐
Product	Student
Sortable	☐
Tab Index	
Visible	true

Rendered Component

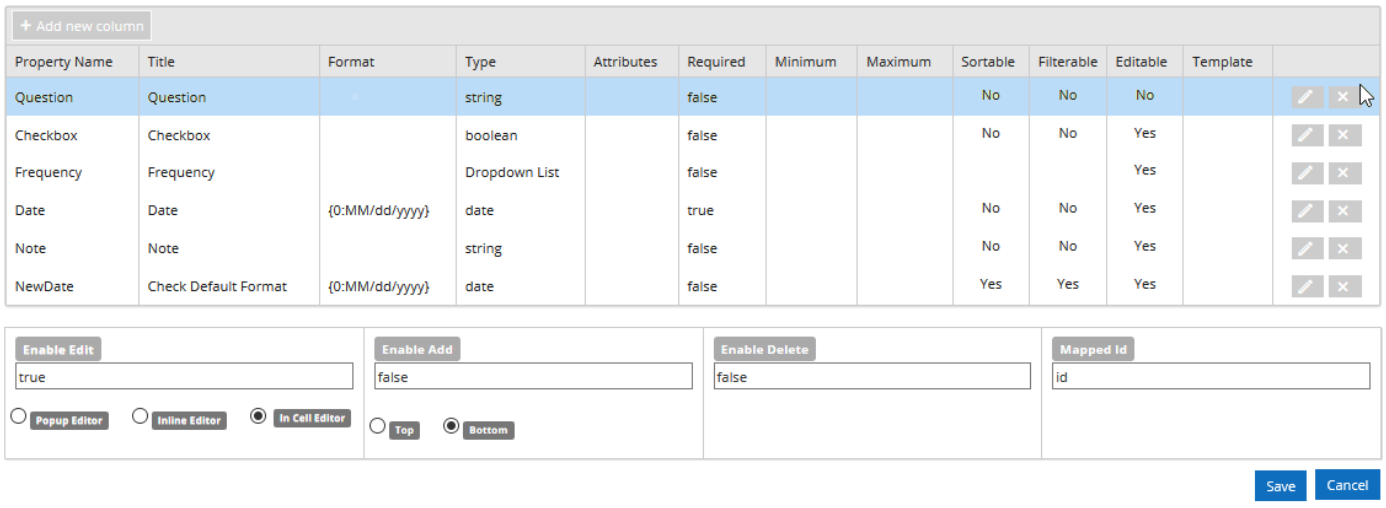


Workflow Argument



The argument name "healthHistory" is derived from the Model value in the Property Settings (vm.-models.healthHistory). The argument type "SerializableDynamicObject" is found in the FormsBuilder.Entities namespace.

Column Specifications for an Editable Grid with Model Data



Model Property

All contents of the grid should be bound via the Model property to a SerializableDynamicObject so that it will be displayed correctly when converted to PDF. This applies to a PDF file saved as attachment to a document tracking record or a PDF file displayed in e-sign processes. You can configure the grid to show specific columns and perform sorting, paging, and filtering operations via its built-in property settings.

In our example, the Model is specified and a workflow argument with a type of SerializableDynamicObject[] is used. Data in row 2 of the grid for the property *Frequency* can be accessed in the workflow with `myarg(1).DataDictionary("Frequency").ToString`.

Model Data Property

The grid property Model Data defines the initial data as a JSON string. If specified, it will initialize the grid by updating the Model value on grid load.

To display the label **<Select>** in a drop-down list, the Model Data value must be initialized as "" (empty).

The Model Data JSON string is passed to the workflow as an argument of type SerializableDynamicObject[]. This argument type will hold the data entered in the grid.

If the Model has been initialized in the workflow to a non-empty array, this data will be used. If an OData query has been specified, this data will not be used.

Example

```
[
    {
    "Question": "What is my question",
    "Checkbox": "false",
    "Frequency": "never",
    "Note": "My note"
    },
    {
    "Question": "What is my question 2",
    "Checkbox": "false",
    "Frequency": "never",
    "Note": "My note2"
    },
    {
    "Question": "What is my question 3",
    "Checkbox": "false",
    "Frequency": "never",
    "Note": "My note3"
    }
]
```

Data Types

In our example, the *Question* column uses a *String* data type. The Model Data JSON string initializes the strings in each row as *What is my question, What is my question 2,* and *What is my question 3*. The *Editable* setting in the column specification is cleared (false) because we don't want the end user to modify the questions.

The *Checkbox* column uses a *Boolean* data type which is rendered as a check box. The Model Data JSON string initializes the check box with the value *false*, i.e., the check box is cleared.

The *Frequency* column uses a *Dropdown List* data type with the values *never, rarely, frequently,* and *often*. The Model Data JSON string initializes the list with the value *never*.

Important: Any Dropdown List must be initialized, otherwise the Grid Component will fail. The initialization can be done in the Model Data value or in the workflow.

The column specification for the *Date* column includes the *Format* specification of *{0:MM/dd/yyyy}*. The Date column is not initialized in the Model Data JSON string.

The *Note* column uses a *String* data type. The Model Data JSON string initializes the strings in each row as *My note, My note2,* and *My note3*. The *Editable* setting in the column specification is selected because we want the end user to modify the notes.

Workflow Initialized List

If the list is based on **dynamic** data and the Model Data static initialization method cannot be used, then follow the step below to create a workflow initialized list.

When you are creating a Workflow Initialized List, the simplest object to use is a NameIdObject. With an array of these, the Text Member will be Name and the Value Member will be Id, and they will be of type string and integer respectively. If you don't need the Id, it is optional to set it.

In the workflow, create a variable (myList in this case). DO NOT use an argument or this will not work.

The type will be NameIdObject[] (array of NameIdObject). You can initialize the object with assign statements, but since variables allow a Default value, use the following example.

In this example we want to create a list of 2 elements, where Yes is value 1 and No is value 2. Set Default to:

```
new NameIdObject(1){new NameIdObject With { .Name="Yes", .Id=1}, new NameIdObject With { .Name-
e="No", .Id=2}}
```

Note some significant syntax here:

- The 1 for the array size is VB syntax for an array of 2 elements, with index 0 and 1.
- There is a dot before each property name in the With sections.

If you were doing this in assign statements, you could break the statements down as follows:

myList = new NameIdObject(1){} - creates a 2-element array that is empty.

myList(0) = new NameIdObject - initialize the first array element with a new object, "With" could have been used here instead of the following two assigns.

myList(0).Name = "Yes"

myList(0).Id = 1

etc.

The Default initialization above, while looking more complex, is less wieldy than a few assign statements in the workflow.

To use this, you must expose this as an Out argument of type NameIdObject[]. After you create this argument, you do this with a final assign statement.

```
myArgList = myList
```

The result is that all drop-down list controls that have vm.models.myArgList as the Model For Value List binding (in the popup), will have a Yes/No list. Their Text Member must be Name, and if you use the Value Member, it must be Id.

Note: If any of the following is true, then a SerializableDynamicObject can be used in the same way. It has none of the following limitations.

a. You need more than two properties
b. The property names cannot be Name and Id
c. The types of the property names cannot be string and int respectively.

However, the initialization for the SerializableDynamicObject is considerably more complex to understand to do the same thing as above (with only 2 elements). Here it is:

```
new SerializableDynamicObject(1){new SerializableDynamicObject With { .DataDictionary = new Dictionary (Of String, Object) From { { "Name", "Yes"}, { "Id", 1} } }, new SerializableDynamicObject With { .DataDictionary = new Dictionary (Of String, Object) From { { "Name", "No"}, { "Id", 2} } } }
```

You must do this with a variable, and then you must assign it to an argument which is bound to the control.

Grid Bound to Results of ExecuteODataQuery

You can use the ExecuteODataQuery activity to bind the contents of a grid to the results of the OData query.

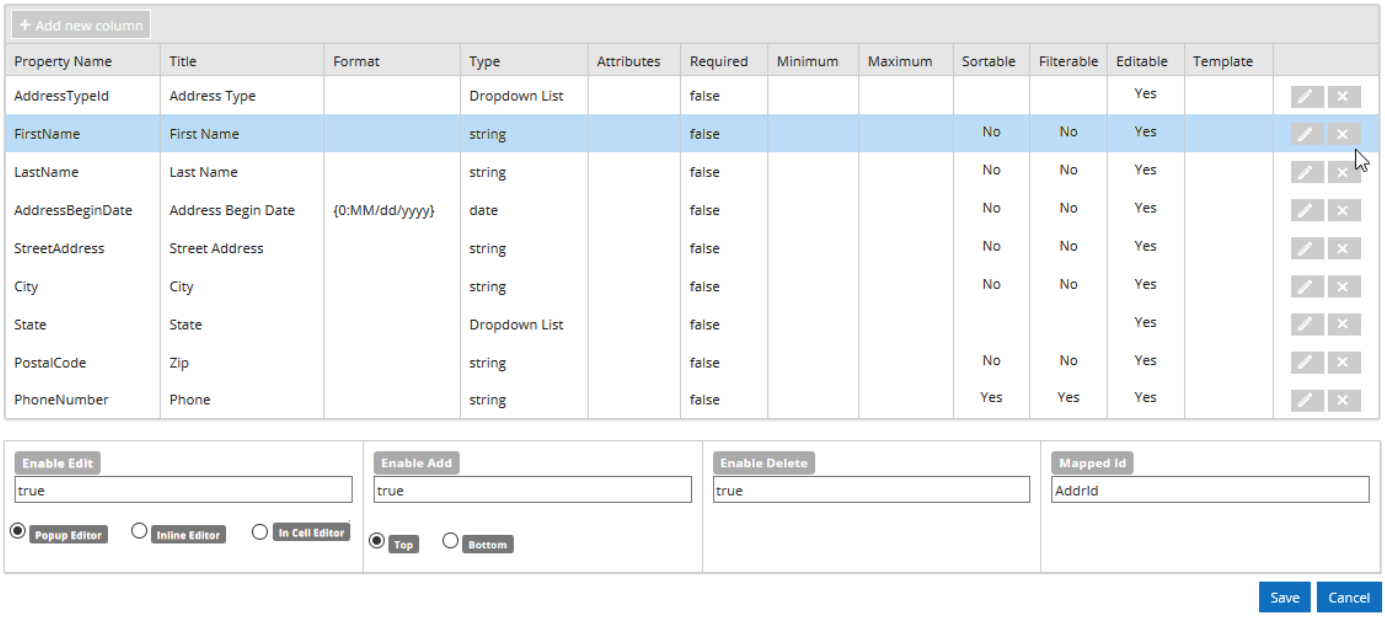
In this example, the form sequence contains a form that enables a student to enter the addresses of related persons. The form prompts for the student's first and last name and presents a grid for the entry/display of the relationship addresses.



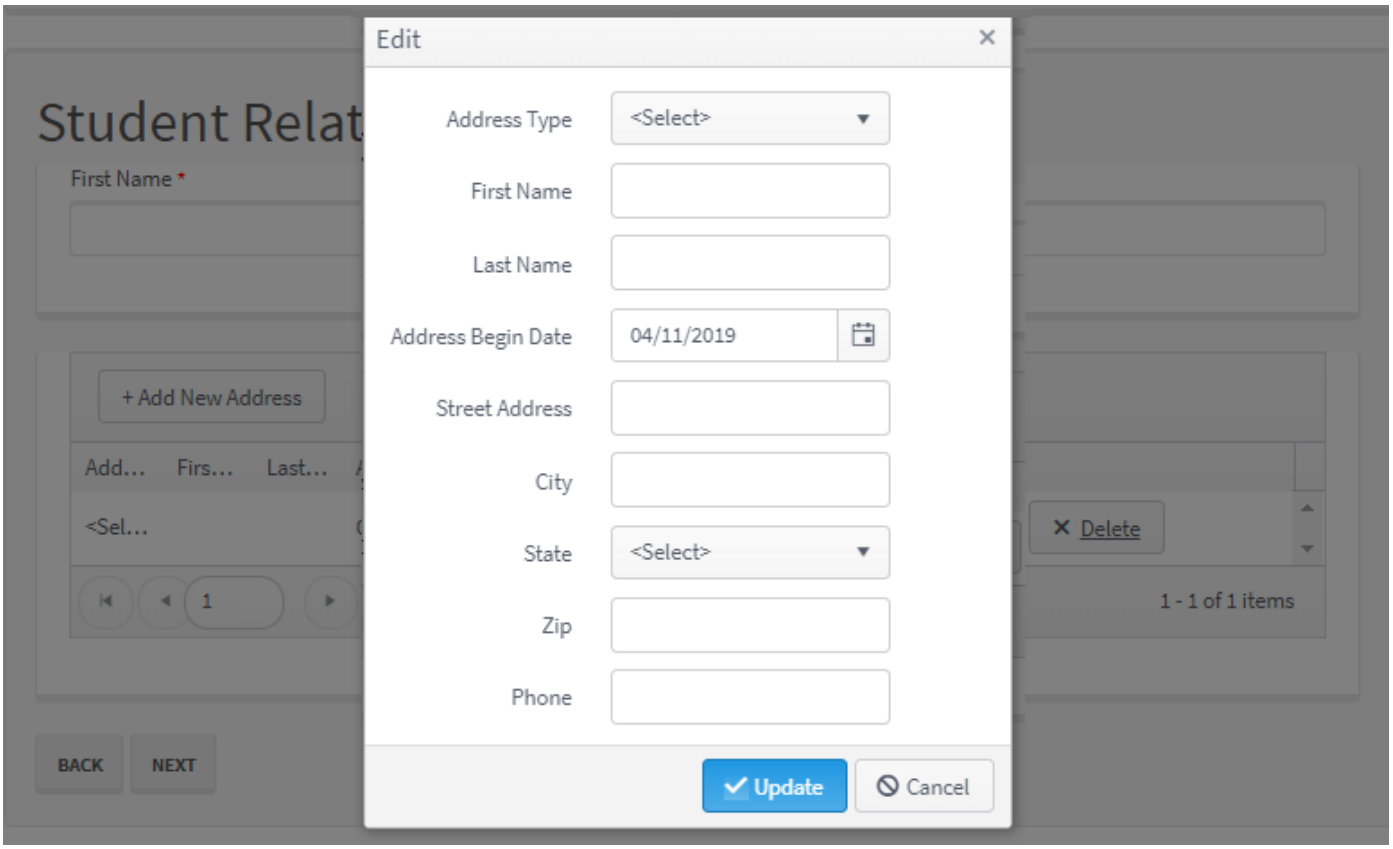
Grid Property Settings

Control Type	Grid
Add Message	Add New Address
Class	
Columns	Edit...
Filterable	☐
Id	id52d0eea4-1799-359b-14d5-640ec70c2d3c
Model	vm.models.myAddresses
Model Data	
OData Query	
Page Size	20
Pageable	☑
Product	Student
Sortable	☐
Tab Index	
Visible	true

Grid Columns Property Settings



Rendered Component



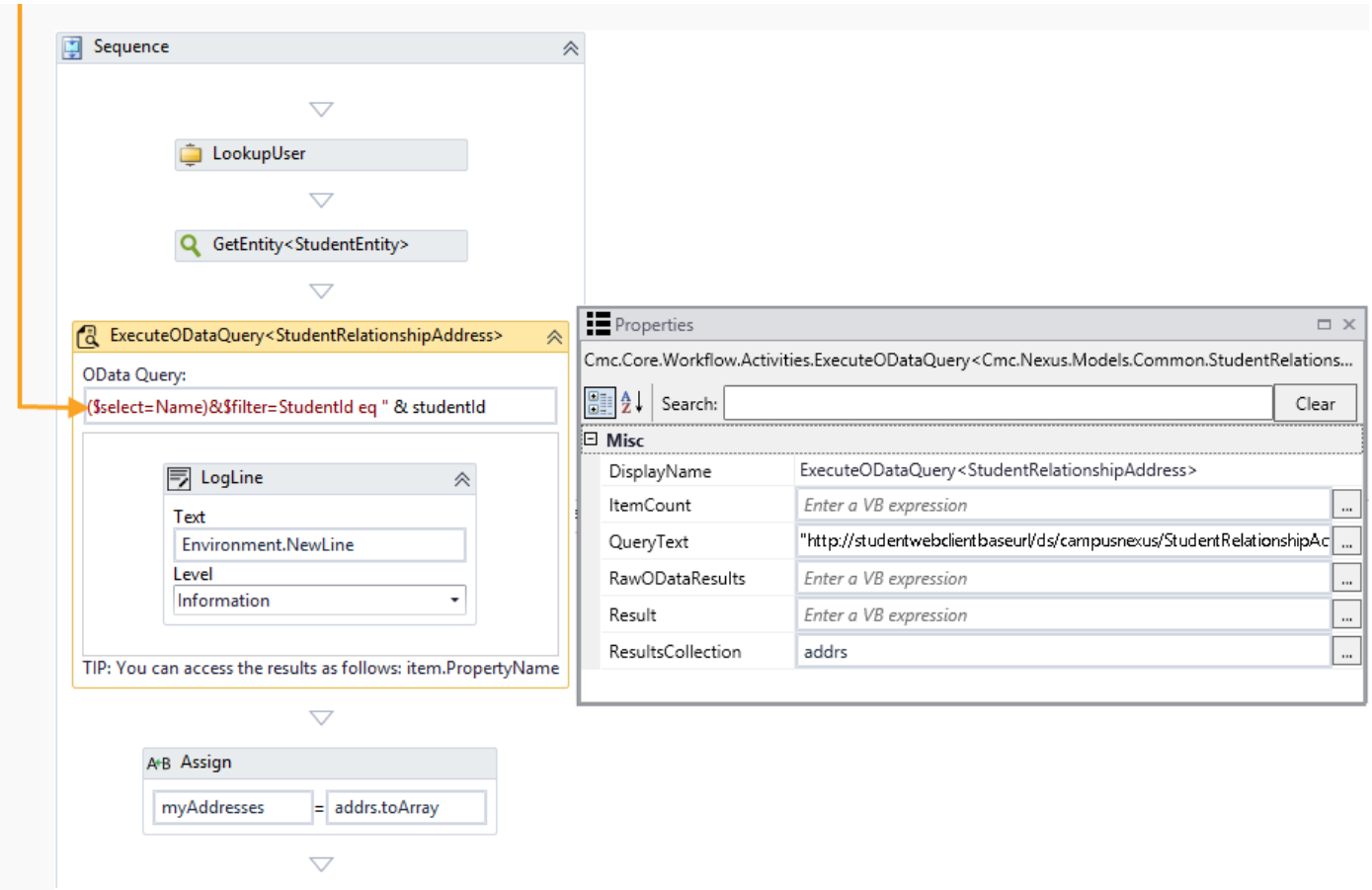
Workflow Arguments

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event	In/Out	ConstructedEvent	*Default value not supported*
myAddresses	In/Out	System.Collections.Generic.IEnumerable<Cmc.Nexus.Models.Common.StudentRelationshipAddress>	*Default value not supported*
studentEntity	In/Out	StudentEntity	*Default value not supported*

The workflow for the sequence looks up the student identifier and retrieves the student entity.

The ExecuteOData activity captures the relationship address information and creates a results collection. The OData Query is:

```
"http://stu-
dent-
web-
cli-
ent-
baseurl/d-
s/cam-
pus-
nex-
us/Stu-
dentRela-
tion-
shipAd-
dresses?$se-
lect-
=AddressTypeId,FirstName,LastName,City,StreetA-
ddress,State,PostalCode,Note,AddressBeginDate&$expand=AddressType
($select=Name)&$filter=StudentId eq >" & studentId
```



The Assign activity writes the Results Collection "addrs.toArray" to the "myAddresses" argument that is defined in the Model property of the grid.

Notes

- The "toArray" attribute in the Assign activity will **not** be displayed in Intellisense.
- A manual import of the following namespaces is required:
    - System.Collections.Generic
    - System.Linq

After the Results Collection has been obtained, further workflow activities can be used to manipulate the grid, e.g., save, modify, or delete records.

The Grid component does not support a "Required" validation on the client-side. However, this can still be achieved via workflow logic to verify a value has been set and if not, display a custom validation error.

Grid Populated by ExecuteQuery

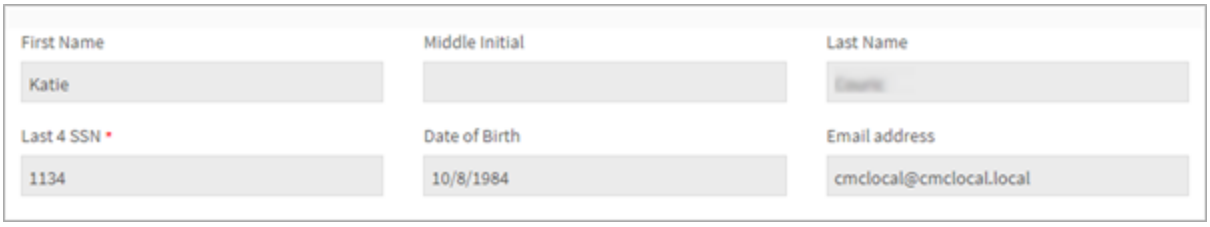
In this example, the Grid component is a read-only grid that displays transactions from the student's ledger card based on a selected term. To collect the data needed to populate the grid, the ExecuteQuery activity is used. The ExecuteQuery activity gets the requested dataset and loops through each row to populate the grid.

Form Sequence

The form sequence has 3 forms and an End State Form.

Form 1

This form has fields for the student's First Name, Middle Initial, Last Name, SSN, Date of Birth, and Email Address. Each field is bound using the Model property.



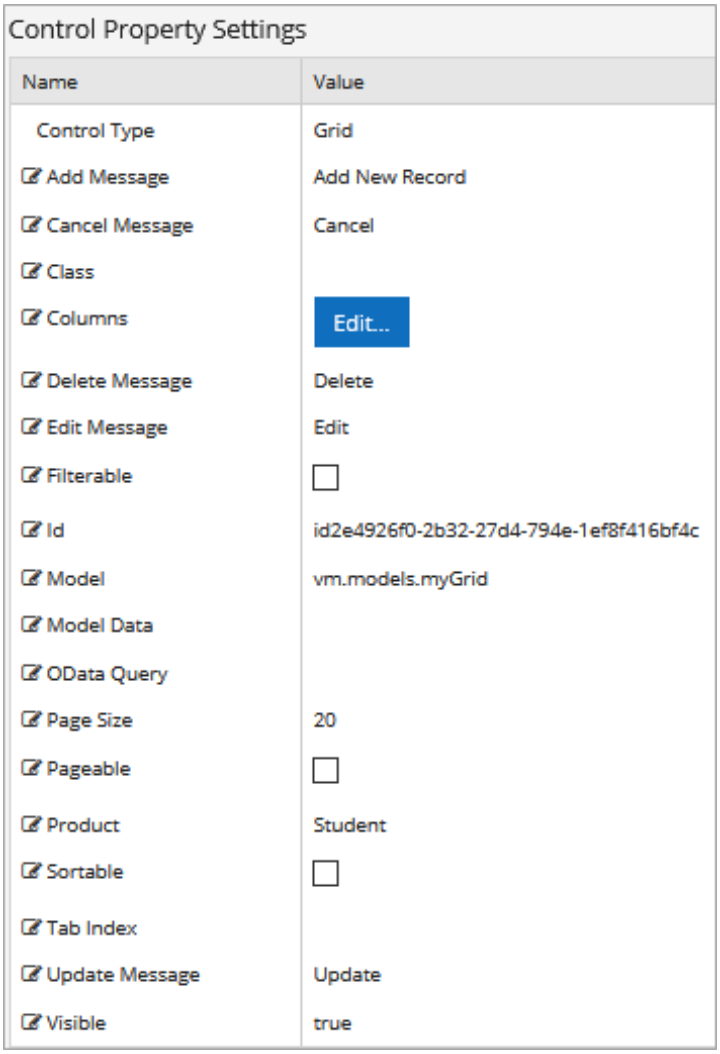
Form 2

This form has a Drop-down List component to select a Term.

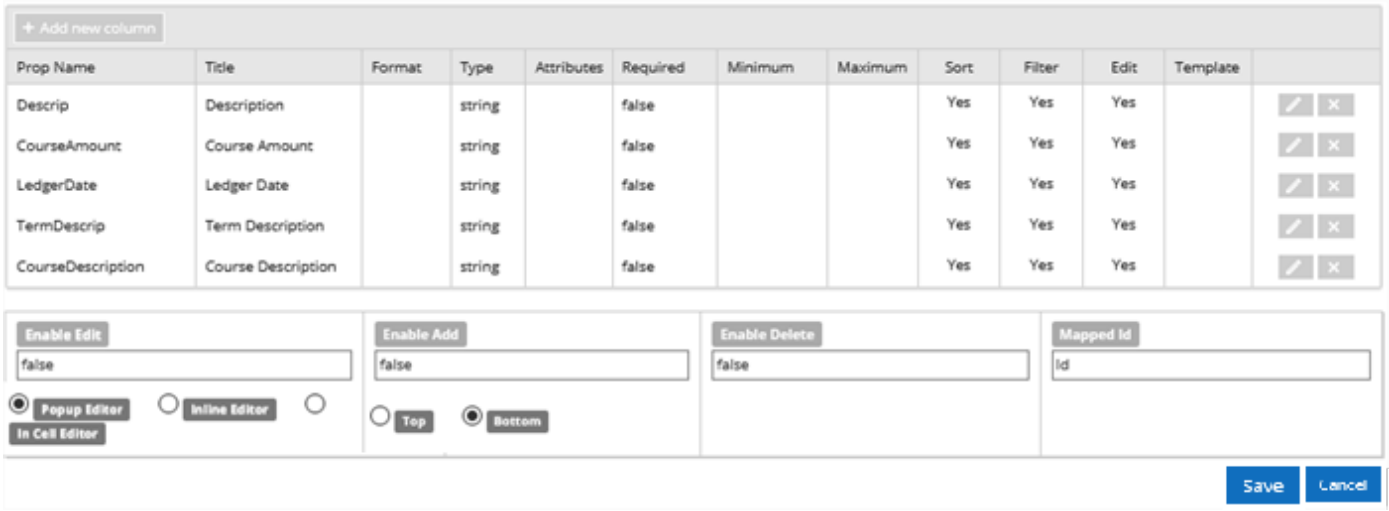


Form 3

This form has a Grid component with the following properties:

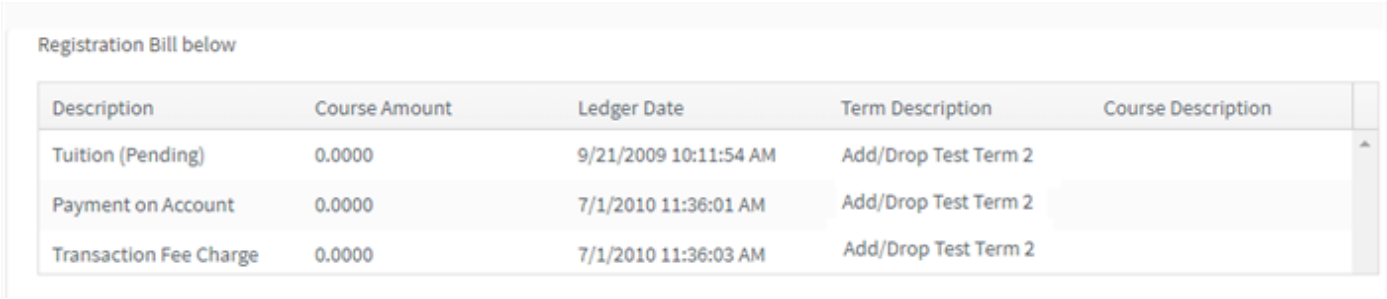


Column Specifications:



The Grid Property Name matches the fields being returned via the ExecuteQuery activity.

The rendered form has a grid with 5 columns populated with data retrieved from the database:



Workflow

Form 1 (State 1)

The workflow begins with a **LookupUser** activity that returns the `SyStudentID`, which is a variable of type Int32.

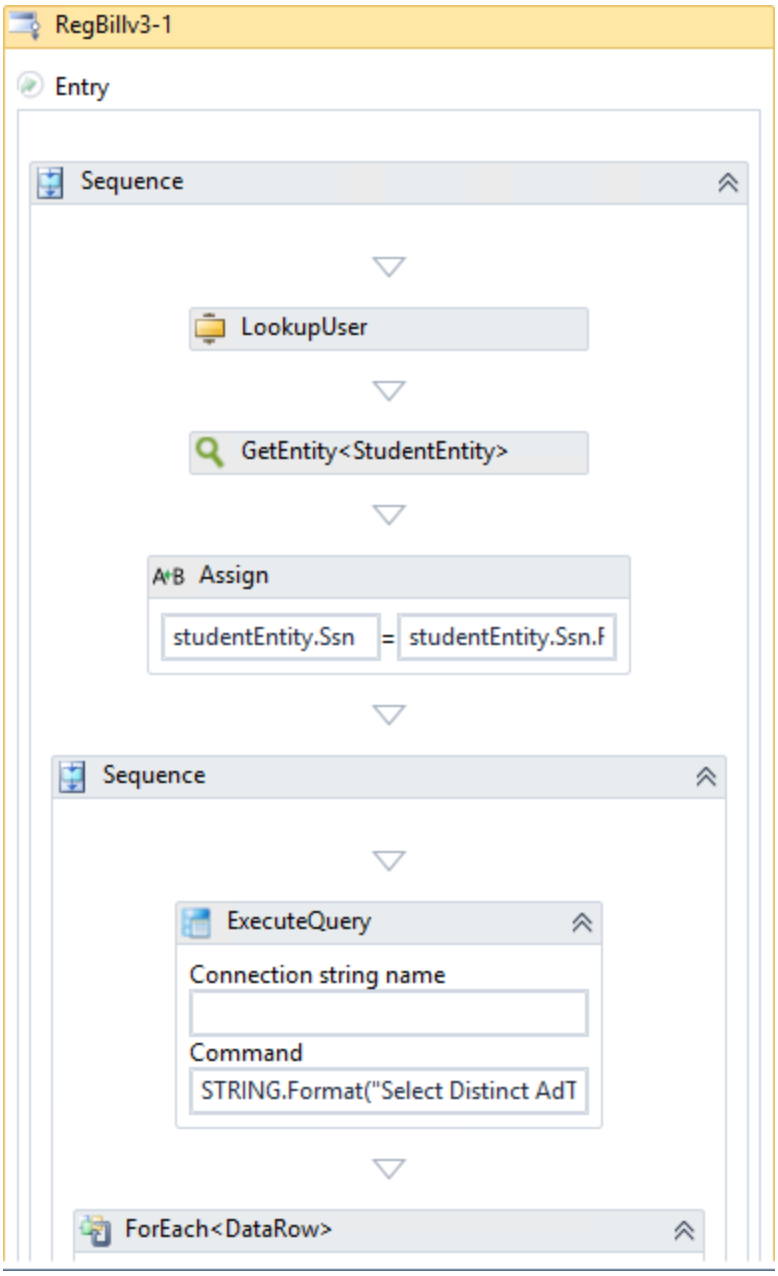
The **GetEntity<StudentEntity>** activity uses the `SyStudentID` and returns the `StudentEntity`, which is an argument of type StudentEntity.

The **Assign** activity assigns the value `studentEntity.Ssn.Remove(1,7)` to `studentEntity.Ssn`. This formats the SSN to display only the last 4 digits. It starts at 1 and removes 7 digits. This includes the dashes (111-11-1111). So that leaves the last 4 of the SSN.

The **ExecuteQuery** activity uses the following CommandText as InArgument and returns the OutArgument `RegistrationBill`, which is a variable of type DataSet. The CommandText selects specific fields from various data tables for the given SyStudentID.

```
STRING.Format("Select Distinct AdTerm.Descrip AS termSelect, SaTrans.AdTermId AS termSelectID from SaTrans join AdTerm on SaTrans.AdTermId = AdTerm.AdTermId WHERE SyStudentID = {0}",studententity.Id)
```

With the ExecuteQuery and subsequent workflow activities, we are basically retrieving all the terms that are associated to ledger card transactions and populating the drop-down list on Form 2 with those terms so the student can select them. After the student selects a term, Form 3 will populate all the ledger card transactions for that selected term.



The **ForEach<DataRow>** activity with `TypeArgument=System.Data.DataRow`follows the ExecuteQuery activity.

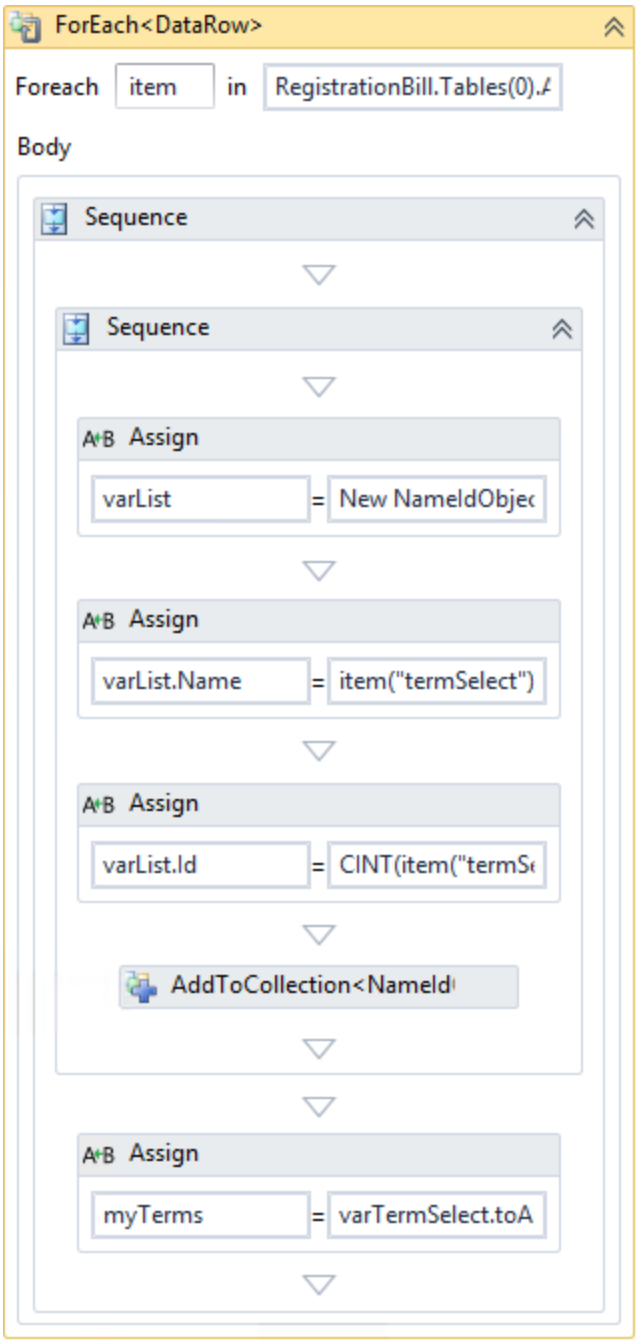
For each `item` in `RegistrationBill.Tables(0).AsEnumerable` the following 3 assignments are made:

To	Value	Notes
`varList`	`New NameIdObject`	varList is a variable of type NamedIdObject.
`varList.Name`	`item("termSelect").ToString`	termSelect is an argument of type String.
`varList.Id`	`CINT(item("termSelectId"))`	

Following the assignments, the **AddToCollection<NameIdObject>** activity uses the InArguments `Collection=varTermSelect` and `Item=varList` and `TypeArgument=Cmc.Nexus.FormsBuilder.Entities.NamedIdObject`, where `varList` and `varTermSelect` are variables of type List<NamedIdObject>.

Next, the following assignment is made:

To	Value	Notes
`myTerms`	`varTermSelect.toArray`	myTerms is an argument of type NamedIdObject[]. varTermSelect is a variable of type List<NamedIdObject>.



Form 2 (State 2)

A **LogLine** activity captures the following text: `Environment.NewLine + "Selecting Term" + Environment.NewLine`

An **Assign** activity assigns `MyTermsSelect` to `Test`, where MyTermsSelect and Test are variables of type Int32.

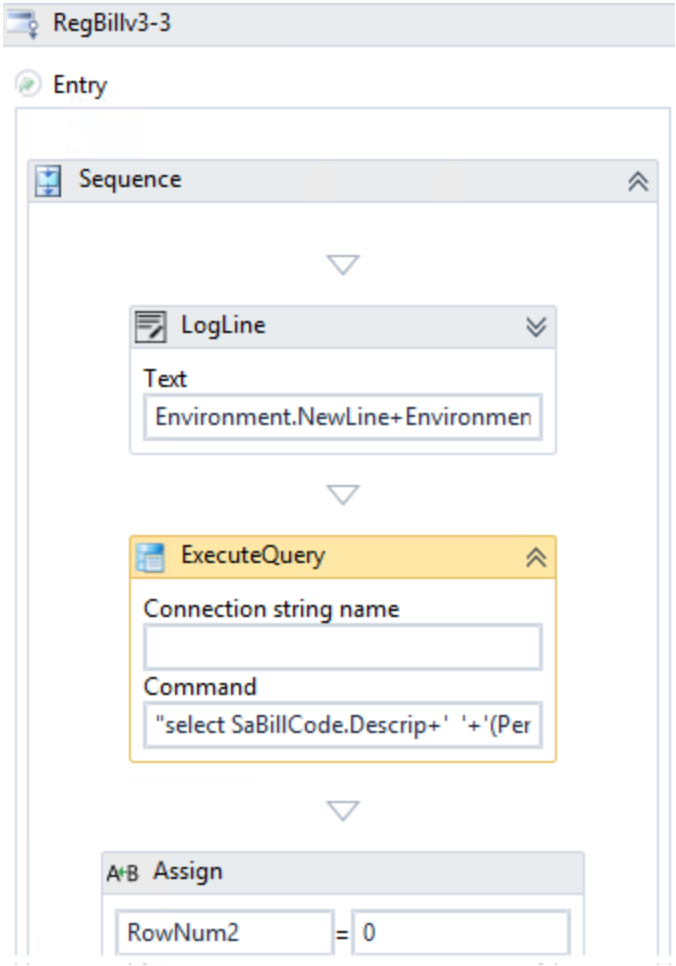
The Next transition to the 3rd form has another **LogLine** activity with `Text=Environment.NewLine + "Entering 3rd Form" + Environment.NewLine`.

Form 3 (State 3)

A **LogLine** activity captures the following text: `Environment.NewLine+Environment.NewLine+ "*** The Student " + SyStudentId.ToString + "***" + Environment.NewLine+Environment.NewLine`

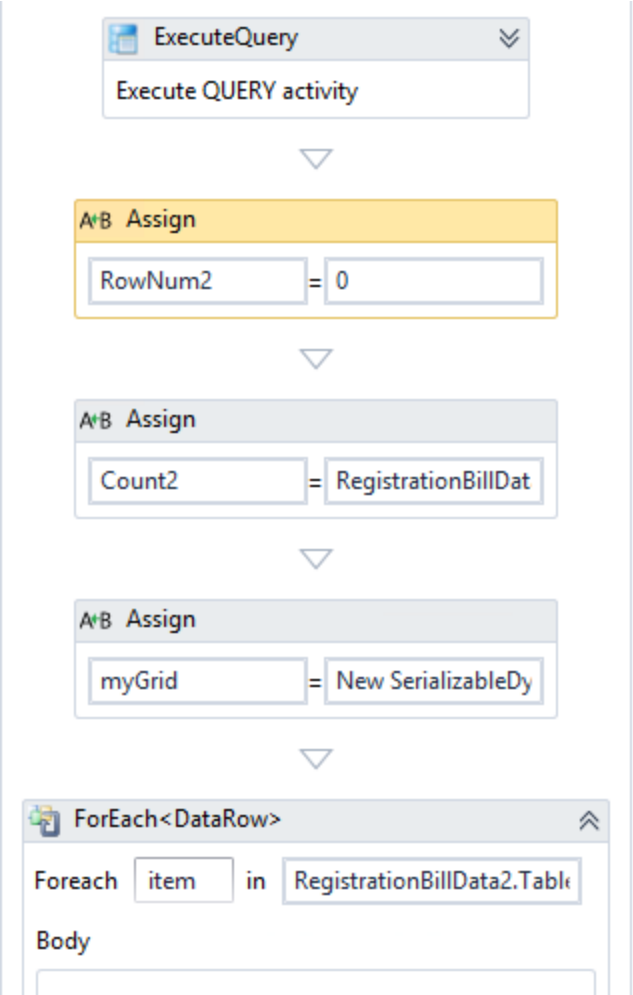
Next, an **ExecuteQuery** activity uses the following CommandText as InArgument and returns the OutArgument `RegistrationBillData2`, which is a variable of type DataSet.

```
"select SaBillCode.Descrip+' '+'(Pending)' as Descrip, SaPendingCharge.Amount, SaPend-
ingCharge.Type, SaPendingCharge.DateAdded as Date, (select AdTerm.Descrip from AdTerm
(nolock) where AdTerm.AdTermID= SaPendingCharge.AdTermID ) as TermDescrip, 0 AS
CourseAmount, '' AS CourseDescription from dbo.SaPendingCharge SaPendingCharge Inner
Join dbo.SaBillCode SaBillCode (nolock) on SaPend-
ingCharge.SaBillCodeID=SaBillCode.SaBillCodeId inner join AdTerm (nolock) on AdTer-
m.AdTermID = case when SaPendingCharge.RegAdTermID IS not null then
SaPendingCharge.RegAdTermID else SaPendingCharge.AdTermID end where SaPend-
ingCharge.SyStudentID =" + SyStudentId.ToString + "and SaPendingCharge.AdTermId =" +
Test.ToString + "Union All Select SaTrans.Descrip, SaTrans.Amount, SaTrans.Type,
SaTrans.[Date] as Date, AdTerm.Descrip as TermDescrip, IsNull (SaTransCourseA-
mount.Amount, 0) AS CourseAmount, AdCourse.Descrip AS CourseDescription From
dbo.SaTrans (nolock) inner join AdTerm (nolock) on AdTerm.AdTermID = SaTrans.AdTermID
left join saTransCourseAmount on saTrans.saTransID = saTransCourseAmount.saTransID
left join AdCourse on AdCourse.AdCourseID = SaTransCourseAmount.AdCourseID where
SaTrans.SyStudentID =" + SyStudentId.ToString + "and SaTrans.AdTermId =" +
Test.ToString
```



The following 3 assignments are made after the ExecuteQuery activity:

To	Value	Notes
`RowNum2`	`0`	RowNum2 is a variable of type Int32.
`Count2`	`RegistrationBillData2.Tables (0).Rows.Count`	Count2 is a variable of type Int32.
`myGrid`	`New SerializableDynamicObject (Count2-1) {}`	myGrid is an argument of type SerializableDynamicObject[].



The **ForEach<DataRow>** activity with `TypeArgument=System.Data.DataRow`follows the assign activities.

For each `item` in `RegistrationBillData2.Tables(0).AsEnumerable` the following 8 assignments are made:

To	Value	Notes
`myGridRow`	`New RepeaterSerializableDynamicObject`	myGridRow is a variable of type SerializableDynamicObject.
`myGridRow.DataDictionary("Descrip")`	`Convert.ToString(item("Descrip"))`	Descrip is an argument of type String.
`myGridRow.DataDictionary("CourseAmount")`	`Convert.ToString(item("CourseAmount"))`	CourseAmount is an argument of type String.
`myGridRow.DataDictionary("LedgerDate")`	`Convert.ToString(item("Date"))`	LedgerDate is an argument of type String.
`myGridRow.DataDictionary("TermDescrip")`	`Convert.ToString(item("TermDescrip"))`	TermDescrip is an argument of type String.

To	Value	Notes
`myGridRow.DataDictionary ("CourseDescription")`	`Convert.ToString(item("CourseDescription"))`	CourseDescription is an argument of type String.
`myGrid(RowNum2)`	`myGridRow`	myGrid is an argument of type SerializableDynamicObject[].
`RowNum2`	`RowNum2 + 1`	RowNum2 is a variable of type Int32.

CRM Grid for One-to-Many Relationships

You can use the Grid component to create a form to retrieve/save records that are in a one-to-many relationship. In a one-to-many relationship, the parent can have a single child record, multiple child records, or zero child records. The child cannot have more than one parent record.

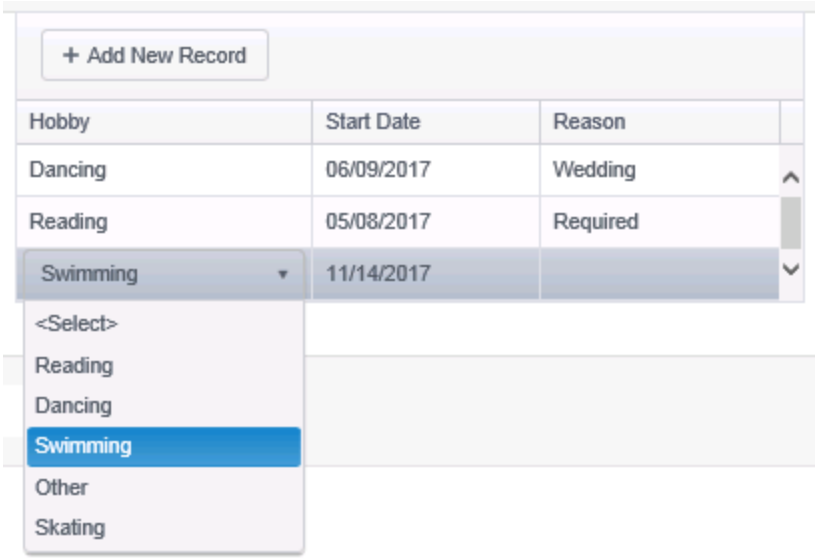
In the example below, the grid is used to populate child records for a CRM Contact entity. The child records contain hobbies that are entered on the rendered grid.

To initialize this type of grid, workflow activities need to be used rather than OData queries. The workflow activities use the "vm.models.HobbyGrid" Model property assigned in the Grid Property Settings. The rendered grid enables the user to add child records (hobbies) for a given parent record (a contact named "Kaly").

Control Property Settings

Control Type	Grid
Add Message	Add New Record
Class	
Columns	Edit...
Filterable	☐
Id	ide81525cc-965a-ba83-9654-20d34c9fa46e
Model	vm.models.HobbyGrid
Model Data	
OData Query	
Page Size	20
Pageable	☐
Product	CRM
Sortable	☐
Tab Index	
Visible	true

Rendered Component



Workflow Arguments

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event	In/Out	ConstructedEvent	*Default value not supported*
contact	In/Out	Contact	*Default value not supported*
HobbyGrid	In/Out	ContactKalyRecordList[]	*Default value not supported*
Create Argument			

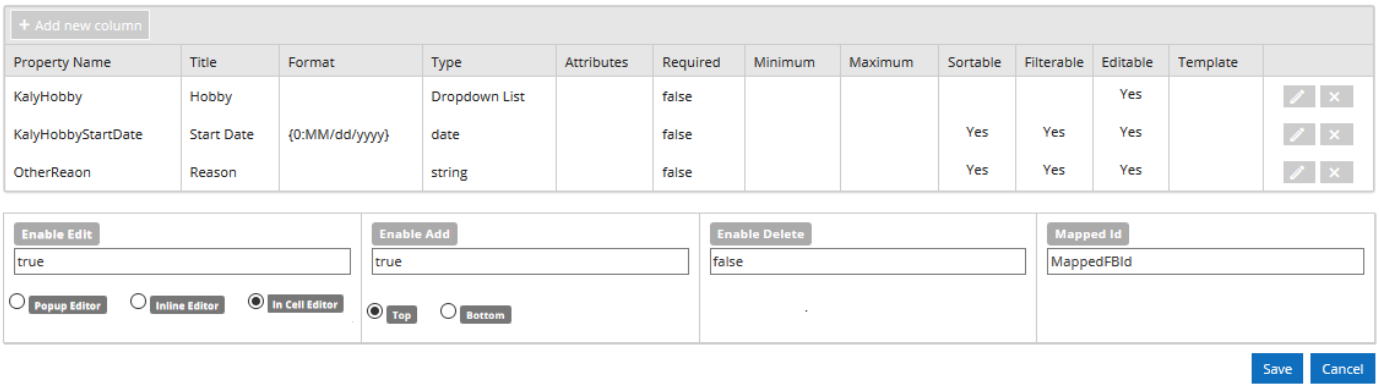
Cmc.NexusCrm.Common.Entities.ContactKalyRecordList[]

Workflow Variables

Name	Variable type	Scope	Default
renderedFormImage	String	StateMachine	*Enter a VB expression*
cnt	Int32	StateMachine	*Enter a VB expression*
contactId	Int32	StateMachine	*Enter a VB expression*
LocalHobbyGrid	ContactKalyRecordList[]	StateMachine	*Enter a VB expression*
Create Variable			

Cmc.NexusCrm.Common.Entities.ContactKalyRecordList[]

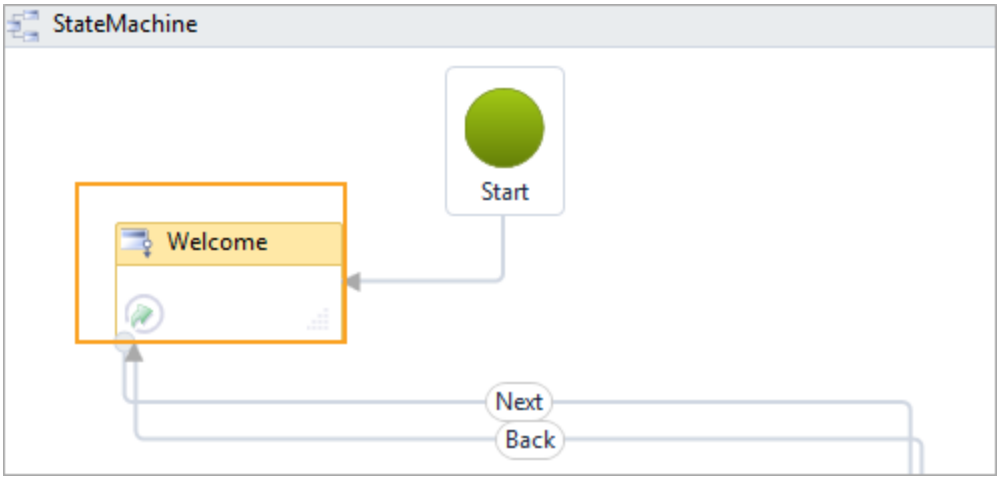
Column Specifications for an Editable Grid Initialized via Workflow Activities



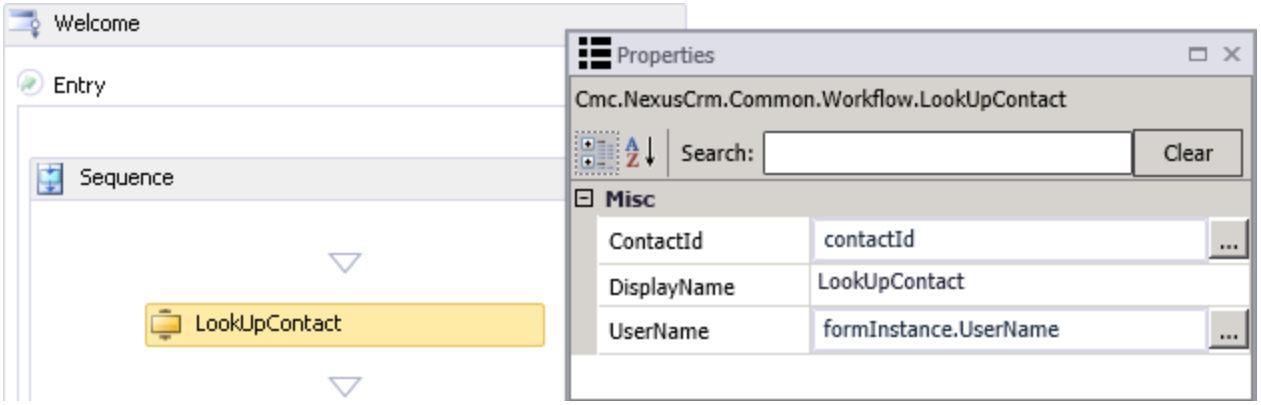
Initializing the Grid Using Workflow Activities

The first state in the form sequence workflows needs to be modified to initialize the grid for the child records.

1. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
2. Double-click the **first state** in the workflow. In our example it is the Welcome form.

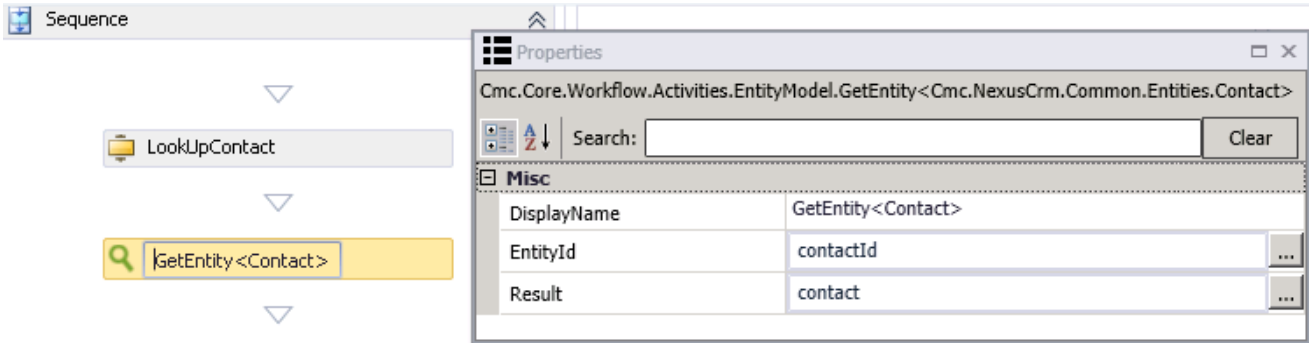


3. Drag a **Sequence** activity into the Entry section of the Welcome form.
4. Drag a **LookUpContact** activity into the new Sequence and specify the following properties:
   - ContactId: **contactIId** (This is a local variable of type Int32.)
   - Display Name: Specify a name or accept the default.
   - UserName: **formInstance.UserName**



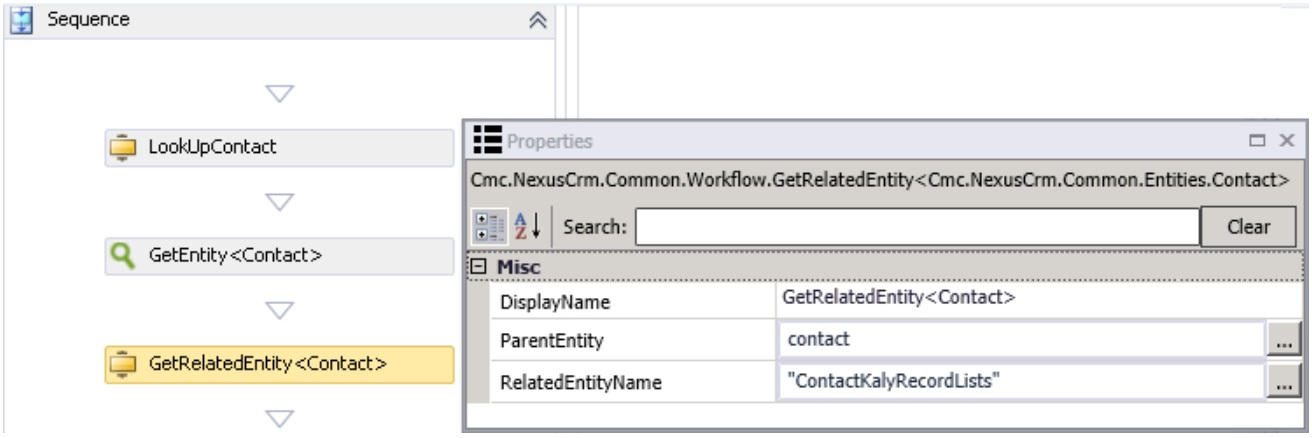
5. Drag a **GetEntity** activity below the LookUpContact activity and specify the following properties:

   - Type: **<Contact>** (Use the "Browse for Type" option to find this type.)
   - EntityId: **contactId** (This is a local variable of type Int32.)
   - Result: **contact** (This is an In/Out argument.)



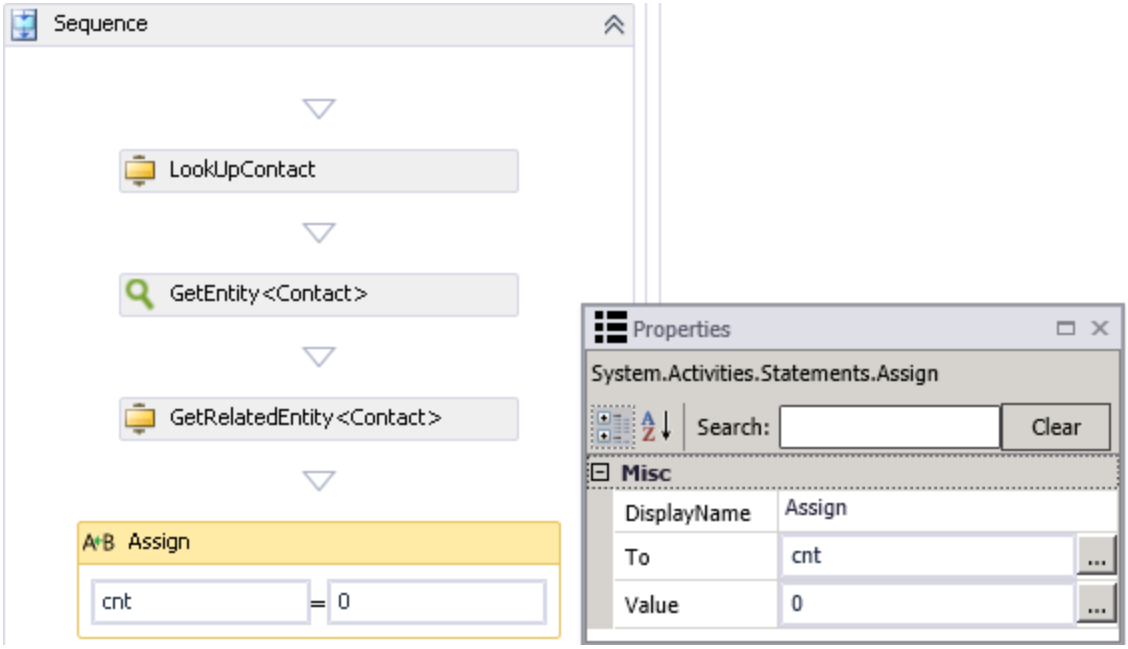
6. Drag a **GetRelatedEntity** activity below the GetEntity activity and specify the following properties:

   - Type: **<Contact>** (Use the "Browse for Type" option to find this type.)
   - ParentEntity: **contact** (This is an In/Out argument.)
   - RelatedEntityName: **"ContactKalyRecordLists"** (This is the logical identifier of the related entity that can be retrieved.)



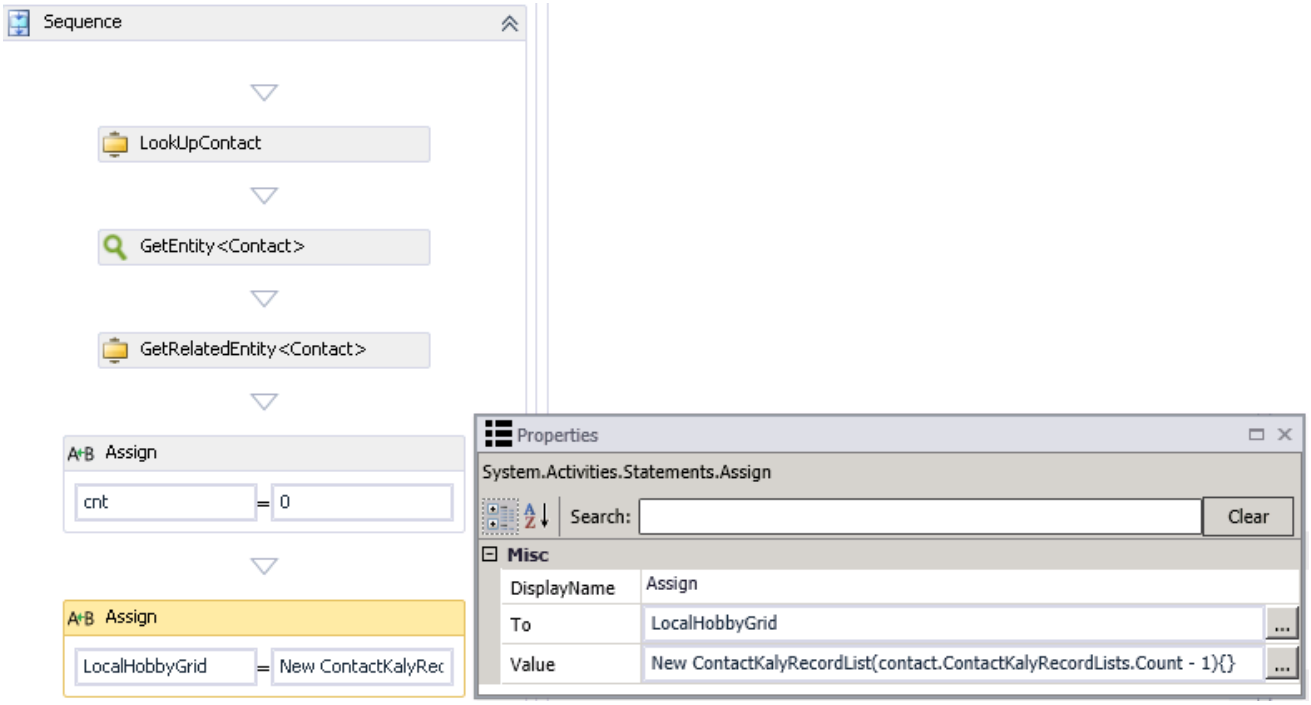
7. Drag an **Assign** activity below the GetRelatedEntity activity and specify the following properties:

   - To: **cnt** (This is a local variable of type Int32.)
   - Value: **0** (This value initializes the grid.)



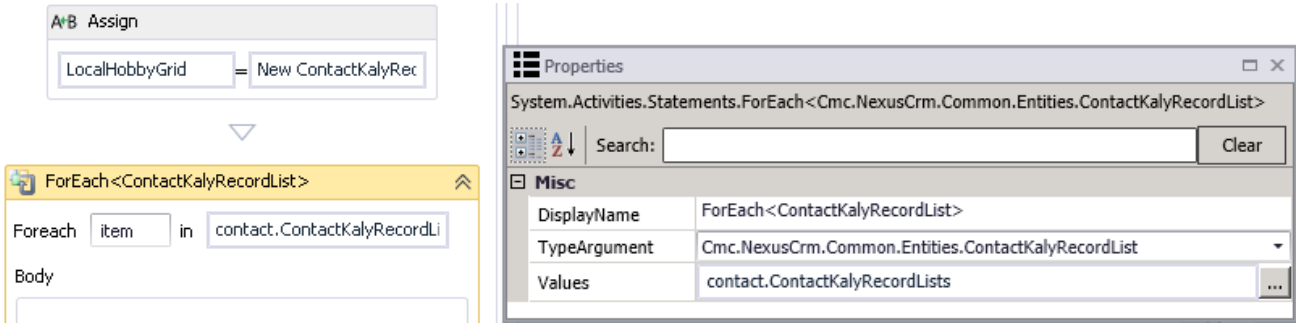
8. Drag another **Assign** activity into the Sequence and specify the following properties:

   - To: **LocalHobbyGrid** (This is a local variable of type Int32.)
   - Value: **New ContactKalyRecordList(contact.ContactKalyRecordLists.Count - 1){}** (This a local variable of type Cmc.NexusCrm.Common.Entities.ContactKalyRecordList[]. It associates the child records with the parent record.)



9. Drag a **ForEach** activity below the Assign activity and specify the following properties:

   - TypeArgument: **Cmc.NexusCrm.Common.Entities.ContactKalyRecordList**
   - Value: **contact.ContactKalyRecordLists**



10. Drag a **Sequence** activity into the Body section of the ForEach activity.
11. In the Sequence (in the Body section of the ForEach activity), drop an **Assign** activity for each row in the following table and type the indicated values:

"To" Field	Value
LocalHobbyGrid(cnt)	New ContactKalyRecordList
LocalHobbyGrid(cnt).KalyHobby	item.KalyHobby
LocalHobbyGrid(cnt).KalyHobbyStartDate	item.KalyHobbyStartDate

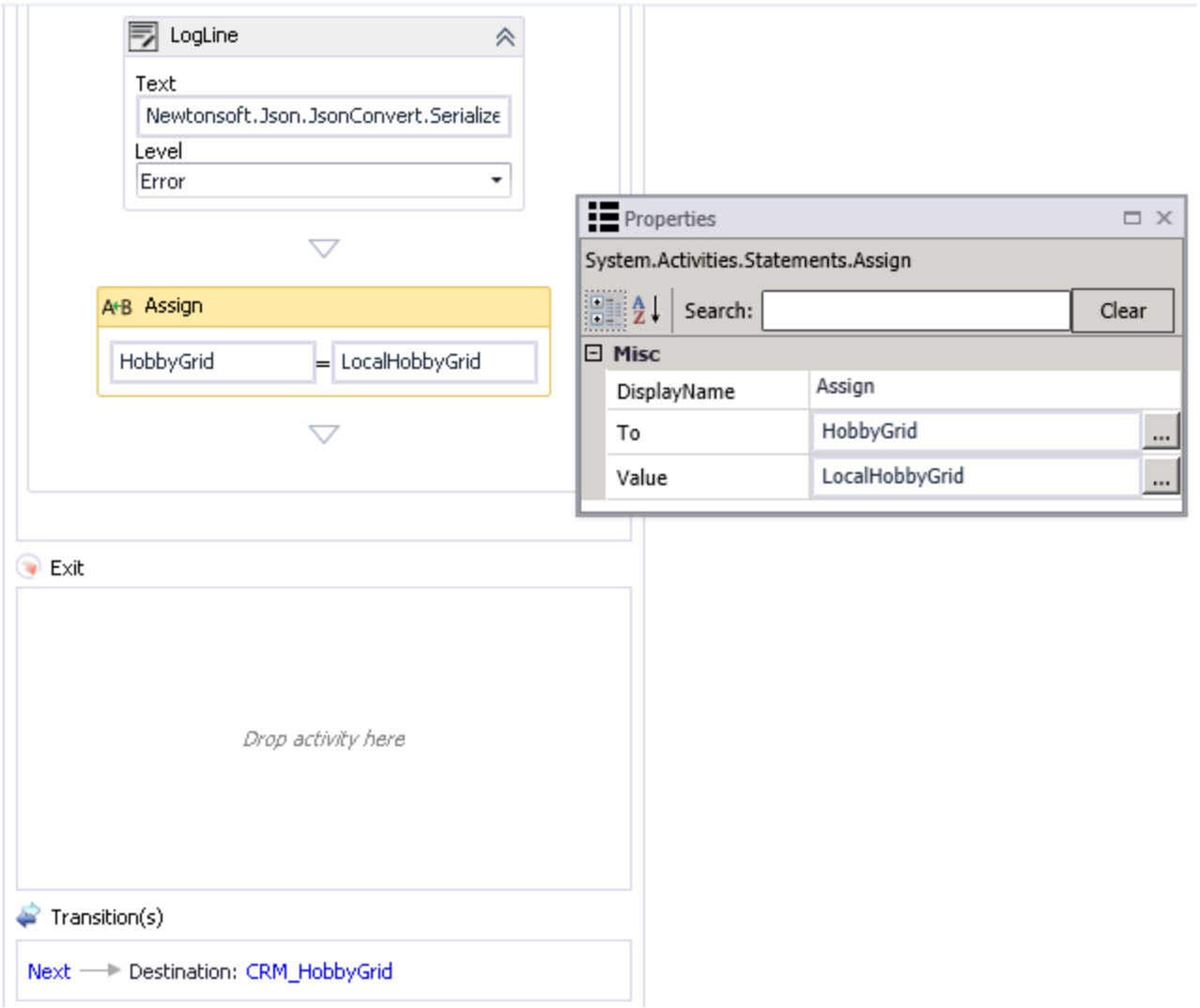
"To" Field	Value
LocalHobbyGrid(cnt).OtherReason	item.OtherReason
cnt	cnt + 1

12. (Optional) Drag a LogLine activity below the Sequence in the Body section of the ForEach activity and specify the following properties:

    - Level: **Information**
    - Text: **Newtonsoft.Json.JsonConvert.SerializeObject(LocalHobbyGrid,Newtonsoft.Json.Formatting.Indented)**

13. Drag an **Assign** activity below the LogLine activity and specify the following properties:

    - To: **HobbyGrid** (This is the name of the In/Out argument of type ContactKalyRecordsList[].)
    - Value: **LocalHobbyGrid** (This is the name of the variable of type ContactKalyRecordsList[].)



Google Map

You can use the Google Map component to embed a Google map in a form sequence. You can specify the map area using latitude and longitude coordinates, set zoom level and map height values, and make the map search-able.

Notes

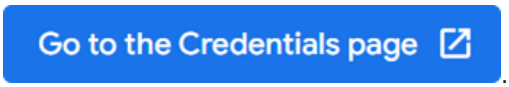
- The Google Map is always displayed in the language of the user's browser, i.e., Forms Renderer localization does not apply.
- If the form sequence contains a View Summary component, the summary view will display the longitude and latitude settings of the Google Map component. The summary view will not display the Search area specified by the user.
- Our recommendation is to use only one Google Map component per sequence, otherwise, if multiple Google Map components are used on same form and/or the sequence contains a View Summary, an error occurs and not all maps are displayed properly.

Prerequisites

1. To use the Google Map component, you need to sign up for your own API key.

   Follow the instructions at https://developers.google.com/maps/documentation/embed/get-api-key.

   For pricing information, refer to https://cloud.google.com/maps-platform/pricing/.

2. In the above instructions link, select 

.

   Select your project and then select 

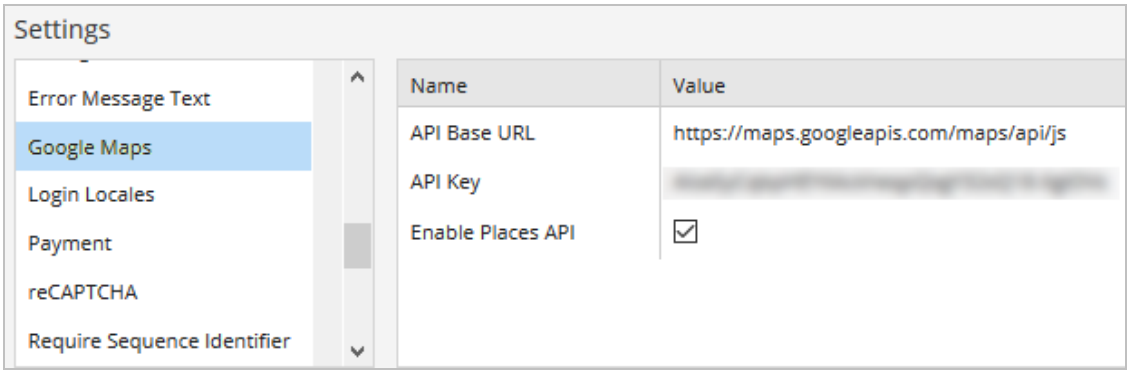
for the following APIs:

   - To enable the Google Maps feature – Geocoding API, Geolocation API, Maps Embed API, and Maps Javascript API
   - To enable search capabilities in Google Maps – Places API

3. In the Forms Builder Settings workspace, select **Google Maps**, populate the **API Base URL**, **API Key**, and select **Enable Places API**.

   The “Enable Places API” option is required when the “Map Search Visible” property is selected on the Google Map component to get a Search box to appear in the map. Otherwise the map is static and just shows the

area selected by the form designer.



Whenever you edit a field in a Settings panel with hidden fields (i.e., sensitive data that are removed from the client-side view like passwords, keys, and secrets), you must re-enter the hidden values, otherwise they will get cleared upon update and resave of all the Settings.

4. Select **Save Setting**.

Properties

Control Property Settings

Control Type	Google Map
Class	
Id	idaf8f15be-7979-c158-a852-0e07d10250cf
Latitude	26.3837862
Longitude	-80.0914254
Map Height	500px
Map Search Visible	☑
Map Zoom	15
Visible	true

Workflow Arguments

If Latitude and Longitude are bound, select argument type “Double”.

Name	Direction	Argument type	Default value
mylatitude	In/Out	Double	*Default value not supported*
mylongitude	In/Out	Double	*Default value not supported*

Rendered Component



- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
  - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
  - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
  - Binding is not supported for this property.
- **Latitude** of map. Must be a decimal value from -90 to 90, or a bound value which must start with {{vm.models., and end with }}. If the value is bound, select argument type “Double”.

  If you want an initial map position, you must supply coordinates and zoom. You can obtain coordinates of a place with online web sources, e.g., https://www.gps-coordinates.net/ and https://www.latlong.net/convert-address-to-lat-long.html.
- **Longitude** of map. Must be a decimal value from -180 to 180, or a bound value which must start with {{vm.models., and end with }}. If the value is bound, select argument type “Double”.

  If you want an initial map position, you must supply coordinates and zoom. You can obtain coordinates of a place with online web sources, e.g., https://www.gps-coordinates.net/ and https://www.latlong.net/convert-address-to-lat-long.html.
- **Map Height** is the height of the map DIV specified in px, em, or %. Default 500 px. Not bindable.
- **Map Search Visible** makes the text box for a search visible. If you do not want the user to search, clear this.

  Search requires an additional license from Google. Refer to Prerequisites.
- **Map Zoom** is the initial zoom level, default 8. The value must be between 1 and 20. If it is bound, it must start with {{vm.models., and end with }}.
  - 1: World
  - 5: Landmass/continent
  - 10: City
  - 15: Streets
  - 20: Buildings

If you want an initial map position, you must supply coordinates and zoom. You can obtain coordinates of a place with online web sources, e.g., https://www.gps-coordinates.net/ and https://www.latlong.net/convert-address-to-lat-long.html.

- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)

HTML

You can use the HTML component to create form content marked up with standard HTML tags. You can also use this component to add custom style fragments or scripts to a form.

> ⚠ A custom JavaScript embedded in an HTML component may fail for several reasons outside of Forms Builder. It might be blocked by the newer versions of the browsers, security improvements to our corporate network, or other factors related to the clients' browsers. Embedding JavaScript into an HTML component is considered custom code and is the responsibility of the client.

Properties

Control Property Settings

Control Type	HTML
Class	
HTML	Please attach the following documents to your application: - Cover letter - Resume - Current transcript
Id	ide860ef7b-c46a-dd65-9df3-cf799d00622c
Visible	true

Rendered Component

Please attach the following documents to your application:

- Cover letter
- Resume
- Current transcript

For more examples of HTML components, please refer to Welcome and Confirmation Forms.

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.

  The Class `forms_builder_page_break` enables you to set page breaks in PDF files with DocuSign signatures. For more information, refer to Error Code "TAB_OUT_OF_BOUNDS" - DocuSign.

- The **HTML** property enables you to format rendered output using a subset of standard HTML markup as a fragment of an HTML page. That is to say, <!DOCTYPE html>, <html>, <head>, <body> and <form> tags are not appropriate in an HTML fragment. While they may not harm the page, they do have the potential to create silent Renderer errors or cause the page rendering to fail completely. The HTML validation parser will point out errors in the HTML fragment and mark them as a warning but will not attempt to enforce rules. Warnings should be corrected to avoid unexpected results.

  An example of an HTML fragment is:

  <h2 class=”myclass”>Campus View</h2>

  <img src="IMG_2671.jpg" alt="Campus View" style="width:304px;height:228px;">

  Also possible are <script> and <style> fragments. This allows a great deal of customization. A model value can be addressed in JavaScript with “window.vmModelsRef”. If you had an argument in a workflow “myKey”, which would also be a model value “vm.models.myKey”, then in external HTML JavaScript this can be addressed with “window.vmModelsRef.myKey” or “window.vmModelsRef[‘myKey’]”.

  Similarly, parameters for the Renderer URL that are addressed with formInstance.QueryParams.DataDictionary(“myKey”) or special case formInstance.QueryParams.DataDictionary (“addonQueryParams”) in the workflow (refer to Renderer URL Query Parameter), can be addressed in external HTML JavaScript as “window.vmQueryParamsRef.myKey” or “window.vmQueryParamsRef ['myKey']. This would allow you to pass information in the Renderer URL to your custom JavaScript (or even to a custom style via a binding). Of course, depending on how it is to be used, make sure your JavaScript and/or workflow validates the information passed, or this could be a security risk.

  You can bind the HTML property to a workflow argument. Refer to Bind HTML to an Argument.

- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
  - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
  - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
  - Binding is not supported for this property.
- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==,

>=, <=). If comparing to a string, it must be in single quotes.

- (true and false must be all lowercase)

Access Model Values Using JavaScript

The following is an example of creating client-side code to do something on a page. Suppose you have a name, address, and state that a student has entered on a page and you want to add a check box which adds the functionality to copy that name, address, and state to fields below that. The check box has the text *"Mailing address different than residence address?"*. The student would select the check box if the mailing address and residence address are different. Furthermore, you want to only make the mailing address fields visible if the student has selected the check box.

The top of the page has 3 fields: 2 text boxes for name and address and a drop-down list to select the state. The job is to copy the model values for the 2 fields and the drop-down list value to fields below.

The models are:

- vm.models.studentName
- vm.models.studentResidenceAddress
- vm.models.studentResidenceState.

Below that you have the fields:

- vm.models.studentName
- vm.models.studentMailingAddress
- vm.models.studentMailingState

The first field has the same model name (because it is assumed the student's name is not going to change).

The following HTML will achieve the goals above:

```
<div>
  <input type="checkbox" ng-model="vm.models.myCheckBoxValue" onclick="myChange(this)" />
</div>
<script type="text/javascript">
  function myChange(chkBx) {
  if (chkBx.checked){
  // vm.models values can be accessed externally with the global variable vmModelsRef (Forms Builder 3.3.
or above only)
  vmModelsRef.studentMailingAddress = vmModelsRef.studentResidenceAddress;
  vmModelsRef.studentMailingState = vmModelsRef.studentResidenceState;
  } else {
  // Clear the fields (not really necessary if you use the checkbox model value to determine if it was checked)
  vmModelsRef.destination1 = "";
  vmModelsRef.studentMailingState = "-1";  // Basically a value that is not in the State list
  }
  }
```

```
</script>
```

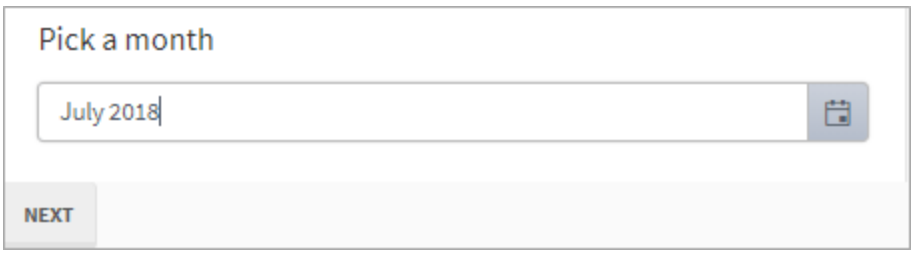
Explanation:

- The `onclick` event of the `checkbox` runs the code `myChange` in the script block.
- The first statement checks if the `checkbox` is selected. If it is, copy the values.
- The internal `vm.models.xxx` values are accessed with the global variable `vmModelsRef.xxx`.
- The `checkbox` value can be made accessible in the workflow with an argument you create `myCheck-BoxValue`(Boolean).
- The Visible property of the mailing fields is set to `vm.models.myCheckBoxValue`. When the checkbox is selected, they appear.
- It is not necessary to handle the `studentName` in the code because it is the same value for both residence and mailing fields.
- The value of the drop-down list is not the same as the text shown, so the way to set the mailing state is to set the same value. This is the vm.models value for both. This means they both must have the same Lookup Query, Workflow Initialized List, or Value List and the same Text Member (Lookup Display Member) and Value Member (Lookup Value Member). (Refer to Drop-down List component).

If you are debugging this in a browser developer tool (F12), the statement “debugger;” inserted in the code will cause the code to break in the debugger so that values can be examined.

DatePicker Widget

You can implement a custom DatePicker control on a form with Forms Builder's HTML component using the HTML code and script below.



The documentation for the example it was derived from and the API for the DatePicker is found here:

- https://demos.telerik.com/kendo-ui/datepicker/index (refer to 2nd control on page for a live example of what it does)
- https://docs.telerik.com/kendo-ui/api/javascript/ui/datepicker (API for control - all properties, methods, and events used below)

```
<!-- Note in the following example "myDate" is defined in a workflow as a string argument. In order to store it as a DateTime object, that can either be done by converting it to and from one in a workflow, or by changing
```

```
the control below to handle a date object for both input and output. -->
<div id="example">
  <div class="k-content">
    <!-- Regardless of what other HTML you use with this custom control, "input" is the element the script works on to create the control. Styling is something you work on after the functionality works. The "value" below is input to the control from the workflow or from a durable instance if the page is returned to after picking a value. It is interpolated from the model value when the view is loaded. If no value is available, it displays default input text. -->
    <input id="myMonthpicker" value="{{vm.models.myDate}}" title="monthpicker" style="width: 100%" />
  </div>
</div>
<script>
  $(document).ready(function() {
    function onMyDateChange() {
      // Using this event handler makes it easier to manipulate the output of the control.
      // The value of the control is obtained with the value function on the control "this".
      var val = this.value();
      // You define the string format you want to present to the workflow with this format.
      var dtStr = kendo.toString(val, 'MMMM yyyy');
      // The model is accessed externally with "vmModelsRef" so we assign "vm.models.myDate" with this statement.
      // This will appear in the workflow as argument "myDate" defined as string.
      vmModelsRef.myDate = dtStr;
      // For FB 3.5 and above you can update Debug JSON displayed at the bottom of the page with the following method.
      // Debug JSON does not update automatically from outside the FB application.
      vmModelsRef.mergeVmModelsRef();
      // Otherwise you can see the new value in the F12 console window of the browser.
      console.log("My Date = " + dtStr);
    }

  // This is the definition of the kendo control using some of the properties and one event
    $("#myMonthpicker").kendoDatePicker({
      // When the value is changed in the control, this event is fired.
      change: onMyDateChange,
      start: "year",
      depth: "year",
      // This defines the format the control is expecting for input. Change this if a different format is used.
      format: "MMMM yyyy",
      dateInput: true
    });
  });
```

```
</script>
```

Set Default Values for Form Fields

You can use script in the HTML property of the HTML component to set default values for fields at the form level. The script references the argument specified in the **Model** property of the component whose default value is set. The script is case sensitive.

It is best practice to put all the HTML components used to set default values in a separate, invisible (Visible=false) form section so that they don't take up extra white space.

Examples:

Text Box

For a Text Box with `Model = vm.models.myText`, a hidden HTML component with the following script will set the default value displayed on the rendered form to "*This is some initial text*".

```
<script>
  vmModelsRef.myText = "This is some initial text";
  vmModelsRef.mergeVmModelsRef();
</script>
```

Control Type	HTML
Class	
HTML	<script> vmModelsRef.myText = "This is some initial text"; vmModelsRef.mergeVmModelsRef(); </script>
Id	id6f09ddb8-ee01-560b-5a50-f0ac75e825db
Visible	false

Drop-down List

For a Drop-down List with `Model = vm.models.studentEntity.CampusId`, a hidden HTML component with the following script will set the default value to the name of the Campus with Id = 1.

```
<script>
  vmModelsRef.studentEntity.CampusId = 1;
  vmModelsRef.mergeVmModelsRef();
</script>
```

This example requires a CreateEntity or GetEntity activity for the studentEntity in the sequence workflow. If no entity object is created, the default value cannot be set.

Numeric Text Box

For a Numeric Text Box with `Model = vm.models.myNum`, a hidden HTML component with the following script will set the default value displayed on the form to 12.

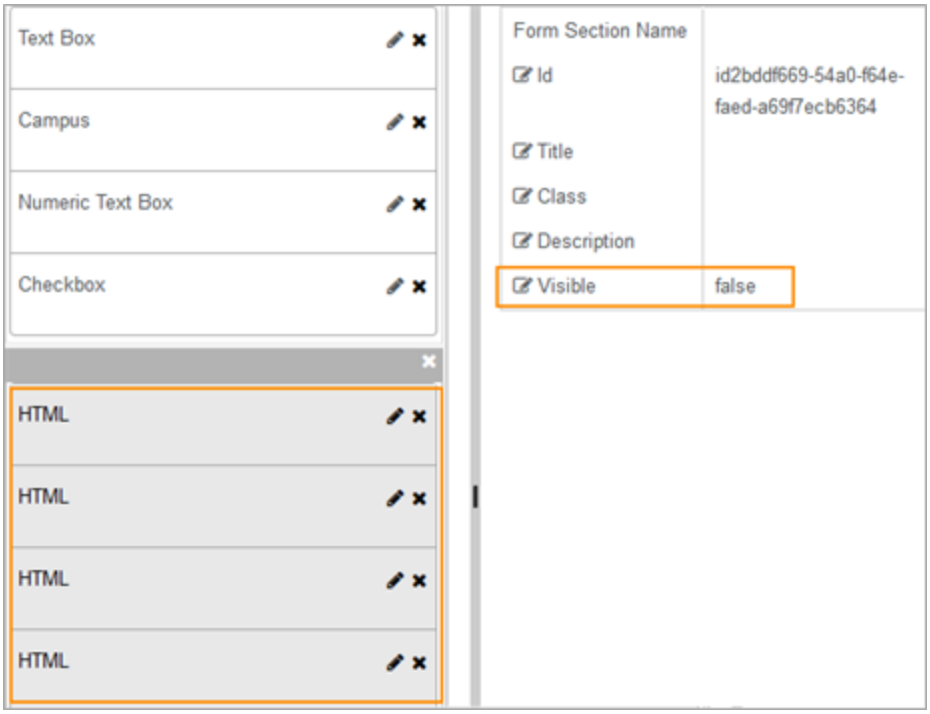
```
<script>
  vmModelsRef.myNum = 12;
  vmModelsRef.mergeVmModelsRef();
</script>
```

Checkbox

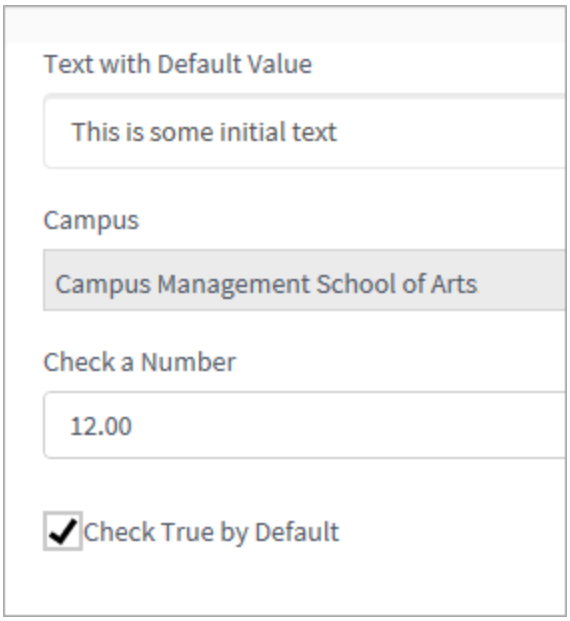
For a Checkbox with `Model = vm.models.myBool`, a hidden HTML component with the following script will set the default value to true (selected).

```
<script>
  vmModelsRef.myBool = true;
  vmModelsRef.mergeVmModelsRef();
</script>
```

HTML components that set default values should be made invisible, either by setting Visible=false on the HTML components or by placing the HTML components in a form section where Visible=false.



The Text Box, Drop-down List, Numeric Text Box and Checkbox examples with hidden HTML components setting default values will be rendered as shown below.

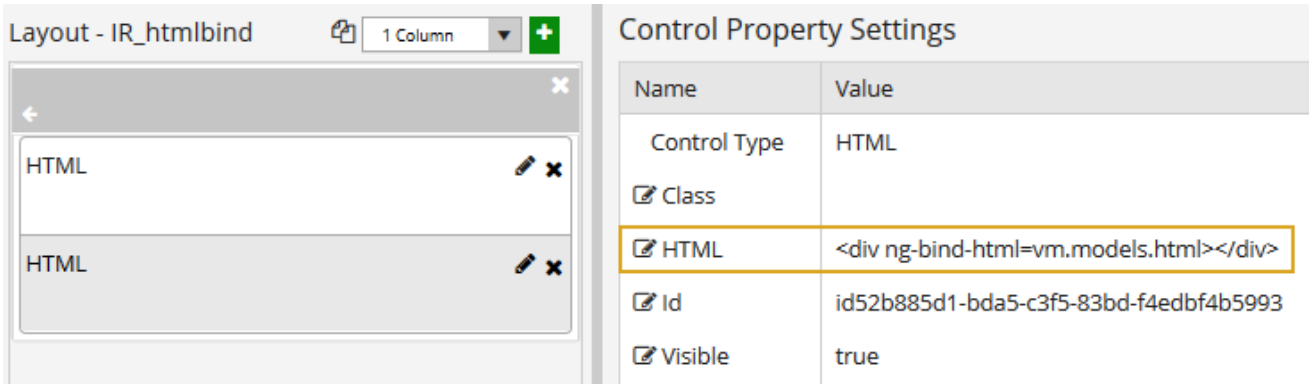


Bind HTML to an Argument

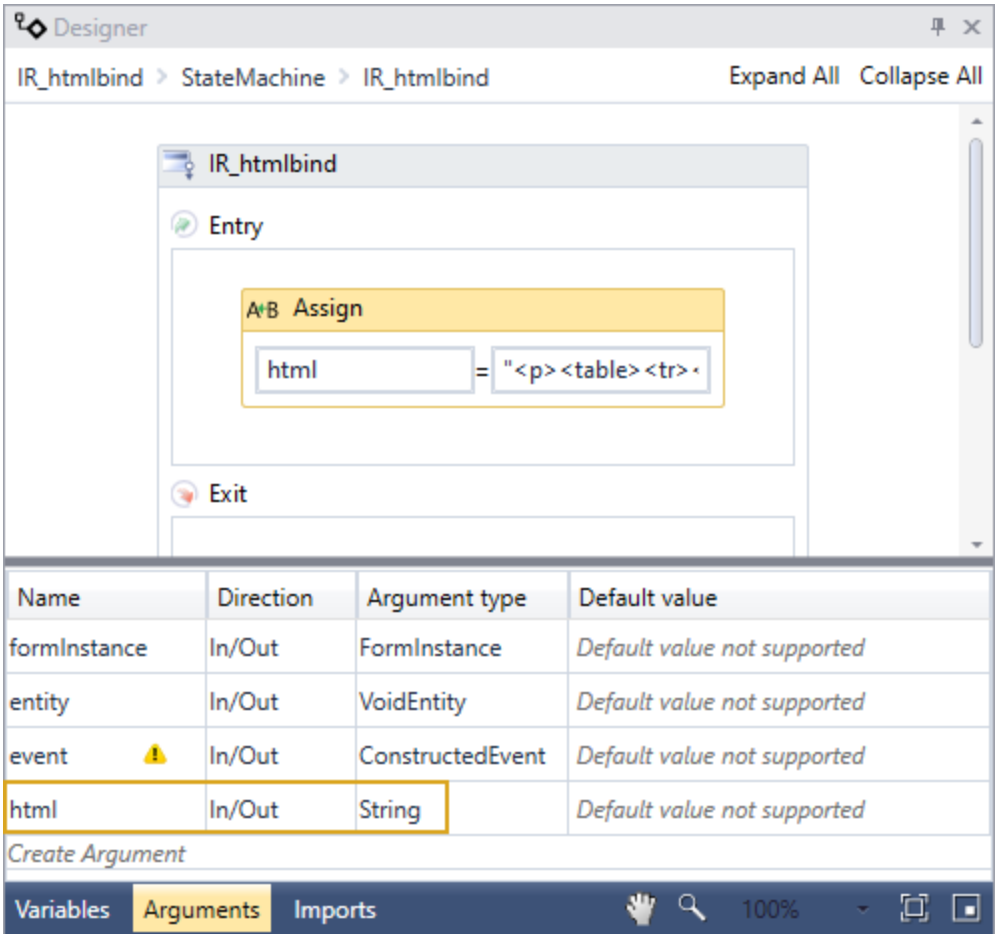
In Forms Builder 3.7 and later, you can populate content on a form by binding the HTML property to an argument and then use the argument in an Assign activity.

1. Specify the value of the HTML property in the HTML component as:

   ** **



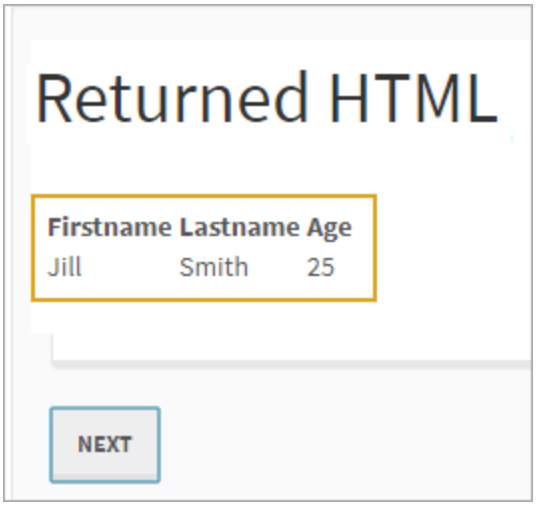
2. Drop an **Assign** activity into the workflow and specify the HTML content as a String value (i.e., enclosed in quotes).



In our example, the html argument is assigned the following value:

" <table><tr><th>Firstname </th><th>Lastname </th><th>Age </th></tr><tr><td>Jill</td><td>Smith</td><td>25</td></tr></table> "

3. The rendered form displays the value of the argument.



Hyperlink

You can use the Hyperlink component to point to a URL from a form.

Do not use the Hyperlink component in a form that will be converted to PDF with the PrintUrlToPdf workflow activity.

Properties

Control Property Settings

Control Type	Hyperlink
Class	
Hyperlink Target	_self
Id	id48ec358e-0d11-2485-fb07-ba60afd32720
Link Text	Google This
Tab Index	
Url	http://Google.com
Visible	true

Rendered Component

Google This

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Hyperlink Target** is set to `_self` by default. This property sets the target window or frame in which to display the webpage linked to when the Hyperlink control is selected. Other values are `_blank`, `_parent`, and `_top`. For more information, refer to https://www.w3.org/TR/html5/browsers.html#browsing-context-names.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.

- Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
- Binding is not supported for this property.

- **Link Text** is the link text displayed in the form. Binding is not supported for this property.
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
  - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
  - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
  - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

    The tab index value should not match another control's tab index.
  - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.
- **Url** is the address of the linked website.
  - It starts with http:// or https://.
  - If spaces are required, they should be replaced with + or %20.
  - If it is bound, it must begin with {{vm.models. and end with }}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)

IFrame

You can use the IFrame component to embed an HTML document or a website into a form. An IFrame (Inline Frame) can insert content from another source into a form, for example, a signed form received from an e-signature provider.

> ⚠ The source files for the content in the IFrame must reside in the same domain as the form.

If you use mobile signing with Adobe Sign, DocuSign, or SignNow, we recommend that you apply the following styles to optimize the portrait view of your e-signature forms:

```
.cmc-div-iframe
{
   min-height: 500px;
}
.cmc-div-iframe > iframe
{
   border: 0;
   overflow: hidden;
   min-width: 600px;
}
```

You can apply the styles in two ways:

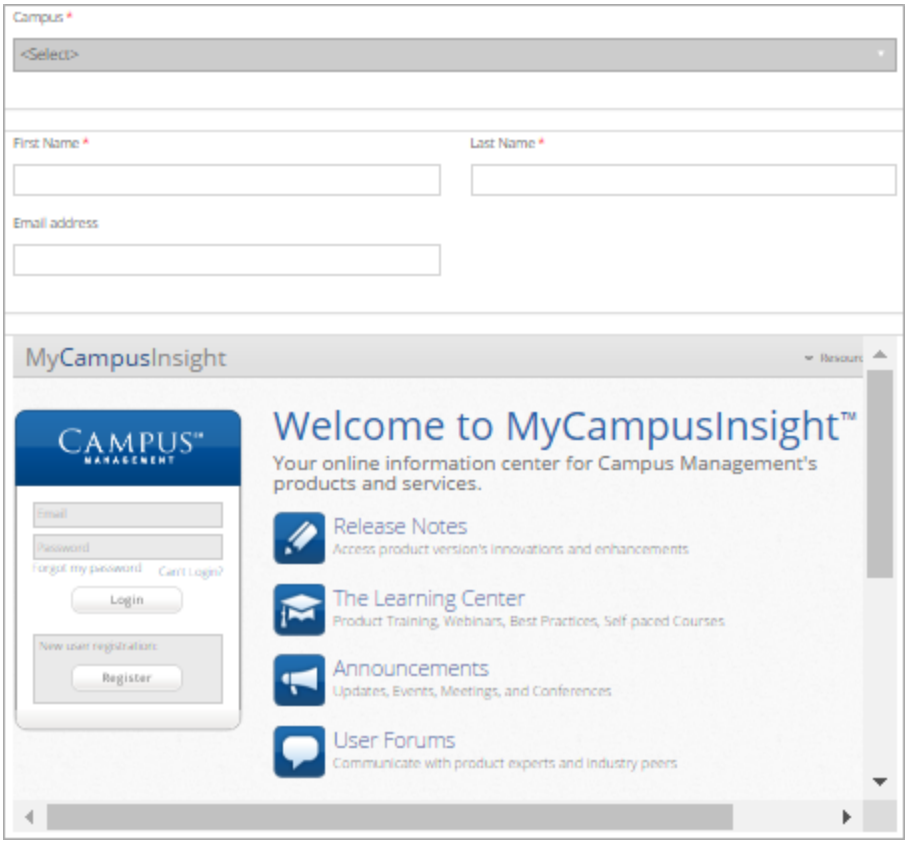
- By adding the styles to an HTML component on the *Default-Frame* form. Refer to CSS Files (Style Sheets).
- By referencing a CSS file using Custom Content and then applying the CSS to the sequence.

Properties

Control Property Settings

Control Type	IFrame
Class	
Id	idd851ec70-7f4f-ee74-6fb8-2d3edb1462fa
Name	MCIframe
Url	http://www.mycampusinsight.com
Visible	true

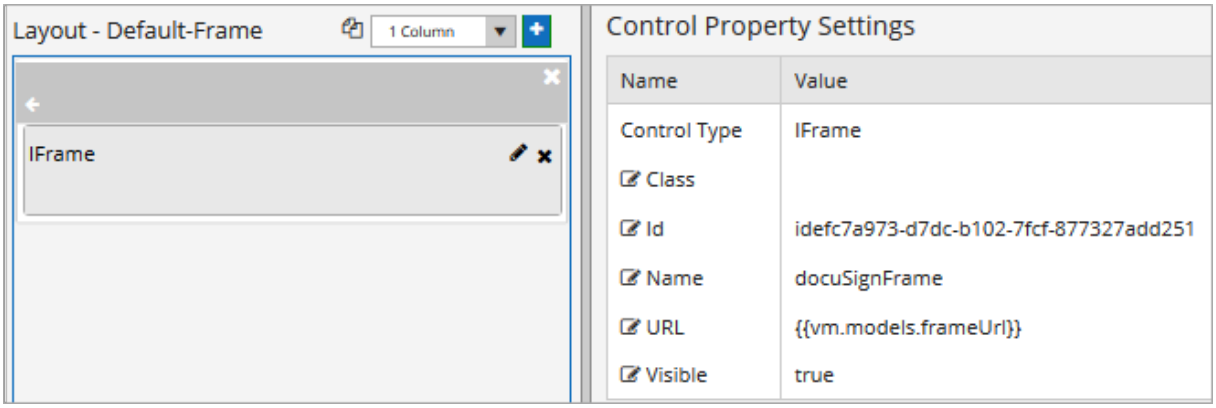
Rendered Component



- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Name** identifies the IFrame.
- **Url** is the address of the document to embed in the IFrame.
    - It starts with http:// or https://.
    - If spaces are required, they should be replaced with + or %20.
    - If it is bound, it must begin with {{vm.models. and end with }}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

When the IFrame component is used capture an e-signature document, the Url property must be specified as `{{vm.models.frameUrl}}`.

> Be sure to use the exact casing shown here.

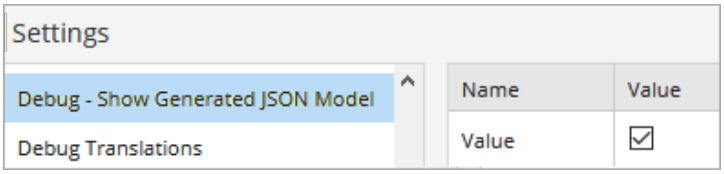


- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)

JSON Debug Info

You can use the JSON Debug Info component to help debug forms in production mode where the JSON information is needed for a single form and not the entire site. The component is handy if you are debugging a single form on a production website. You can use the Visibility property to turn it off easily or even enable it from a workflow.

To enable the display of JSON debug information, in the Forms Builder Settings workspace, select **Debug - Show Generated JSON Model**, select the **Value** check box, and select **Save Settings**.



When the JSON Debug Info component is added to a form, the values for objects on the form are shown at the bottom of the rendered form. This data can be helpful for troubleshooting, especially for complex components on a page where knowing the data that is available to a workflow during a transition will aid in debugging a workflow.

On form load, the Generated Model for Debugging section shows the Renderer Media Variables. As the form fields are populated with values, the debugging section displays the values associated with each object on the form, i.e., all model entity data on the page and new values entered are displayed in real time.

JSON Debug Info output is not rendered when the sequence contains a View Summary component or a PDF is created.

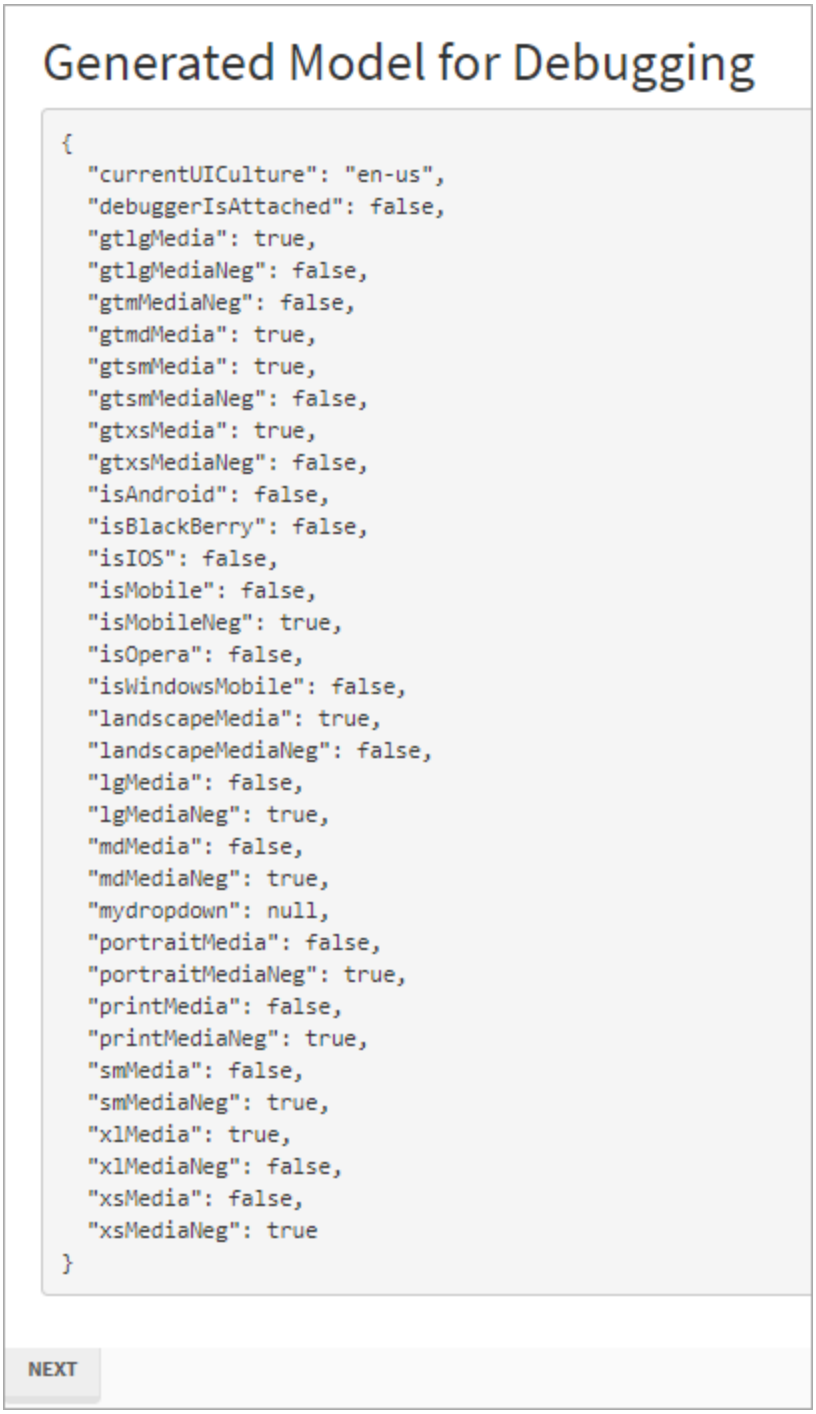
Properties

Control Property Settings

Control Type	JSON Debug Info
Class	
Visible	true

If JSON debugging is enabled for the site and the JSON Debug Info component is added to a form, the JSON information on the rendered form will be duplicated.

Rendered Component



- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Visible** makes the control visible or hidden.

- Can be bound to a workflow argument or another control's value. This property is dynamic.
- A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

Label

You can use the Label component to assign a stand-alone label, that is, a label not associated with a control (controls have their own labels).

Properties

Control Property Settings

Control Type	Label
Class	
Id	26e72270-1d54-3498-bf80-f64ad735717e
Label	Required Documents
Visible	true

Rendered Component



This example uses the Label component for the text "Required Documents" and the HTML component for the listed items.

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Label** is the value displayed in the label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

  You can use HTML markup to customize the appearance of the label text (font, color, etc.).

If label text is not specified, some empty space will be displayed on the form.

The style of the label control is a default Bootstrap style. It can be modified. For more information, refer to Modify the CSS for the Label.

- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)

Modify the CSS for the Label

The following styles determine the default styles for labels. They can be modified as needed. Copy the styles and save your changes in a custom style sheet. For more information, refer to Custom Content and Custom Styles.

Default Style for Labels

The default style for the Label component can be modified by uncommenting the style definition below.

```
/* Defaults for a Label component. Applying one or more of these styles will override a default bootstrap style, applying all will make it look like a control label. For the default theme you may just want to set font-size: 100% (bootstrap sets it to 75%).
*/

/*.cmc-div-label.form-group span {
   white-space: normal;
   display: inline-block;
   color: rgb(102,102,102);
   background-color: white;
   font-weight: 400;
   text-align: left;
   font-size: 14px;
   font-style: normal;
   line-height: normal;
   font-family: "Source Sans Pro","Helvetica Neue",Helvetica,Arial,sans-serif";
   }*/
```

You can adjust the style per your needs. For example, you may just want to specify a font size of 100% using the following CSS code:

```
.cmc-div-label.form-group span {
  font-size: 100%;
  }
```

You can also use an HTML component to style the label.

Text Wrapping on Labels

The text wrapping on the Label component is controlled by the style definition shown below.

```
/* This rule applies automatically to divs surrounding the labels to make them wrap around */

.label-wrap > span {
  white-space: normal;
  }
```

Locale

You can use the Locale component to display the cultures supported by controls in a form sequence. A culture defines specific information for number formats, week and month names, date and time formats, currencies, etc.

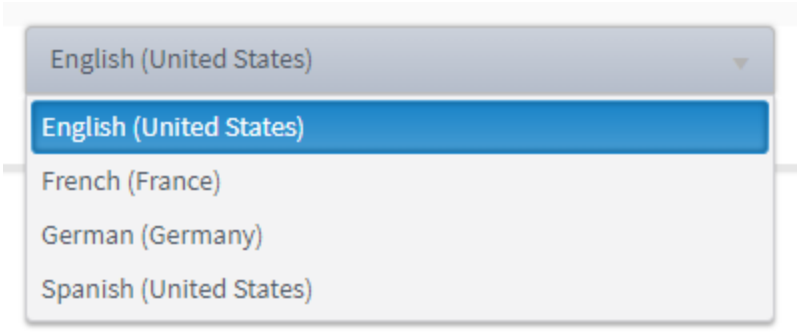
In Form Designer, you select a set of cultures and specify which one will be the default. In Forms Renderer, the Locale component will list all the locales selected in Form Designer, with the default culture automatically selected on initialization of the form. The user's locale selection is preserved on all forms in the sequence (refer to Internationalization).

Properties

Control Property Settings

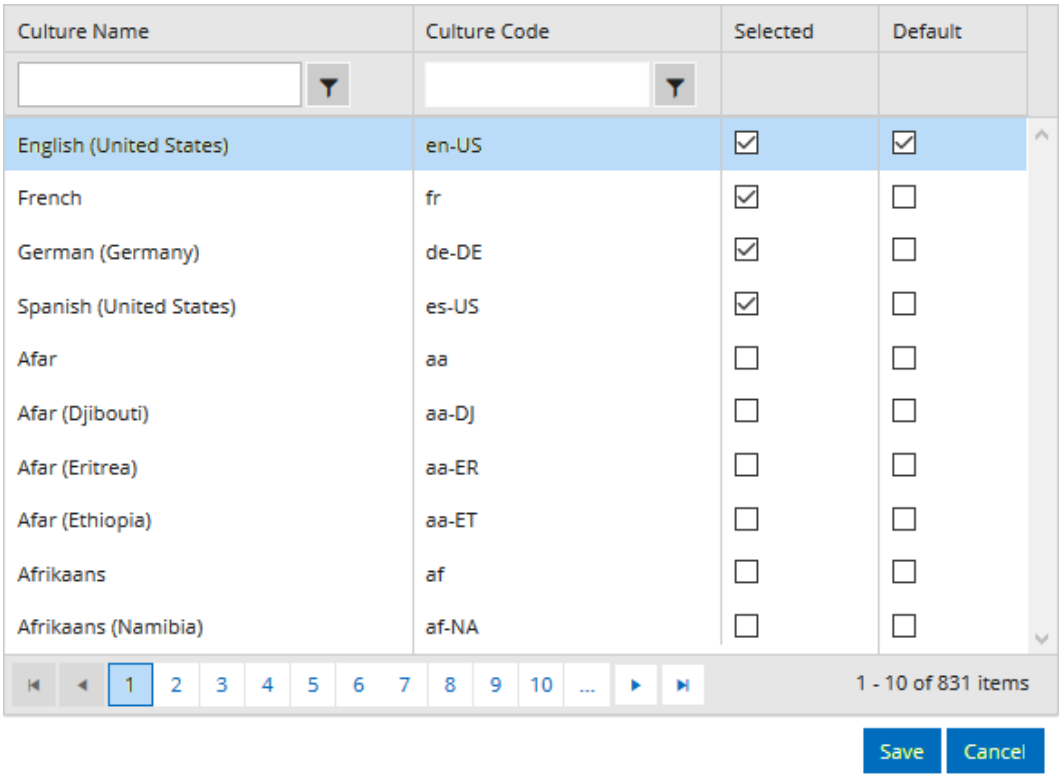
Control Type	Locale
Class	
Cultures	Edit...
Id	id22a87640-b8b2-8544-74d9-c8651d90c799
Label	
Visible	true

Rendered Component



- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Cultures** — This property displays the full list of 700+ locales supported by Kendo controls.

Select the **Edit** button to select the cultures to be listed in the Locale component in a form. Designate one of the selected cultures as the default. After you have saved your selections and you select Edit again, the selected cultures will be displayed at the top of the list.



- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Label** is the value displayed in the label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].

- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

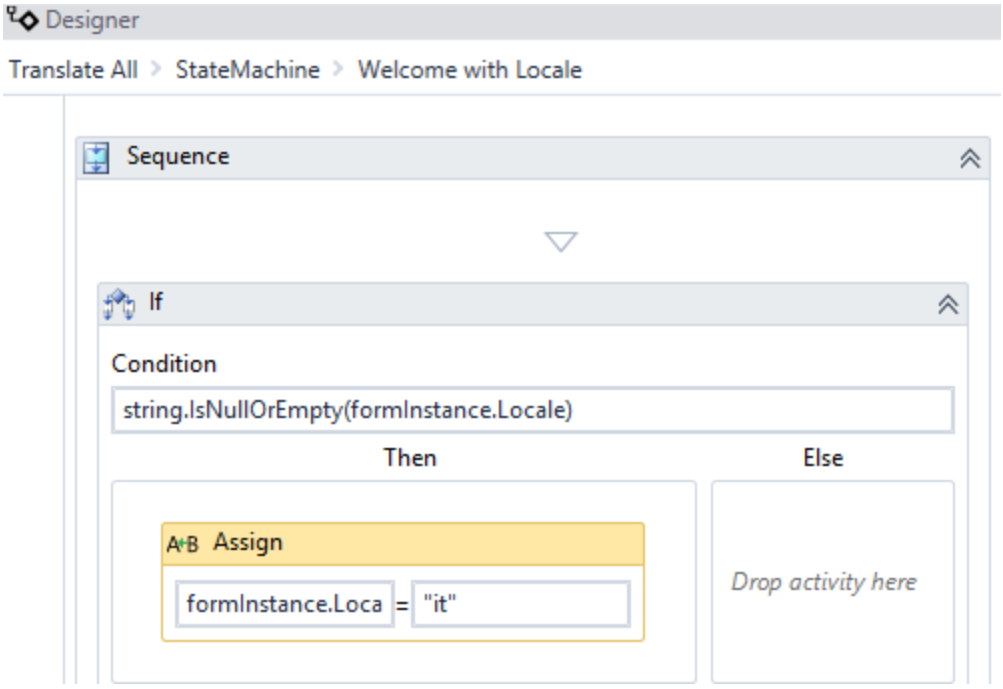
Locale Assignment Using Workflow

You can set the default locale for a sequence using an Assign activity, e.g., formsInstance.Locale="it" (for Italian).

If the default locale is set in a workflow, the following condition needs to be provided in an If activity:

```
string.IsNullOrEmpty(formInstance.Locale)
```

The condition ensures that a user can update the value in the rendered Locale component and keep the updated value when selecting Next / Back.



Masked Text Box

You can use the Masked Text Box component for text that must conform to a specific format, e.g., social security numbers.

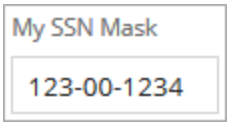
If the Masked Text Box component is used to specify a mask for phone numbers and the mask in Forms Builder differs from the default mask for phone numbers in Anthology Student, unexpected results may occur. The default mask for phone numbers in Anthology Student is "(###)###-####". The mask for any phone number on a Forms Builder form should align with this mask.

Properties

Control Property Settings

Control Type	Masked Text Box
Class	
Disabled	false
Format	###-##-###
Id	ef54480c-54e3-599e-7f1f-02e03b9c4d54
Label	My SSN Mask
Model	vm.models.myMask
Placeholder	
Read-only	false
Required	false
Required Message	
Tab Index	
Tooltip	
Tooltip Duration	750
Validation Message	
Validation Regex	
Visible	true

Rendered Component



Workflow Arguments

Name	Direction	Argument type	Default value
myMask	In/Out	String	*Default value not supported*

For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Disabled** sets a control to disabled.
    - Must be true or false, or a binding beginning with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Format** controls the input format for masked text values. The Mask Rules table lists the valid input characters.

Mask Rules

Rule	Description
0	Digit. Accepts any digit between 0 and 9.
9	Digit or space. Accepts any digit between 0 and 9, plus space.
#	Digit or space. Like 9 rule but allows also (+) and (-) signs.
L	Letter. Restricts input to letters a-z and A-Z. This rule is equivalent to [a-zA-Z] in regular expressions.
?	Letter or space. Restricts input to letters a-z and A-Z. This rule is equivalent to [a-zA-Z] in regular expressions.
&	Character. Accepts any character. The rule is equivalent to \S in regular expressions.
C	Character or space. Accepts any character. The rule is equivalent to . in regular expressions.
A	Alphanumeric. Accepts letters and digits only.
a	Alphanumeric or space. Accepts letters, digits and space only.
.	Decimal placeholder. The decimal separator will be taken from the current culture used by Kendo.
,	Thousands placeholder. The display character will be taken from the current culture used by Kendo.
$	Currency symbol. The display character will be taken from the current culture used by Kendo.

- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Label** is the value displayed in the label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.
    - The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
    - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
    - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
    - The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
    - If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

  If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

  If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

  Construction of the model in the workflow is done by assigning data from a provider.
- **Placeholder** is the prompt text displayed in a ghost style in an input box when nothing has been entered.
    - If this property is bound, it must start with {{vm.models. and end with }}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Read-only** makes the control read-only. It is set to false by default. If you want the component to be read-only, set the property value to true. It is typically used for an input box.

- Can be bound to a workflow argument or another control's value.
- A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
- If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

- **Required** makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.
  - Can be bound to a workflow argument or another control's value.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
  - If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)
- **Required Message** is optional. It overrides the default "Required" message.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
  - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
  - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
  - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

    The tab index value should not match another control's tab index.
  - A blank value (default) will not add a tab index in the HTML.

For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.

- **Tooltip** is the value to display when hovering over the control's label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Validation Message** is the message shown when the user's input does not match the pattern of the regular expression defined in the Validation Regex property.
- **Validation Regex** is the regular expression pattern to validate the input. Use site like RegExLib.com to search and test Regex patterns, or construct your own with Regex tools like Expresso. If the user input does not produce a Regex match, the "Validation message" will be displayed.

  Examples:
    - Regex that enforces a minimum of 11 digits for a phone number where the first digit must be "1": `1\d{10}`
    - Regex that enforces a phone number in the format (###)###-####: `^\([0-9]{3}\)[0-9]{3}\-[0-9]{4}$`
    - Regex that matches a hyphen-separated social security number: `^\d{3}-\d{2}-\d{4}$`
- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
    - When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

Multiselect

You can use the Multiselect component to select multiple values from a list of values.

Forms Builder supports the following types of Multiselect controls:

- Default Multiselect controls for fields in the CampusNexus entity model. Built-in Lookup Queries retrieve the list values for default Multiselect controls. Refer to example below,
- Custom Multiselect controls using a Value List defined using the Edit button on the Value List property. Refer to Custom Multiselect with Value List.
- Custom Multiselect controls using a Workflow Initialized List defined using the Edit button on the Value List property. Refer to Custom Multiselect with Workflow Initialized List.

The example below shows a default Multiselect control for the Ethnicities field.

Refer to Multiselect for Single Property Collections for a special use case for this Multiselect component.

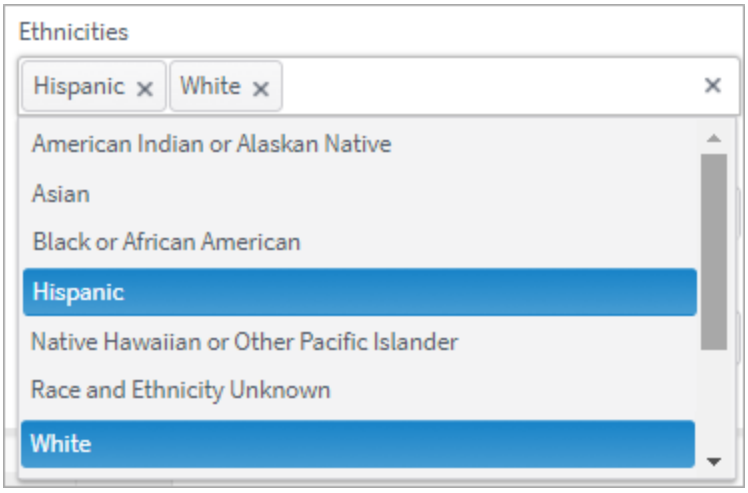
Properties

Example: Ethnicities

Control Property Settings

Control Type	Multiselect
Class	
Disabled	false
Filter Type	contains
Header Template	
Id	id90204daa-251d-476c-6ce9-dd327083207d
Item Template	
Label	Ethnicities
Lookup Display Member	Name
Lookup Query	Ethnicities?$select=Code, Name, Id&$filter=IsActive eq true&$orderby=Name
Lookup Sort Member	Name
Lookup Translation Members	Name, Code
Lookup Value Member	Id
Model	vm.models.prospectInquiryEntity.Student.EthnicitiesList
Option Label	<Select Multiple>
Product	Student
Read-only	false
Required	false
Required Message	
Tab Index	
Tag Template	
Tooltip	
Tooltip Duration	750
Value List	Edit...
Visible	true

Rendered Component



Workflow Argument

Name	Direction	Argument type	Default value
prospectInquiryEntity	In/Out	ProspectInquiryEntity	*Default value not supported*

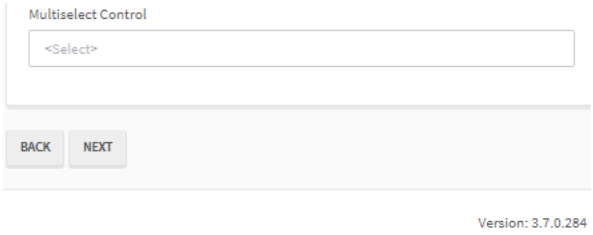
For workflow arguments used with the Calendar/Scheduler in Forms Builder 3.5 and later, refer to Default Argument Types for Components.

Example: Degrees (with Templates)

Control Property Settings

Control Type	Multiselect
Class	
Disabled	false
Filter Type	startswith
Header Template	{{'DEGREES' | translate}}
Id	idd550ba5b-6be9-7cce-83c3-c533a1d5b391
Item Template	#: data.Degree.Id #--#: data.Degree.Name #
Label	Multiselect Control
Lookup Display Member	Degree.Name
Lookup Query	StudentPreviousEducation?$filter=St... eq 51850 and CollegeId gt 0&$select=Id,CollegeId,DegreeId,IsG...
Lookup Sort Member	DegreeId
Lookup Translation Members	
Lookup Value Member	DegreeId
Model	vm.models.CollegeDegrees
Option Label	<Select>
Product	Student
Read-only	false
Required	false
Required Message	
Tab Index	
Tag Template	#: data.Degree.Id #--#: data.Degree.Name #
Tooltip	
Tooltip Duration	750
Value List	Edit...
Visible	true

Rendered Component



- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Disabled** sets a control to disabled.
    - Must be true or false, or a binding beginning with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Filter Type** defines how search values will be filtered. Select from the following filter types: *contains* (default), *endswith*, and *startswith*. Values typed using the keyboard will be used to filter the list according to the selection.
- **Header Template** adds a header to the drop-down list. Specify the header using HTML. If quotes are used in the HTML, they must be single quotes.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **ItemTemplate** controls the layout of the data displayed in the drop-down list. Specify the header using HTML. If quotes are used in the HTML, they must be single quotes.
- **Label** is the value displayed in the label.

- If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
- Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Lookup Display Member** is the name of the property in the OData query string to use for the display.

  For example, if the query string for a list of Ethnicities contains the Code, Name, and ID fields, the Lookup Display Member value can be set to Code, Name, or ID.

- **Lookup Query** is the OData query string to retrieve values for the control.

  The following is an example of an OData query string that retrieves the Code, Name, and ID values from the Ethnicities table, where `isActive` equals true and the returned values are sorted by Name.

  ```
  Ethnicities?$select=Code,Name,Id&$filter=IsActive eq true&$orderby=Name
  ```

  Lookup Query is not used if a custom Value List is specified. Refer to Custom Multiselect with Value List.

- **Lookup Sort Member** is the name of a property in a Lookup query string to sort on. By default, the Lookup query sort order is used.

- **Lookup Translation Members** is a comma separated list of property names in an OData query string to be translated. You should always validate the query will work in a browser. Only basic errors can be detected in Form Designer.

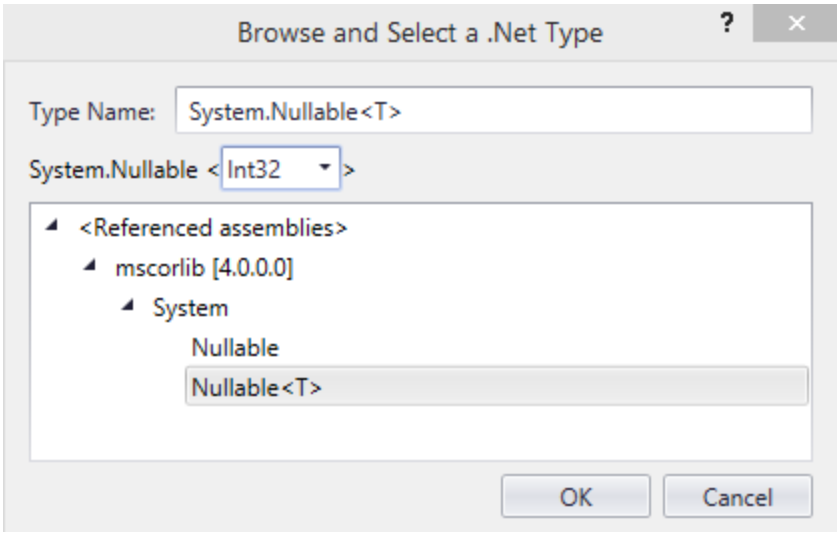
- **Lookup Value Member** is the name of the property in the OData query string to use as the value.
  - If the Lookup Value Member is an Id, the associated data type in Workflow Composer is `Int32`.
  - If the Lookup Value Member is a Code or Name, the associated data type in Workflow Composer is `String`.

- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.
  - The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
  - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
  - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
  - The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
  - If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

  If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

Construction of the model in the workflow is done by assigning data from a provider.
When the Multiselect is used to retrieve integer values and the field is optional (can be empty or null), this must be accounted for when defining the variable for the model binding in the workflow. Instead of defining the variable as an `Int32`, it must be defined as `Nullable<Int32>`. To do so: In the "Browse and Select a .Net Type" window, browse to Type **System.Nullable<T>**and select **Int32** in the System.Nullable field.



- **Option Label** is the label in a drop-down list when no list value is selected. The default value is <Select>.
- **Product** indicates the product from which OData query results are returned. Select from:
    - Student
    - CRM
    - Occupation Insight

  The selected product must be configured in the <products> section of the Renderer web.config file.

  The default Product value will be "Student" if "Student" is selected in the <Select Provider> list on the Fields tab.

  The default Product value will be "CRM" if "CRM" is selected in the <Select Provider> list on the Fields tab.

  Select "Occupation Insight" in the Product property if the source of the query will come from a different data source other than Student/CRM. For more information, refer to Build Queries for Occupation Insight.

  A form can have multiple controls that retrieve data from different providers. For example, a form can have a control that is populated by a query to the Student database. The same form can have another control that retrieves data from Occupation Insight.

Specify the query to retrieve data from the selected provider using the Lookup Query or OData Query property (as applicable for the control). The query contains only the URL specific part of an OData URI. The Base URL and Product will be supplied by the configuration.

- **Read-only** makes the control read-only. It is set to false by default. If you want the component to be read-only, set the property value to true. It is typically used for an input box.
    - Can be bound to a workflow argument or another control's value.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Required** makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.
    - Can be bound to a workflow argument or another control's value.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Required Message** is optional. It overrides the default "Required" message.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
    - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
    - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.

- A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

  The tab index value should not match another control's tab index.

- A blank value (default) will not add a tab index in the HTML.

For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.

- **Tag Template** controls the layout of the data displayed in the selected items of the drop-down list. Specify the header using HTML. If quotes are used in the HTML, they must be single quotes.
- **Tooltip** is the value to display when hovering over the control's label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Value List** is an optional property. Select the **Edit** button to specify the source of the values to be displayed in the Multiselect control.

  For examples of custom Multiselect controls with Value Lists, refer to Custom Multiselect with Value List and Custom Multiselect with Workflow Initialized List.
- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)
  - When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

Lookup Queries for CampusNexus CRM Metadata

For any drop-down or search controls that will be populated via a lookup query, the CampusNexus CRM user needs to enter values for the **Lookup Display Member** and **Lookup Sort Member** attributes. The **Lookup**

Query and **Lookup Value Member** property settings should have default values (if applicable for the selected property) as these are currently specified in the metadata.

Custom Multiselect with Value List

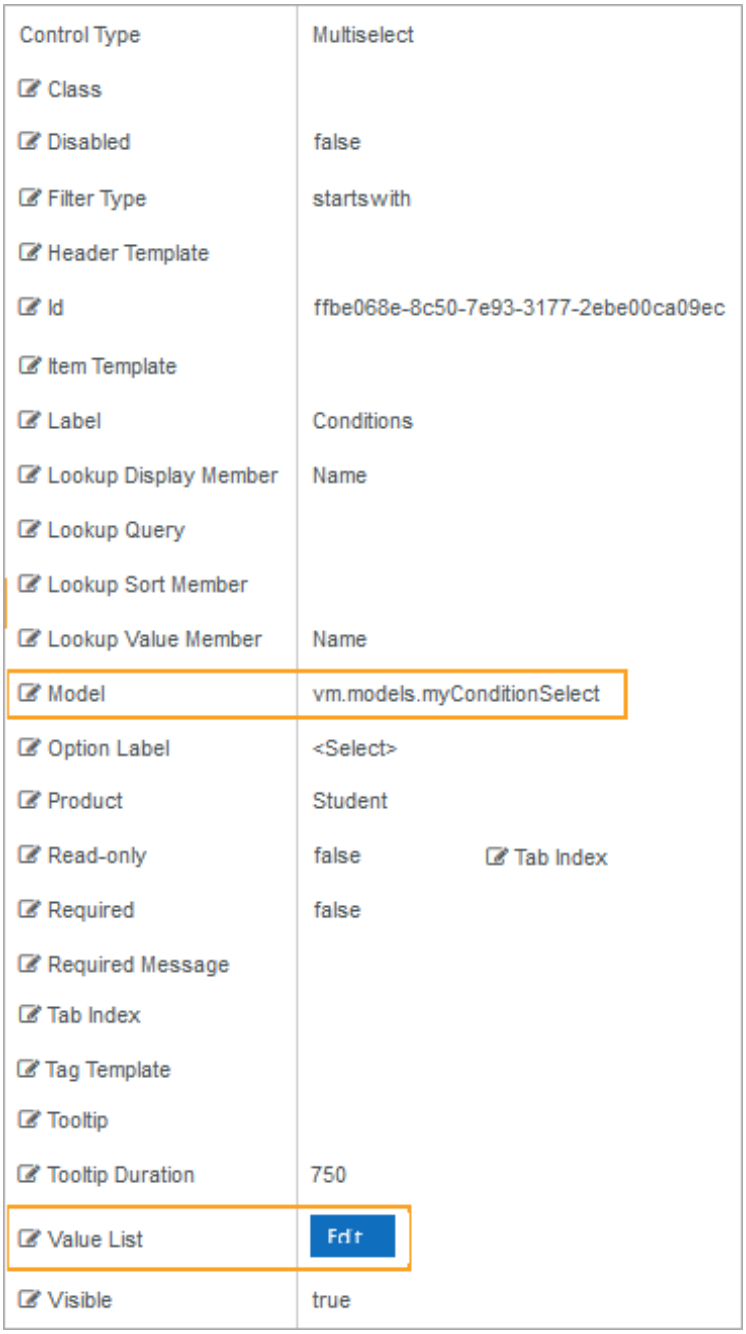
You can use the Multiselect component to create a custom list of values for selection. In our example, the custom list contains values describing certain health conditions.

In a custom list, the Model property needs to define the Model binding for the selected value (e.g., vm.models.myConditionSelect), and the argument type in the workflow needs to be set properly based on the list values (e.g., String). The selection itself will NOT be of type SerializableDynamicObject[].

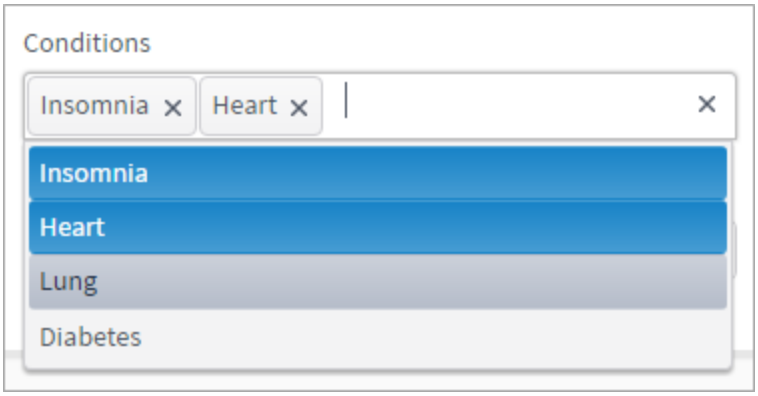
A custom value list applies only to a specific component and cannot be reused for multiple Multiselect components on same page. For example, if "Conditions" is a Multiselect component for multiple sections on same page, each Multiselect component must define the value list with unique bindings.

This topic describes only the Value List property of the Multiselect component. Refer to the Multiselect topic for property settings other than Value List.

Control Property Settings



Rendered Component



Workflow Arguments

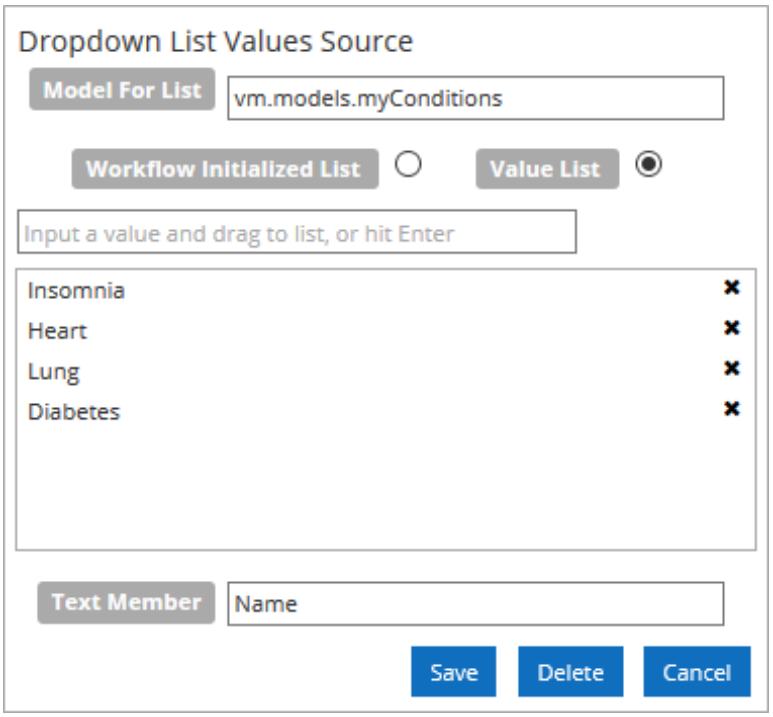
Name	Direction	Argument type	Default value
myConditionSelect	In/Out	String[]	*Default value not supported*

Use an argument of type String[] to capture the selections on the form.

Name	Direction	Argument type	Default value
myConditions	In/Out	SerializableDynamicObject[]	*Default value not supported*

Create a matching argument of type SerializableDynamicObject[] to make the Value List available in a workflow.

Value List is an optional property. Select the **Edit** button to specify the source of the values to be displayed in the Multiselect .



- In the **Model For Value List** field, specify the Model property to which the Multiselect will be bound. The value must start with "vm.models.", for example vm.models.myConditions.
- Select **Value List** to create a custom list. Use the Model property to bind the Value List. The Value List overrides an OData Lookup Query.

  The Value List is available in a workflow if a matching argument of type SerializableDynamicObject[] is created.

  To create the list, type each value in the input field and drag it to the list area. Select ✖ in the list area to delete a value.
- In the **Text Member** field, specify the value that will be used as the DataTextField. This is a required field. In a custom Value List, the Text Member value can be any string, e.g., Name.

Select **Save** to save the values source settings.

Custom Multiselect with Workflow Initialized List

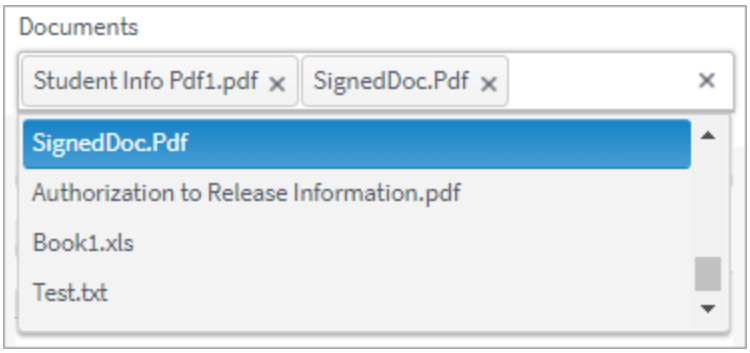
You can use the Multiselect component to create a workflow initialized list of values for selection. Our example contains a list of documents retrieved from the database using a workflow.

This topic describes only the Value List property of the Multiselect component. Refer to the Multiselect topic for property settings other than Value List.

Control Property Settings

Control Type	Multiselect
Class	
Disabled	false
Filter Type	startswith
Header Template	
Id	590c922c-8666-fc78-233e-b358615db21a
Item Template	
Label	Documents
Lookup Display Member	OriginalFileName
Lookup Query	
Lookup Sort Member	
Lookup Value Member	Id
Model	vm.models.myDocument
Option Label	<Select>
Product	Student
Read-only	false
Required	false
Required Message	
Tab Index	
Tag Template	
Tooltip	
Tooltip Duration	750
Value List	Edit
Visible	true

Rendered Component

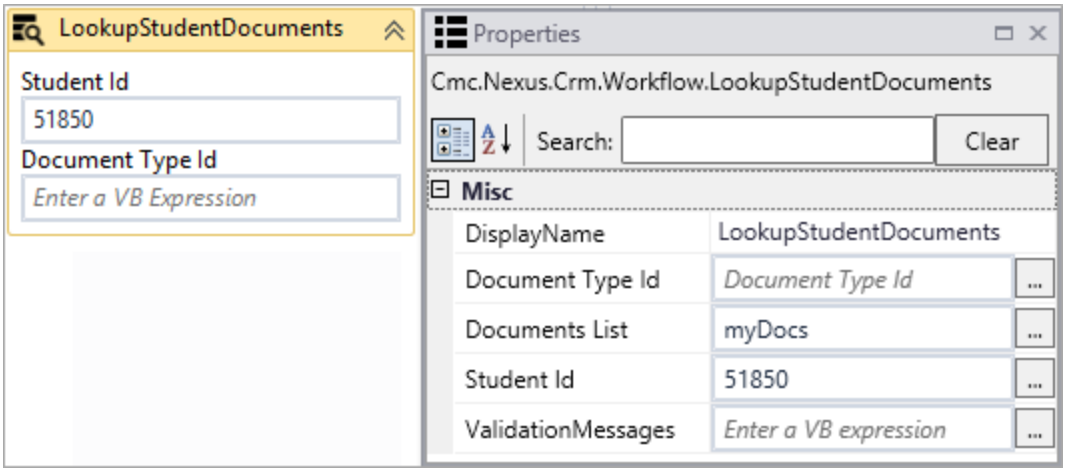


Workflow Arguments

Name	Direction	Argument type	Default value
myDocument	In/Out	String[]	*Default value not supported*
myDocs	In/Out	DocumentEntity[]	*Default value not supported*

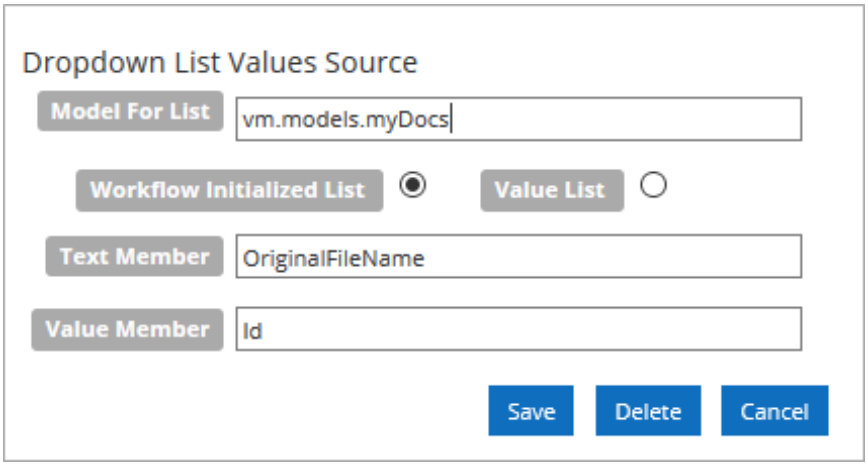
Use an argument of type String[] to capture the selections on the form.
Use another argument to capture the values of the entity Lookup activity.

Workflow Activity



Use a Lookup activity to initialize the Multiselect. The out-argument of the activity holds the entity values. The Lookup activity must be placed on a form that precedes the form with the Multiselect component.

Value List is an optional property. Select the **Edit** button to specify the source of the values to be displayed in the Multiselect control.



- In the **Model For List** field, specify the Model property to which the Multiselect will be bound. The Model For List is required for a Workflow Initialized List. The value must start with "`vm.models.`", for example `vm.models.myConditions`.
- Select **Workflow Initialized List** to bind the values in the list to a Model property value. With this option, the Multiselect values are set in the workflow. The workflow initialized list overrides an OData Lookup Query.

  An easy way to create a workflow initialized list is to use one of the lookup activities (LookupStudentTasks, LookupStudentAdvisors, etc.) if applicable.

  When you are creating a Workflow Initialized List, the simplest object to use is a **NameIdObject**. With an array of these, the Text Member will be Name and the Value Member will be Id, and they will be of type string and integer respectively. If you don't need the Id, it is optional to set it.

  In the workflow, create a variable (myList in this case). DO NOT use an argument or this will not work.

  The type will be NameIdObject[] (array of NameIdObject). You can initialize the object with assign statements, but since variables allow a Default value, use the following example.

  In this example we want to create a list of 2 elements, where Yes is value 1 and No is value 2. Set Default to:

  ```
  new NameIdObject(1){new NameIdObject With { .Name="Yes", .Id=1}, new NameIdObject With { .Name="No", .Id=2}}
  ```

  Note some significant syntax here: the 1 for the array size is VB syntax for an array of 2 elements, with index 0 and 1. There is a dot before each property name in the With sections.

  If you were doing this in assign statements, you could break the statements down as follows:

  ```
  myList = new NameIdObject(1){}  - creates a 2-element array that is empty.
  ```

myList(0) = new NameIdObject - initialize the first array element with a new object, “With” could have been used here instead of the following two assigns.

myList(0).Name = “Yes”

myList(0).Id = 1

etc.

The Default initialization above, while looking more complex, is less wieldy than a few assign statements in the workflow.

To use this, you must expose this as an Out argument of type NameIdObject[]. After you create this argument, you do this with a final assign statement.

```
myArgList = myList
```

The result is that all drop-down list controls that have vm.models.myArgList as the Model For Value List binding (in the popup), will have a Yes/No list. Their Text Member must be Name, and if you use the Value Member, it must be Id.

If any of the following is true, then a SerializableDynamicObject can be used in the same way. It has none of the following limitations.

- You need more than two properties
- The property names cannot be Name and Id
- The types of the property names cannot be string and int respectively.

However, the initialization for the SerializableDynamicObject is considerably more complex to understand to do the same thing as above (with only 2 elements). Here it is:

new SerializableDynamicObject(1){new SerializableDynamicObject With { .DataDictionary = new Dictionary (Of String, Object) From { { "Name", "Yes"}, { "Id", 1} } }, new SerializableDynamicObject With { .DataDictionary = new Dictionary (Of String, Object) From { { "Name", "No"}, { "Id", 2} } } }

You must do this with a variable, and then you must assign it to an argument which is bound to the control.

- In the **Text Member** field, specify the value that will be used as the DataTextField. This is a required field. In a Workflow Initialized List, the Text Member value must match a property in the workflow object collection used to populate it, e.g., `Id`. In a custom Value List, the Text Member value can be any string, e.g., `Name`.
- In the Value Member Field, specify the value returned when item is selected in the Multiselect . It may be the same as Text Member.

Select **Save** to save the data source settings for the list values.

Numeric Text Box

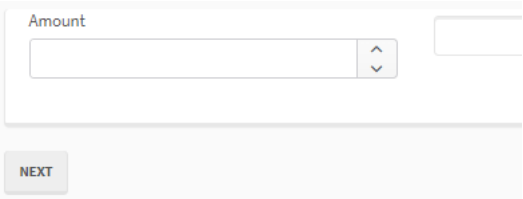
You can use the Numeric Text Box component to capture a single line of numeric information. In contrast to the Text Box component, the Numeric Text Box does not use the Type property. Instead, the Numeric Text Box provides the Decimals, Format, Restrict Decimals, Round, Step, and properties which can be used to qualify or restrict the numeric input.

Properties

Control Property Settings

Control Type	Numeric Text Box
Class	
Decimals	
Disabled	false
Format	c2
Id	id3d47aaf8-ef32-adc0-32fd-63b9a2f3e5c4
Label	Amount
Max Length	50
Maximum Value	5
Minimum Value	5
Model	vm.models.depositEntity
Placeholder	
Read-only	false
Required	false
Required Message	
Restrict Decimals	false
Round	false
Step	1
Tab Index	
Tooltip	Deposit amount
Tooltip Duration	750
Visible	true

Rendered Component



The Numeric Text Box is rendered with a selector that increments or decrements the typed number within the range of the Max Number and Min Number values. For example, if Max Number=10 and Min Number=0, the increments are 1, 2, 3, ...10.

Workflow Argument

Name	Direction	Argument type	Default value
depositEntity	In/Out	DepositEntity	*Default value not supported*

For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Decimals** specifies the number precision. Can be blank, zero, or a positive integer. If blank, the current culture precision is used. If the user enters a number with a greater precision than is currently configured, the value will be rounded. For example, if Decimals is 2 and the user inputs 12.346, the value will become 12.35. If the user inputs 12.99, the value will become 13.00.
    - If this property is bound, it must start with {{vm.models. and end with }}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
- **Disabled** sets a control to disabled.
    - Must be true or false, or a binding beginning with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Format** property settings depend on the control type used:

- **Date Picker**:

  Specifies the format of the date and, when text input is allowed (Disable Input Text = false), the parsing of typed text.

  Examples: yyyy-MM-dd, MM-dd-yy (case sensitive: d for day of month, M for month, y for year)

- **Time Picker**:

  Specifies the format of the time and, when text input is allowed (Disable Input Text = false), the parsing of typed text.

  Examples: HH:mm, hh:mm:ss tt (case sensitive: H or h for hour, m for minute, s for second)

  Refer to Kendo UI documentation for more information.

- **Date Time Picker**:

  Combines the two above with both date and time.

- **Numeric Textbox**:

  Specifies format of the numeric value. e.g., n2 - 2 decimal places, c - currency with cents

> ⚠ The date parsing and formats for these controls changed in Forms Builder 3.5. If you used date formats that are no longer supported, you need to update and re-save affected forms.

- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
  - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
  - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
  - Binding is not supported for this property.
- **Label** is the value displayed in the label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Max Length** is the maximum number of characters in a text box.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessagel}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Maximum Value** specifies the maximum value for a Text Box with `Type=number` or a Numeric Text Box. Leave blank for no maximum.

- **Minimum Value** specifies the minimum value for a Text Box with `Type=number` or a Numeric Text Box. Leave blank for no maximum.
- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.
    - The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
    - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
    - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
    - The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
    - If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

  If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

  If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

  Construction of the model in the workflow is done by assigning data from a provider.
- **Placeholder** is the prompt text displayed in a ghost style in an input box when nothing has been entered.
    - If this property is bound, it must start with {{vm.models. and end with }}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Read-only** makes the control read-only. It is set to false by default. If you want the component to be read-only, set the property value to true. It is typically used for an input box.
    - Can be bound to a workflow argument or another control's value.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)

- **Required** makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.
    - Can be bound to a workflow argument or another control's value.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Required Message** is optional. It overrides the default "Required" message.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Restrict Decimals** specifies whether the decimals length should be restricted during typing. The length of the fraction is defined by the Decimals value. The default is false.
    - If this property is bound, it must start with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Round** specifies whether the value should be rounded on truncated. The length of the fraction is defined by the Decimals value. The default is false.
    - If this property is bound, it must start with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Step** specifies the value used to increment or decrement the widget value. The default is 1.

- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
    - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
    - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
    - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

      The tab index value should not match another control's tab index.
    - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.
- **Tooltip** is the value to display when hovering over the control's label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
    - When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

Popup

You can use the Popup component to add a modal popup window to a form. The user selects a button to open the popup. While the popup is open, a semi-transparent background hides the rest of the page so that the user cannot interact with the page until the popup is closed. The user can select the background or the × button in the upper right corner of the popup to exit the popup.

You can design the layout of the popup using Fields, a subset of the Components, and Form Sections in the same manner as you would design the layout of any other form. Custom styling is also supported.

When you select the **Edit** button in the property settings of a popup, you can insert the following components into the popup:

- Calendar/Scheduler
- Checkbox
- Date Picker
- Date Time Picker
- Drop-down List
- Expand/Collapse Panel
- File Upload
- Grid
- HTML
- Hyperlink
- Label
- Masked Text Box
- Multiselect
- Numeric Text Box
- Radio Button
- Text Box
- Textarea
- Time Picker
- Typeahead

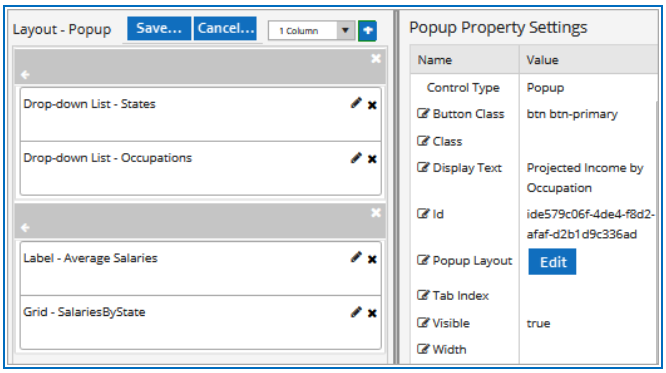
Notes

- Required fields are supported in popups. If required fields are included in a form and/or within popups, users will not be allowed to navigate further unless values are provided in all required fields.
- The contents of a Popup is not displayed in PDF and View Summary, only the popup button is shown.
- On mobile devices, rendering of Popup components has some limitations. For example, the Grid and Calendar components will not be displayed, other components will be reduced in size.

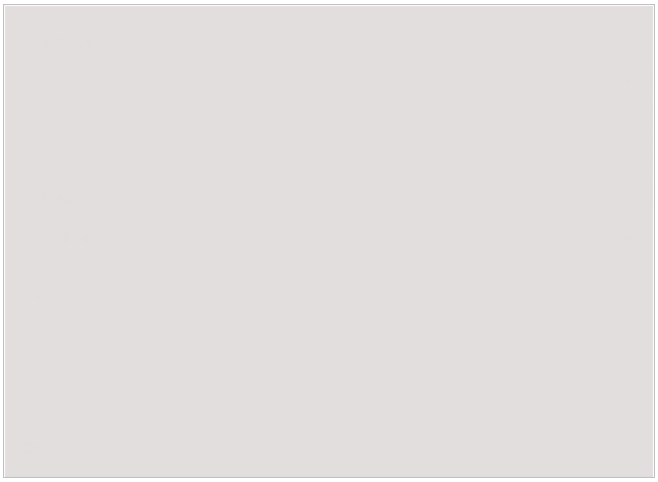
Properties

Control Property Settings

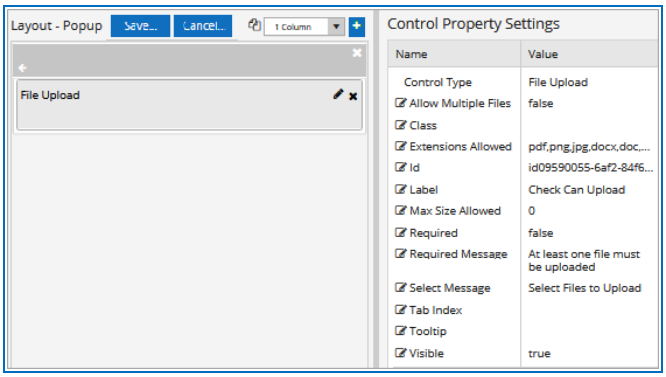
Popup with data retrieved from Occupation Insight



Rendered Component



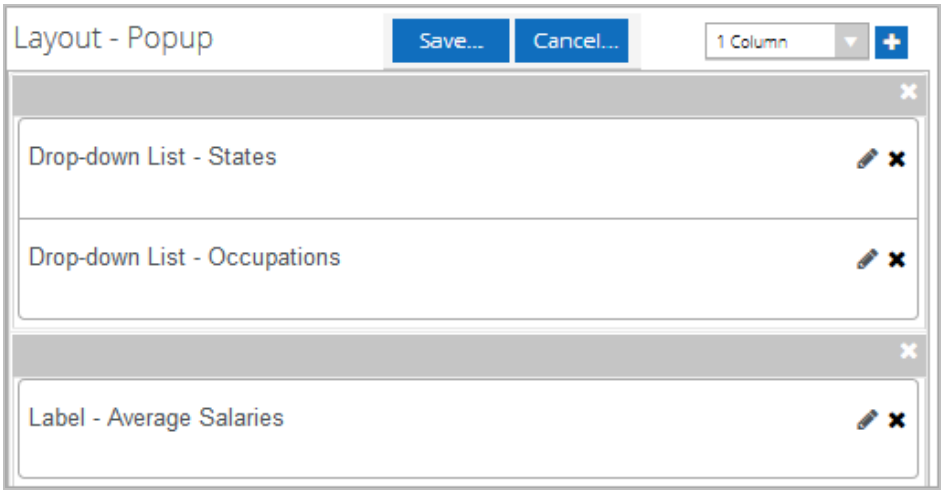
Popup with File Upload component



- **Button Class** is the CSS class for the button. The default is `btn btn-primary`.
  - The class `btn btn-primary` is a Bootstrap class that renders as a blue button with white text.
  - You can customize the CSS Class by selecting a different Bootstrap class or adding your own class in a custom CSS.

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Display Text** is the label of a button.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
  - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
  - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
  - Binding is not supported for this property.
- **Popup Layout** — Select the **Edit** button to access the layout pane for the Popup. The Popup layout pane uses the same space as the Form layout pane but provides separate Save and Cancel buttons while disabling the Forms tools bar.

  The example rendered above has the following layout.



Design the layout of the Popup using Fields, a subset of the Components, and Form Sections in the same manner as you would design the layout of any other form.

Select the **Save** or **Cancel** above the Popup layout pane and then save the Form that contains the Popup.

- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
    - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
    - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
    - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

      The tab index value should not match another control's tab index.
    - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.
- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Width** specifies the width of the popup. If not set, it defaults to a width defined by the popup.
    - Numeric values are treated as pixels.
    - Binding is not supported for this property.

Progress Bar

You can use the Progress Bar component to visualize the progress of an operation such as progress through a form sequence. The component needs to be added to every form within the sequence.

The Visible property can be used to show or hide the Progress Bar on individual forms. The Visible property should be bound to the workflow if the same form will be used with and without a Progress Bar.

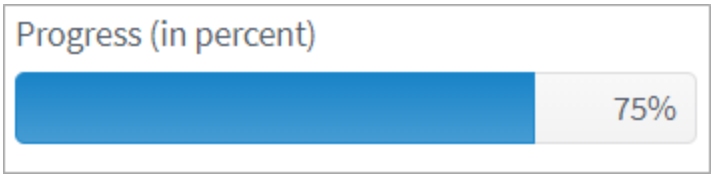
The Current Value property must be dynamically bound and is updated in the workflow as the user progresses though the sequence. If the Progress Bar is added to an end state form, a WaitForFormBookmark activity must be added to update the Current Value on the final step.

Properties

Control Property Settings

Control Type	Progress Bar
Class	
Current Value	{{vm.models.progress}}
Custom Message	
Id	id8907cb0e-fecf-5913-fbf0-beab84a4
Label	Progress (in percent)
Maximum Value	4
Minimum Value	0
Orientation	horizontal
Reverse	false
Segment Count	
Show Value	true
Type	percent
Visible	vm.models.proVisible

Rendered Component

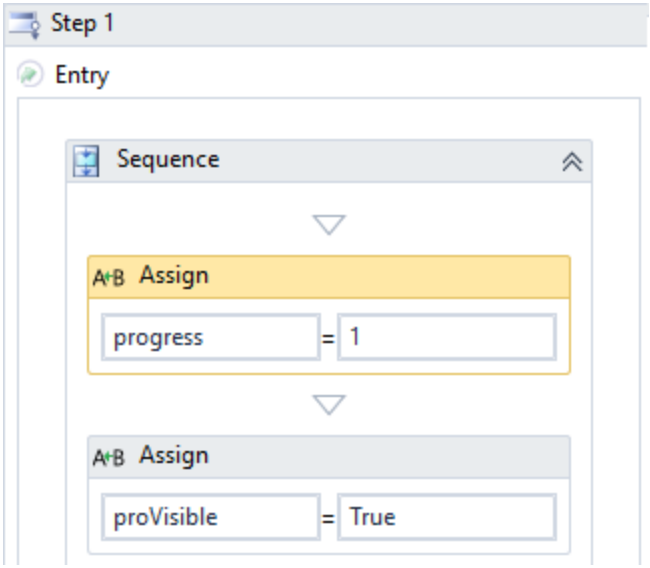


Workflow Argument

Name	Direction	Argument type	Default value
progress	In/Out	Int32	*Default value not supported*
proVisible	In/Out	Boolean	*Default value not supported*

The workflow assigns an integer value to the "progress" argument starting with 1 on the Step 1 form and incrementing the integer for successive forms that display the Progress Bar.

The Visible property is bound to the "proVisible" argument. A value of true is assigned to make the Progress Bar visible on a form.



- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Current Value** specifies the value for the Progress Bar, typically a positive integer. It is usually a bound value from a workflow, e.g., {{vm.models.progress}}. This value can be dynamically changed in the client. To show updated progress through form sequence, this value must be incremented in workflow forward transitions and decremented in workflow backward transitions. This is a required property.
- **Custom Message** is the message that is displayed when the Type property is "custom".
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

  When the Type property is "percent", "segment", or "value", by default, the Label of the Progress Bar displays the Current Value of the progress indicator, e.g., "90%". You can hide the default Label by setting

the Show Value property to false and use the Custom Message property to display a custom message, e.g., "You are almost done - only one more form to complete!"

- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Label** is the value displayed in the label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Maximum Value** specifies the high-end range of the Progress Bar, typically a positive integer indicating the highest value when the Progress Bar is filled. This is a required property.
- **Minimum Value** specifies the low-end range of the Progress Bar. The Minimum Value should be the progress value on the first form, usually 0 or 1.

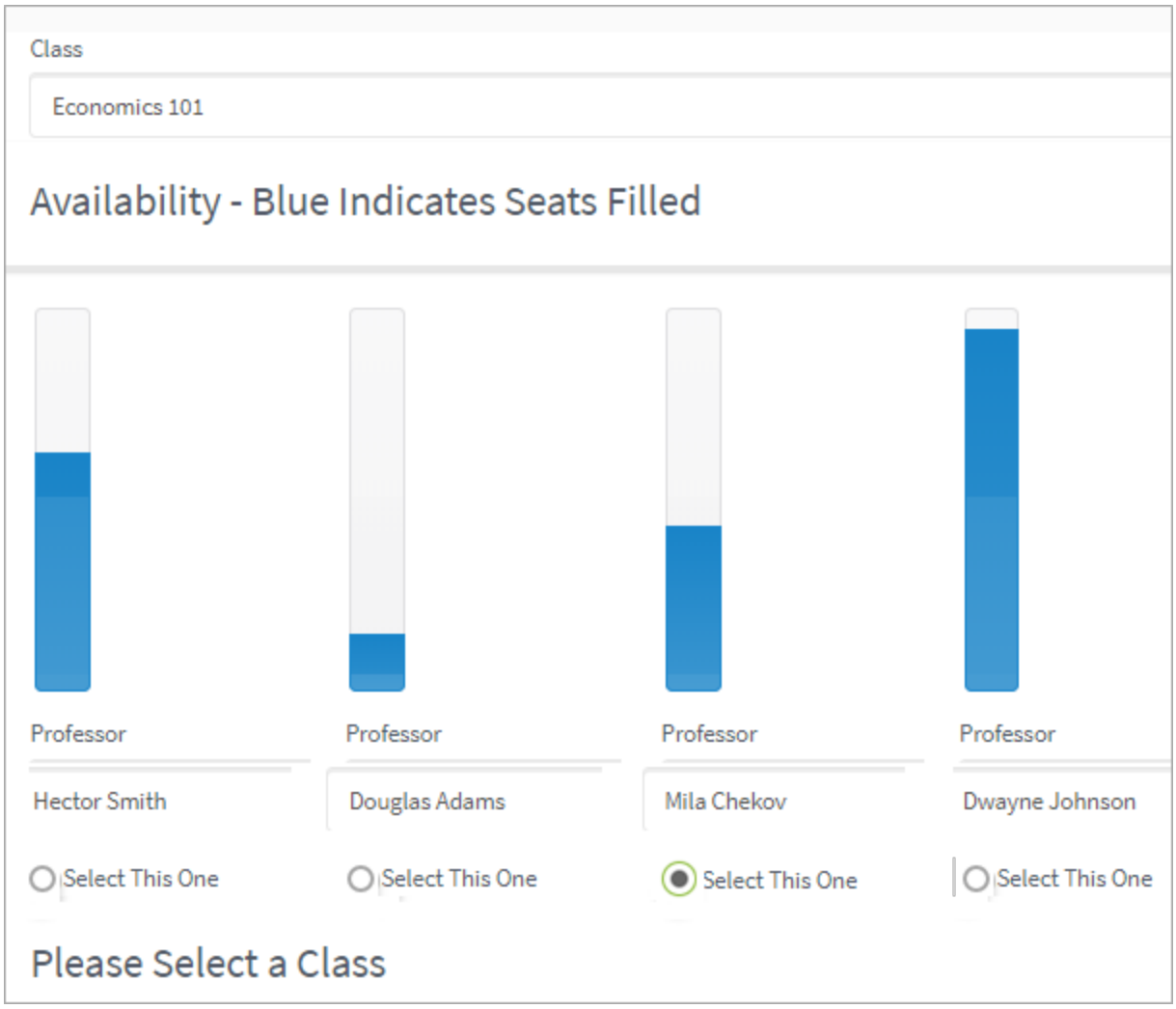
  The Minimum Value and Maximum Value should be the first and last value for the Current Value property. The number of forms in a sequence should be shown (including the end state if a filled Progress Bar is to be displayed there). The Maximum Value should equal the number of forms or number of forms-1 depending on if the first form shows progress or is empty. These values would be different only if the Progress Bar is being used in another way (refer to Use Case: Compare Numeric Data) .
- **Orientation** specifies the orientation of the Progress Bar. Choose from supported orientations.
    - horizontal (default)
    - vertical
- **Reverse** specifies if the progress direction will be reversed. The default is false.
    - This value cannot be changed dynamically.
    - An expression cannot be used.
    - (true and false must be all lowercase)
- **Segment Count**, if the Type property is set to `segment`, defines the number of sections displayed in the Progress Bar. Typically, this will be set to the same value as the Maximum Value (i.e., you have to know the maximum number of steps in the sequence).
- **Show Value** specifies if the progress status (number or percent) will be shown.

- Must be true or false (default), or a binding beginning with "vm.models.".
- This value cannot be changed dynamically, so an expression cannot be used.
- (true and false must be all lowercase)

- **Type** specifies the measurement unit type of the Progress Bar. Choose from supported types:
  - `custom` display based on Custom Message property
  - `percent` display based on percentage
  - `segment` display based on segments as set by the Segment Count
  - `value` (default) display based on numeric value
- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)

Use Case: Compare Numeric Data

The Progress Bar component can also be used as a bar graph to compare numeric data. The example below shows a form that displays the seats taken in different classes. When the user selects a seat in a class, the workflow increments the number of seats taken and the bar graph displays the current value.

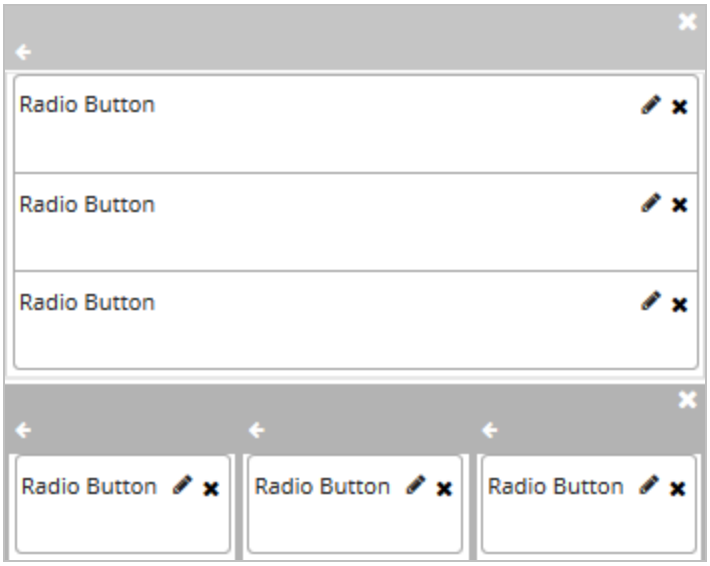

Class
Economics 101
Availability - Blue Indicates Seats Filled
Professor
Hector Smith
Select This One
Professor
Douglas Adams
Select This One
Professor
Mila Chekov
Select This One
Professor
Dwayne Johnson
Select This One
Please Select a Class

Radio Button

You can use the Radio Button component to represent a set of options, only one of which can be selected at any time. The selected value will be accessible within a workflow definition.

The mutually exclusive options must be associated with a **Group**. For example, a Group could be labeled "Priority" and could contain three options labeled "Low", Medium", and "High".

For each option, drag a Radio Button component into the Layout pane. Arrange the components horizontally or vertically. Specify the property settings for each control.

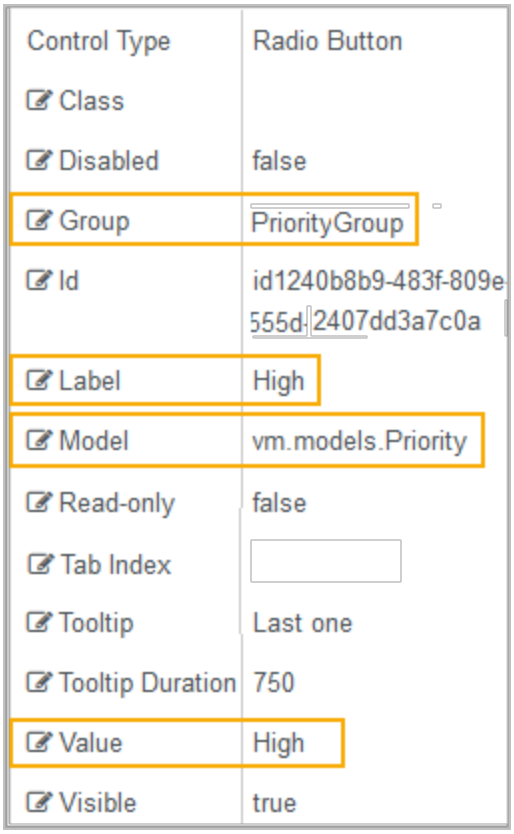


Properties

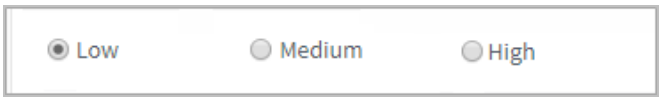
Control Property Settings

Control Type	Radio Button
Class	
Disabled	false
Group	PriorityGroup
Id	61e8af77-1960-9e1c-fb47-524dd0e2b0e1
Label	Low
Model	vm.models.Priority
Read-only	false
Tab Index	
Tooltip	
Tooltip Duration	'50
Value	Low
Visible	true

Control Type	Radio Button
Class	
Disabled	false
Group	PriorityGroup
Id	idd2eb25c1-5f58-d515-69df-5dcc6f52d815
Label	Medium
Model	vm.models.Priority
Read-only	false
Tab Index	
Tooltip	
Tooltip Duration	750
Value	Medium
Visible	true



Rendered Components



Workflow Argument

Name	Direction	Argument type	Default value
Priority	In/Out	String	*Default value not supported*

For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Disabled** sets a control to disabled.
    - Must be true or false, or a binding beginning with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties['yyyyy'].

- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
- If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

- **Group** is the name for a group of radio controls. All radio buttons that work together must have the same group name. Use a unique argument name.

  As a best practice, place grouped radio buttons in their own individual form sections. Additionally, it's recommended to copy the text of this field into the **Title** field of the form section. The copied value will then be:

  - Displayed at the top of the section when the form is rendered.
  - Read by a screen reader application that describes the section.

- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
  - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
  - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
  - Binding is not supported for this property.
- **Label** is the value displayed in the label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.
  - The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
  - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
  - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
  - The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
  - If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

Construction of the model in the workflow is done by assigning data from a provider.

- **Read-only** makes the control read-only. It is set to false by default. If you want the component to be read-only, set the property value to true. It is typically used for an input box.
    - Can be bound to a workflow argument or another control's value.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
    - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
    - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
    - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

      The tab index value should not match another control's tab index.
    - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.

  > The tab index property applies to the radio button group. If a different tab index is specified for each radio button in a group, the tab index of the first radio button will be used for the group.

- **Tooltip** is the value to display when hovering over the control's label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.

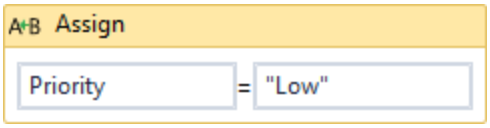
- If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
- Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Value** of the individual radio control in the group. This must be a string value, which may be a number. If the value needs to be a number, it must be converted to one in a workflow.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
  - Values for each control in the group should be different.

- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)

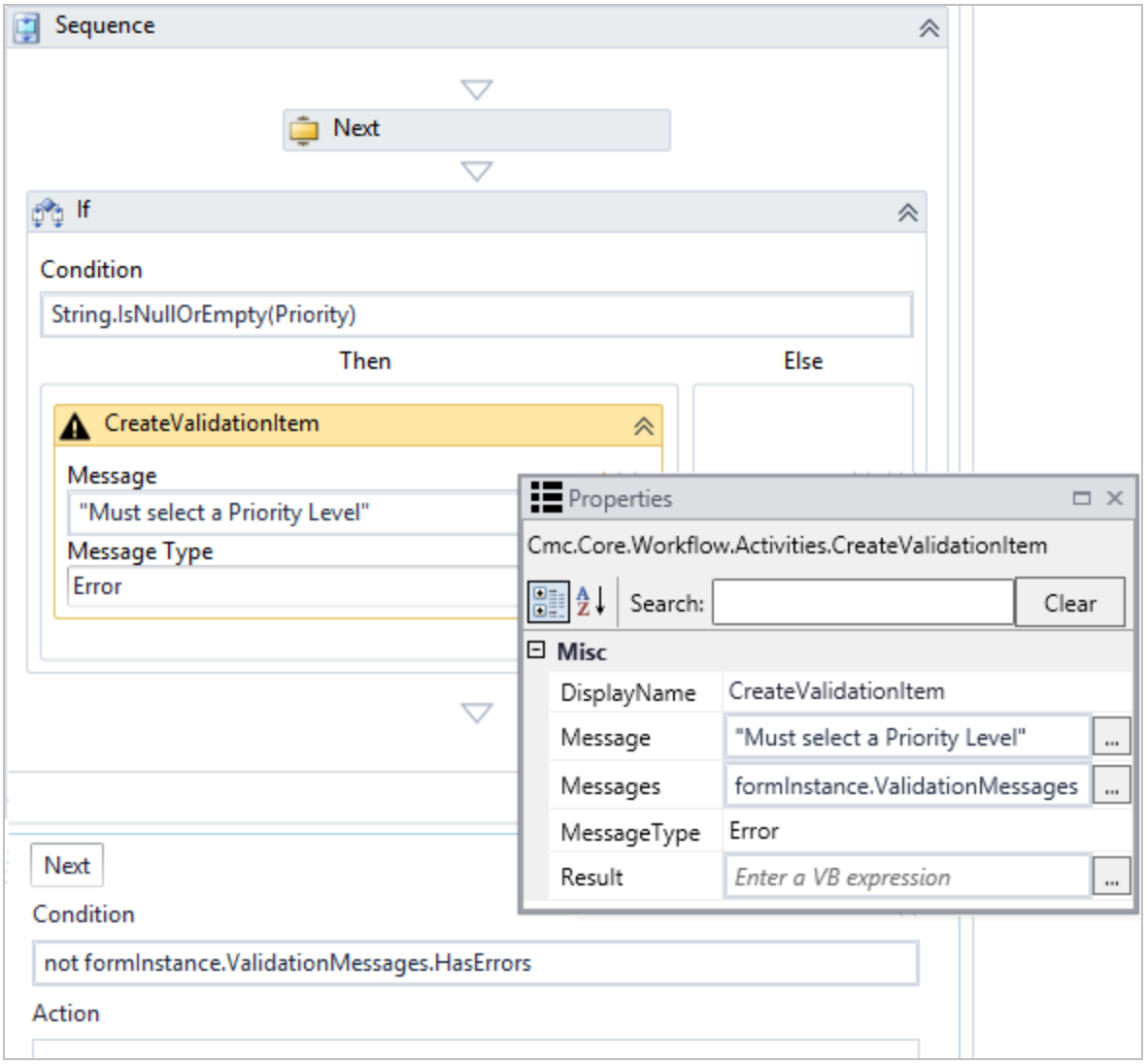
Specify a Default Selection

The Radio Button component does not set a default selection for the Group. If you want to preselect an item in the Group, use an **Assign** activity in the workflow to initialize the Group. Place the Assign activity in the workflow State associated with the form with the Radio Button components.



Create a Validation Item

The Radio Button component does not provide a Required setting for the Group. If you want to ensure that the user selects an option in the Radio Button Group, add a **CreateValidationItem** activity to the workflow. Place the activity on the Next transition that follows the form with the Radio Button components.


Sequence
Next
If
Condition
String.IsNullOrEmpty(Priority)
Then
Else
CreateValidationItem
Message
"Must select a Priority Level"
Message Type
Error
Properties
Cmc.Core.Workflow.Activities.CreateValidationItem
Search:
Clear
Misc
DisplayName
CreateValidationItem
Message
"Must select a Priority Level"
Messages
formInstance.ValidationMessages
MessageType
Error
Result
Enter a VB expression
Next
Condition
not formInstance.ValidationMessages.HasErrors
Action

Repeater

You can use the Repeater component to create a collection of form sections that will be repeated or deleted when the user selects the Add or Delete button. When the collection is repeated, it will show the fields (blank or initialized) to be filled in while retaining the input of the previously completed section. When the Delete button is selected, the collection added last is removed.

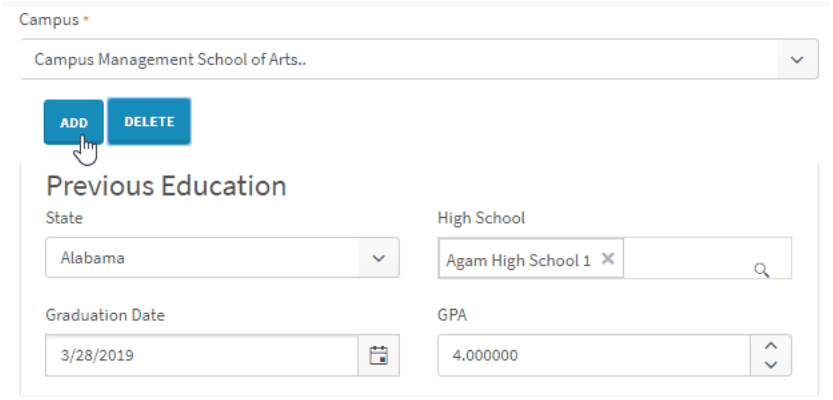
The Repeater is basically a tabular component comprised of rows of components like Text Boxes, Drop-down Lists, etc. Each one of these components has all the properties that they have as individual components.

Properties

Control Property Settings

Name	Value
Control Type	Repeater
Add Button Text	Add
Add Button Visible	true
Button Class	btn btn-primary
Class	
Delete Button Text	Delete
Delete Button Visible	true
Display First Row	false
Id	idc1fb6002-3b96-620c-6d7a-a5f64773abfe
Include Form Sections	☐
Model	vm.models.myPrevEd
Repeater Layout	Edit...
Tab Index	
Visible	true

Rendered Component



- **Add Button Text** is the value displayed in a link or button.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
  - Mixed text and bindings are allowed.
- **Add Button Visible** determines if the button is visible.
  - Must be true or false, or a binding beginning with "vm.models.".
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<,!-!=,==,>=,<=).
  - If comparing to a string, it must be in single quotes.
  - (true or false must be all lowercase)
- **Button Class** is the CSS class for the button. The default is `btn btn-primary`.
  - The class `btn btn-primary` is a Bootstrap class that renders as a blue button with white text.
  - You can customize the CSS Class by selecting a different Bootstrap class or adding your own class in a custom CSS.
- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.

- **Delete Button Text** is the value displayed in a link or button.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
    - Mixed text and bindings are allowed.
- **Delete Button Visible** determines if the button is visible.
    - Must be true or false, or a binding beginning with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<,!-!=,==,>=,<=).
    - If comparing to a string, it must be in single quotes.
    - (true or false must be all lowercase)
- **Display First Row** causes the first Repeater row to display even if there is no data supplied by a work-flow argument.
    - The first bound array value will be created with empty property values.
    - Delete will not be available for the first row.
    - This is not dynamically bindable (cannot take an expression) and must be set before the form is rendered.
    - If data is defined for the first row, this actually does nothing except prevent the first row from being deleted.
    - (true or false must be all lowercase)
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Include Form Sections** causes each Repeater row to be enclosed in a form section panel.
    - When checked, each repeat is embedded in form section panels corresponding to the number of form sections in the Repeater layout. "Merge With Next" and "Merge With Previous" should be

checked appropriately on each of the layout form sections so that each repeat is in one containing panel.

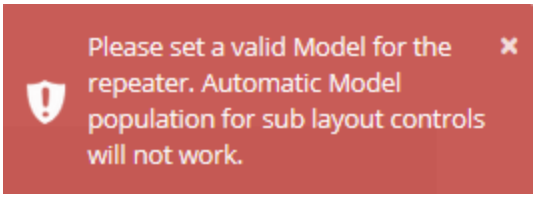
- When unchecked, individual repeats are not in form sections; however, the entire Repeater control is by default, unless it is merged with other form sections.

- **Model** is required for binding to a workflow argument or another control.
  - The model value must always start with "vm.models.", e.g., vm.models.myArgument.
  - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
  - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
  - The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
  - If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

Construction of the model in the workflow is done by assigning data from a provider.

> Models for controls used within the Repeater Layout that are row specific must use this model with a special array row number index, **[$RepeaterRowNumber]**, followed by a dot and the property name. $RepeaterRowNumber is replaced by the actual row number index during render.
>
> An Entity Field property within it can be addressed with **vm.models.myRepeater[$RepeaterRowNumber].myEntityProperty**.
>
> ⚠ For a Component property, the model value is prepopulated with a placeholder **vm.models.myRepeater[$RepeaterRowNumber].<Repeater Property Placeholder>** that needs to be replaced by the user. For example, a Text Box Component in the Repeater can be addressed with **vm.models.myRepeater[$RepeaterRowNumber].myText**
>
> These model values will be available in workflow arguments in exactly the same way as other arguments without the vm.models. prefix.
>
> Note the types of these arguments are defined as **dummy underscore** arguments when the workflow is initially created. If modifications are made later, the workflow arguments can be updated following the same pattern.
>
> For more information, refer to Repeater Workflow Arguments.

If the model is not specified in the Property Settings for the Repeater Control, the following message is displayed when you select the Repeater Layout **Edit** button.



- **Repeater Layout** — Select the **Edit** button to specify the layout of the Repeater component. Refer to Example: Create a Repeater below.
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
    - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
    - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
    - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

      The tab index value should not match another control's tab index.
    - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.
- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)

Example: Create a Repeater

1. Drag the **Repeater Component** into the Layout pane for the form.
2. Specify the **Control Property Settings** for the Repeater. Refer to Properties above.

   In our example, we specified **vm.models.myPrevEd** and accepted the defaults for the remaining properties.
3. Select the **Edit** button for the Repeater Layout.

   The Repeater Layout pane will replace the Form Layout pane, and the Components tab will show a subset of components that are available for use in a Repeater.
4. Select the number of **Columns** for the Repeater section and select the button.

5. Drag **Fields** and/or **Components** into the Repeater Layout pane and specify the properties for the selected controls.

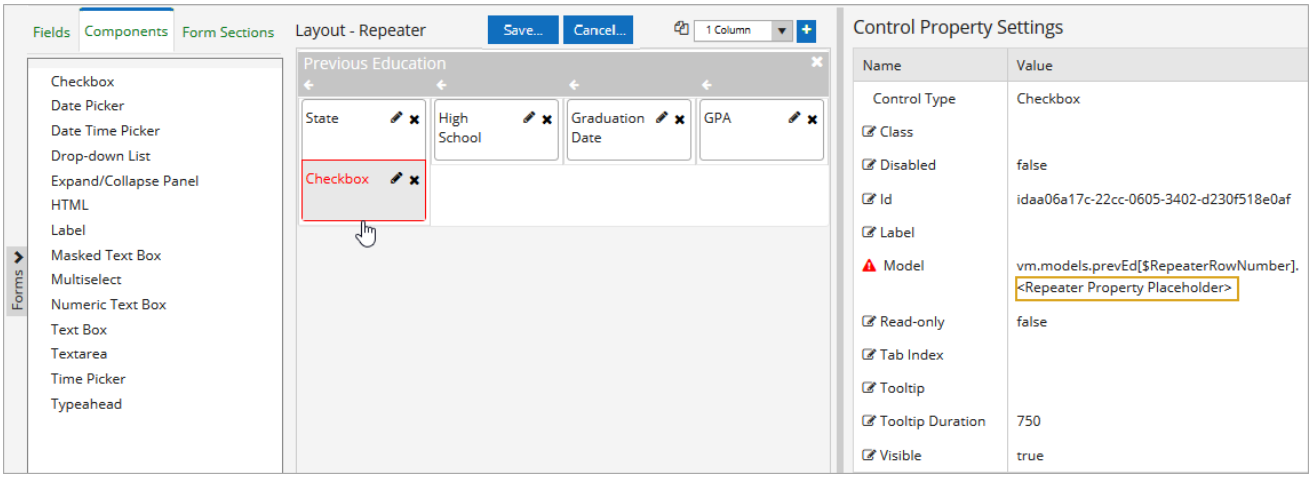
   Bound values in the Repeater are addressed with the model value and a row index placeholder which is substituted when the form is rendered. For more information, refer to Binding Values in a Repeater Layout.

   In our example, the Repeater has 4 bound entity **Fields** that are assigned the following model values in the Repeater Layout pane:

Example: Model Values for Entity Fields in the Repeater Layout Pane

Field	Control Type	Model Value
State	Drop-down List	vm.models.myPrevEd[$RepeaterRowNumber].State
High School	Single-select Search	vm.models.myPrevEd[$RepeaterRowNumber].HighSchoolId
Graduation Date	Date Picker	vm.models.myPrevEd[$RepeaterRowNumber].Gpa
GPA	Numeric Text Box	vm.models.myPrevEd[$RepeaterRowNumber].GraduationDate

When you drag a **Component** into the Repeater Layout pane, the model value in the Control Property Settings is populated with a **<Repeater Property Placeholder>** as shown below for a Checkbox. You must replace the placeholder with an appropriate value, e.g., "myCheckbox".

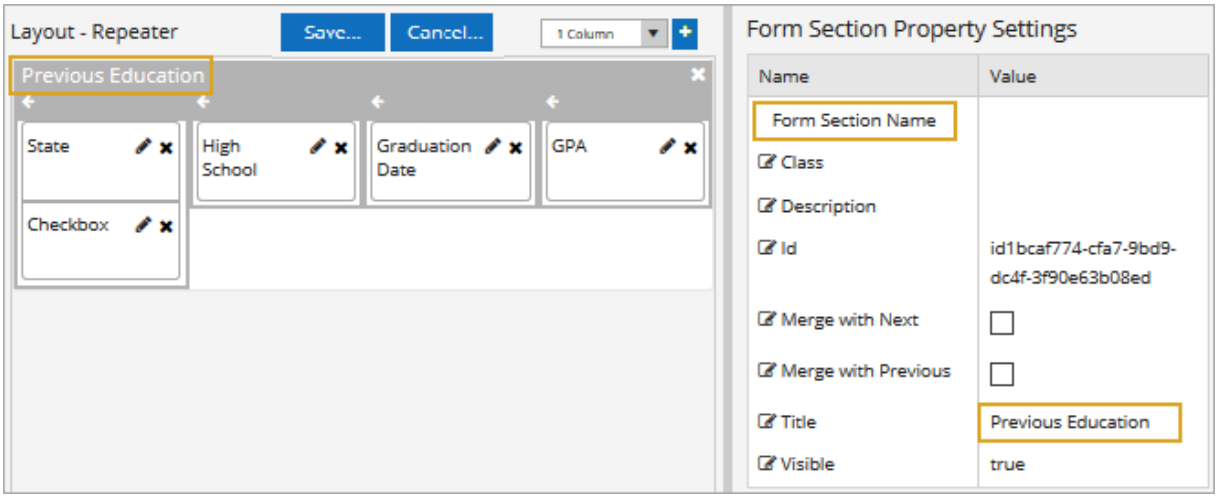


Example: Model Value for a Component in the Repeater Layout Pane

	Model Value
Prepopulated value:	vm.models.myPrevEd[$RepeaterRowNumber].**<Repeater Property Placeholder>**
Edited/saved value:	vm.models.myPrevEd[$RepeaterRowNumber].**myCheckbox**

If you do not replace the placeholder, the following message will be displayed when you try to save the form: "*Errors were detected in some controls, some form sections or the form. This may mean your form will either not render as expected or may malfunction completely. Do you still want to save?*"

6. Select the section bar of the Repeater in the Layout pane and specify the **Form Section Property Settings**. In our example, we specified the Title "Previous Education". You can style the Repeater like any other form section. For more information, refer to Form Sections.



If you had two form sections in a layout, the first one could have "Merge with Next" checked, and the 2nd one would have "Merge with Previous" checked. In that case, the two form sections would have minimal space and no border between them. These properties work in pairs with one form section merging into the next.

7. Select the **Save** button above the Repeater Layout pane to save your design (or select **Cancel** to discard your changes). You will be returned to the form layout.
8. Add the form with the Repeater component to a sequence and test the rendered sequence.
9. Locate the workflow for the sequence in Workflow Composer and review the initial arguments for the sequence.

   The **myPrevEd** argument of type **RepeaterSerializableDynamicObject[]** was created based on the model value in the Repeater Control Property Settings.

   The dummy arguments **_myPrevEd_studentEntity** and **_myPrevEd_studentPreviousEducationEntity** of type StudentEntity and StudentPrevious EducationEntity were created.

   The dummy arguments **_myCheckbox**, **_myPrevEd_State**, **_myPrevEd_Gpa**, etc. were created for the model values in the Repeater controls.

   For more information, refer to Repeater Workflow Arguments.

Name	Direction	Argument type
formInstance	In/Out	FormInstance
entity	In/Out	VoidEntity
event	In/Out	ConstructedEvent
_myPrevEd_studentEntity	In/Out	StudentEntity
_myPrevEd_studentPreviousEducationEntity	In/Out	StudentPreviousEducationEntity
_myPrevEd_Gpa	In/Out	Nullable<Decimal>
_myPrevEd_GraduationDate	In/Out	Nullable<DateTime>
_myPrevEd_HighSchoolId	In/Out	Nullable<Int32>
_myPrevEd_State	In/Out	String
myPrevEd	In/Out	RepeaterSerializableDynamicObject[]

Binding Values in a Repeater Layout

Bound values in the Repeater are addressed with the Repeater model and a row index placeholder which is substituted when the sequence is rendered.

Example

The model for the Repeater Control Property Settings in the is **vm.models.myRepeater**.

Any binding in the layout which needs to be row-relative (i.e., row-specific) needs to provide an index to the property.

For example, Renderer addresses a Text Box control in the Repeater Layout as:

```
vm.models.myRepeater[$RepeaterRowNumber].myText
```

$RepeaterRowNumber should be treated as case sensitive.

If a component property is not row-relative, that is to say it has the same value on every Repeater row, then it can just be a regular binding like vm.models.myLabel.

The Debug JSON will correspond to the path. In the example above, it would look like this:

```
"myRepeater": [
  {
  "myText": "This is text in row 1"
  },
  {
  "myText": "This is text in row 2"
  }
]
```

This JSON shows “myRepeater” is an array of two objects, with each object having one property with a value. Please refer to https://www.w3schools.com/js/js_json_syntax.asp for more information on JSON syntax.

Repeater Workflow Arguments

When a sequence is saved for the first time, workflow arguments are created for a Repeater. The Repeater model is created as a **RepeaterSerializableDynamicObject[]**, an array which works exactly like a SerializableDynamicObject[] with a DataDictionary to reference its values.

In addition, each of the model properties within the Repeater Layout also gets a workflow argument. However, these are **dummy arguments**. They are not used in either a form or the workflow and only tell Forms Builder the type of each Repeater Layout argument. They are used during deserialization of the JSON Entities string to workflow arguments and tell the deserializer what type to use.

The format of the dummy workflow arguments is:

`_<repeater model name>_<control model name>`=> Direction: In/Out => Argument type of control model

In the "myRepeater" Text Box the additional argument created would be:

`_myRepeater_myText`of type String

For an Entity Field such as "State" from the Student Entity bound to a model "vm.models.myRepeater[$RepeaterRowNumber].State", the dummy argument would be:

`_myRepeater_State` of type String

For Entity Fields you can also use the ObjectCopier<> activity to copy one object to another object without having to iterate through the object properties.

SignNow

You can use the SignNow component to integrate SignNow fields into a form. Forms Builder supports several types of SignNow fields, e.g., Checkbox, Dropdown, Email, Initial, Signature, and Text. The fields are selected in the **Type** property of the Control Property Settings.

Before using the SignNow component, you must set up and account with SignNow and configure the SignNow Settings in Forms Builder.

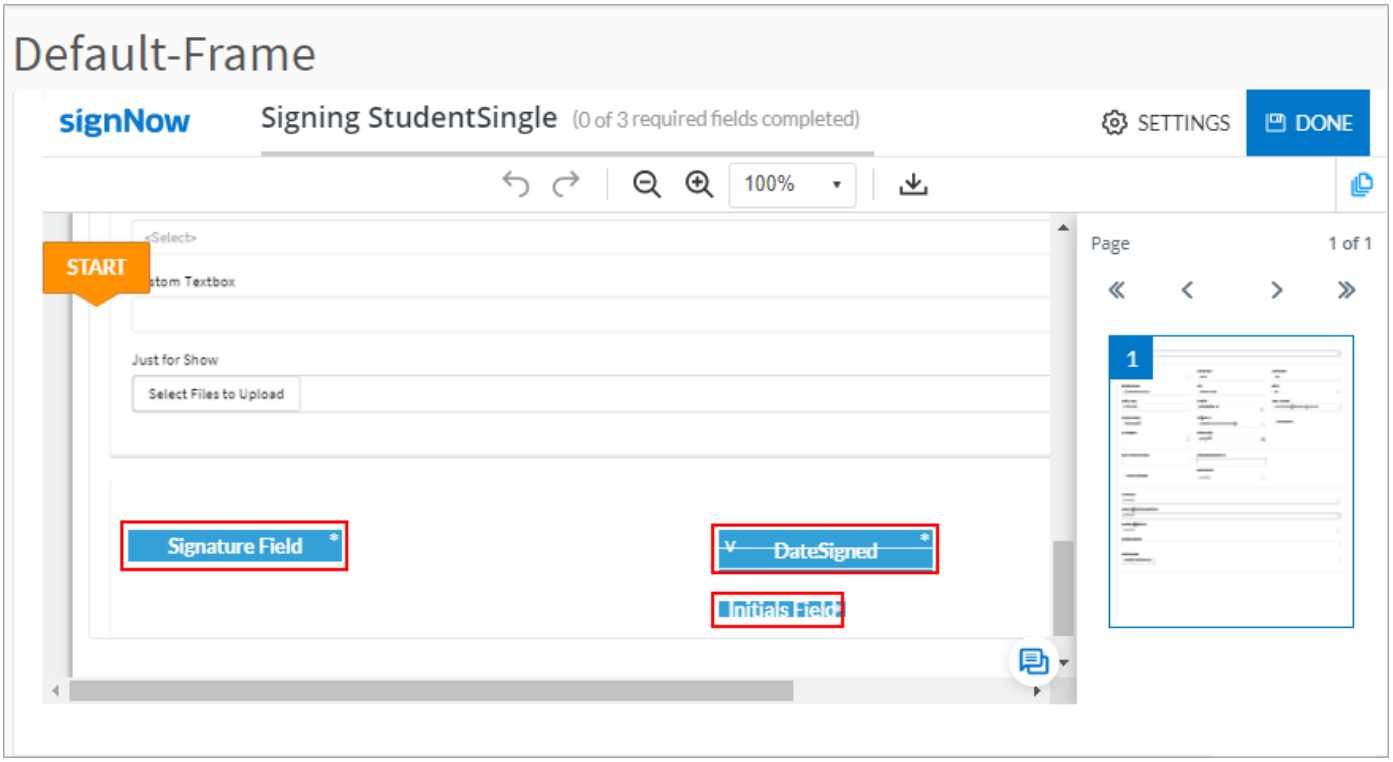
Properties

Control Property Settings

Control Type	SignNow
Class	
Height	
Options	
Required	☑
Signer	Self - Signer 1
Signer Tooltip	
Type	Signature
Visible	true
Width	

Rendered Components

- Type=Signature
- Type=Date Signed
- Type=Initial



- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Height** of the field. Defaults to 15 px if not specified.
- **Options** for a Dropdown field. Specify the values as a comma separated list with a space after each comma e.g., `option1, option2, option3, ...`

  If the spaces are missing, the Dropdown field is not displayed.
- **Required** is set to false by default. Select the property if input is required for a field. This property is applicable only to Checkbox, Dropdown, Email, Phone, SSN, and Text fields.
- **Signer** provides the following options:
  - Self - Signer 1
  - Signer 2
  - Signer 3
  - Signer 4
  - Signer 5
- **Signer Tooltip** is a label on a Type field that displays at signing for the specified signer. A Signer Tooltip can be applied to any Type field, except Checkbox.

  If no Signer Tooltip is applied to a Type field, a default for the Type will be displayed.

  Spaces in Signer Tooltips are stripped out. If special characters are used, the field will not be displayed.

Example:

When a Signer Tooltip is specified as "Mom Email", the Email field will be displayed as: 

.

Without the Signer Tooltip, the Email field will be displayed as: 

Signer Tooltip is displayed only to the specified signer at the time of signing and will not be shown in the final pdf. To include a label for the Type field that will be displayed in the final pdf, use an HTML or Label component.

- **Type** provides the following SignNow fields:

Field	Description
Checkbox	Allows the recipient to select or clear a check box.
Date Signed	Allows the recipient to specify the date the document was signed.
Dropdown	Allows the recipient to select one of the available values. The values are hard-coded as a comma-separated listed in the Options field.
Email	Allows the recipient to enter text that is validated for proper email format.
Initial	Allows the recipient to initial the document.
Phone	Allows the recipient to enter text that is validated for proper phone number format.
Signature	Allows the recipient to sign a document.
SSN	Allows the recipient to enter text that is validated for proper social security number format.
Text	Allows the recipient to enter any text.

- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Width** of the field. Defaults to 100 px if not specified.

Working with the SignNow Component

You are designing a form in which you want the person completing the form to sign and date the form electronically.

1. Drag two SignNow components into the Layout pane of your form in Form Designer.
2. On the first SignNow component, set the Signer to **Self - Signer 1** and set the Type to **Signature**.
3. On the second SignNow component, set the Signer to **Self - Signer 1** and set the Type to**Date Signed**.

When Anthology Student fields are inserted into a form, the student is automatically directed to the section that needs an e-signature.

If the fields are not inserted into a form but e-signature is enabled, the student is directed to electronically sign the document and can drag out the signature anywhere on the document.

Single-select Search

You can use the Single-select Search component to enable users to find and select a single item from a list. When the user selects the component in a form, the Search Name dialog is displayed. The dialog allows the user to search for a name, filter the list, or browse through the list pages, and select one item.

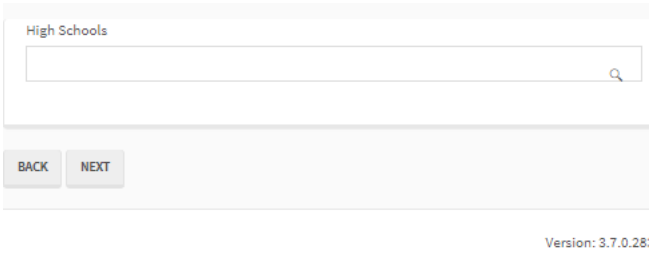
The CampusNexus data model specifies the DisplayControlType of "SearchControl" for many properties. When the Single-select Search component is dropped onto the Layout pane in Form Designer, the Single-select Search component will be the default control type for properties that have the DisplayControlType of "SearchControl" in the metadata.

Properties

Control Property Settings

Control Type	Single-select Search
Class	
Disabled	false
Filter Type	contains
Grid Columns	[{"field": "Name", "title": "Name"}]
Id	id5a20f1da-7b5d-5ca0-3184-e4daff4d0ee0
Label	High Schools
Lookup Display Member	Name
Lookup Query	HighSchools?$select=Code,Name,Id&$orderby=Name
Lookup Translation Members	
Lookup Value Member	Id
Model	vm.models.mySearch
Placeholder	
Product	Student
Required	false
Required Message	
Tab Index	
Tooltip	
Tooltip Duration	750
Visible	true

Rendered Component



Workflow Argument

Name	Direction	Argument type	Default value
mySearch	In/Out	Int32[]	*Default value not supported*

For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

> When using OData queries in the properties of the Single-select Search component, do not mix dotted notation to access members from an $expand and non-dotted notation to access member from the main $select.

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Disabled** sets a control to disabled.
  - Must be true or false, or a binding beginning with "vm.models.".
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
  - If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)

- **Filter Type** defines how search values will be filtered. Select from the following filter types: *contains* (default), *endswith*, and *startswith*. Values typed using the keyboard will be used to filter the list according to the selection.
- **Grid Columns** is set to an array of JSON objects. The "field" value is mandatory for each object. Use this property to specify the columns displayed in the search results. For more details, refer to Adjust the Search Grid Columns Property.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Label** is the value displayed in the label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Lookup Display Member** is the name of the property in the OData query string to use for the display.

  Also refer to Adjust the Search Grid Columns Property.
- **Lookup Query** is the OData query string to retrieve values for the control.

  Example

  HighSchools
- **Lookup Translation Members** is a comma separated list of property names in an OData query string to be translated. You should always validate the query will work in a browser. Only basic errors can be detected in Form Designer.
- **Lookup Value Member** is the name of the property in the OData query string to use as the value.
    - If the Lookup Value Member is an Id, the associated data type in Workflow Composer is `Int32`.
    - If the Lookup Value Member is a Code or Name, the associated data type in Workflow Composer is `String`.
- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.

- The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
- This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
- Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
- The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
- If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

Construction of the model in the workflow is done by assigning data from a provider.

The argument type for list items in Workflow Composer will be an integer array: `System.Int32[]`

- **Placeholder** is the prompt text displayed in a ghost style in an input box when nothing has been entered.
  - If this property is bound, it must start with {{vm.models. and end with }}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Product** indicates the product from which OData query results are returned. Select from:
  - Student
  - CRM
  - Occupation Insight

The selected product must be configured in the <products> section of the Renderer web.config file.

The default Product value will be "Student" if "Student" is selected in the <Select Provider> list on the Fields tab.

The default Product value will be "CRM" if "CRM" is selected in the <Select Provider> list on the Fields tab.

Select "Occupation Insight" in the Product property if the source of the query will come from a different data source other than Student/CRM. For more information, refer to Build Queries for Occupation Insight.

A form can have multiple controls that retrieve data from different providers. For example, a form can have a control that is populated by a query to the Student database. The same form can have another control that retrieves data from Occupation Insight.

Specify the query to retrieve data from the selected provider using the Lookup Query or OData Query property (as applicable for the control). The query contains only the URL specific part of an OData URI. The Base URL and Product will be supplied by the configuration.

- **Required** makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.
    - Can be bound to a workflow argument or another control's value.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Required Message** is optional. It overrides the default "Required" message.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
    - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
    - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
    - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

      The tab index value should not match another control's tab index.
    - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.
- **Tooltip** is the value to display when hovering over the control's label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
    - When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

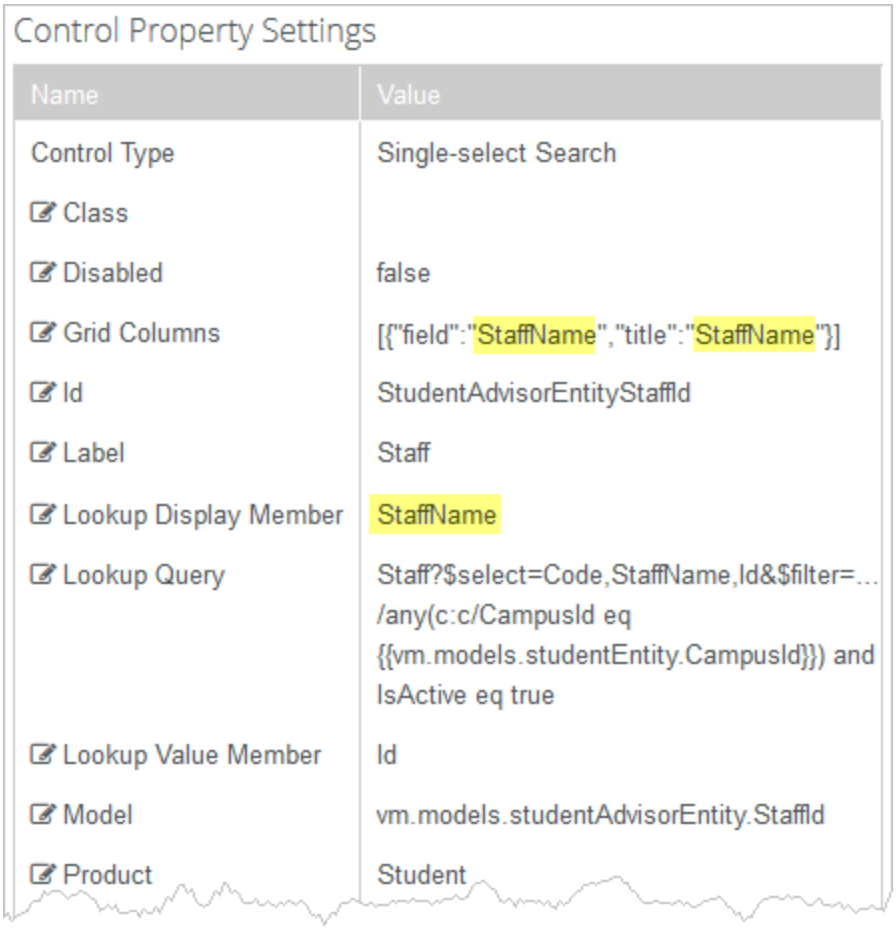
Lookup Queries for CampusNexus CRM Metadata

For any drop-down or search controls that will be populated via a lookup query, the CampusNexus CRM user needs to enter values for the **Lookup Display Member** and **Lookup Sort Member** attributes. The **Lookup Query** and **Lookup Value Member** property settings should have default values (if applicable for the selected property) as these are currently specified in the metadata.

Adjust the Search Grid Columns Property

The **LookupDisplayMember** value is used to populate default value for the Grid Columns property on Single-select Search control types.

For example, when the LookupDisplayMember value is StaffName, the Grid Columns property default is [{"field":"StaffName","title":"StaffName"}].



If no LookupDisplayMember value is present, the default Grid Columns property will be [{"field": "Name", "title": "Name"}] .

To display the Single-select Search control with two grid columns for the StaffName field and Code field, modify the Grid Columns property as follows:
[{"field": "StaffName", "title": "StaffName"},**{"field": "Code", "title": "Code"}]**

TabStrip

The TabStrip component enables you to place a container of tabs on a form or form section. Each tab represents an existing form.

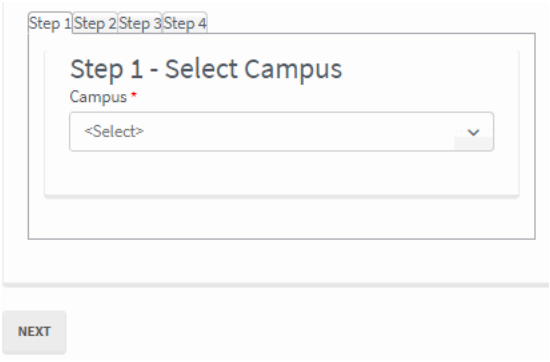
> If you delete a form that is contained in a tab, the form will still be rendered, but attempts to export the sequence will fail. The export error message will indicate the name of the missing form.
>
> If you update a form that is contained in a tab, the tab version will not get the update until the form with the tab reference is also re-saved.

Properties

Control Property Settings

Name	Value
Control Type	Tabstrip
Animation	false
Class	
Collapsible	☐
Id	idb28ee98c-5889-15d4-6ea9-961fed6ba190
Tab Position	top
TabStrip Configuration	Edit...
Visible	true

Rendered Component



- **Animation** - This value must be false or a valid JSON specification (true is not valid).

  Example

```
{
  "open": {
    "effects": "fade:in",
    "duration": 750
  },
  "close": {
    "effects": "fade:out",
    "duration": 1000
  }
}
```

The Animation property is not used when the Position is left or right.

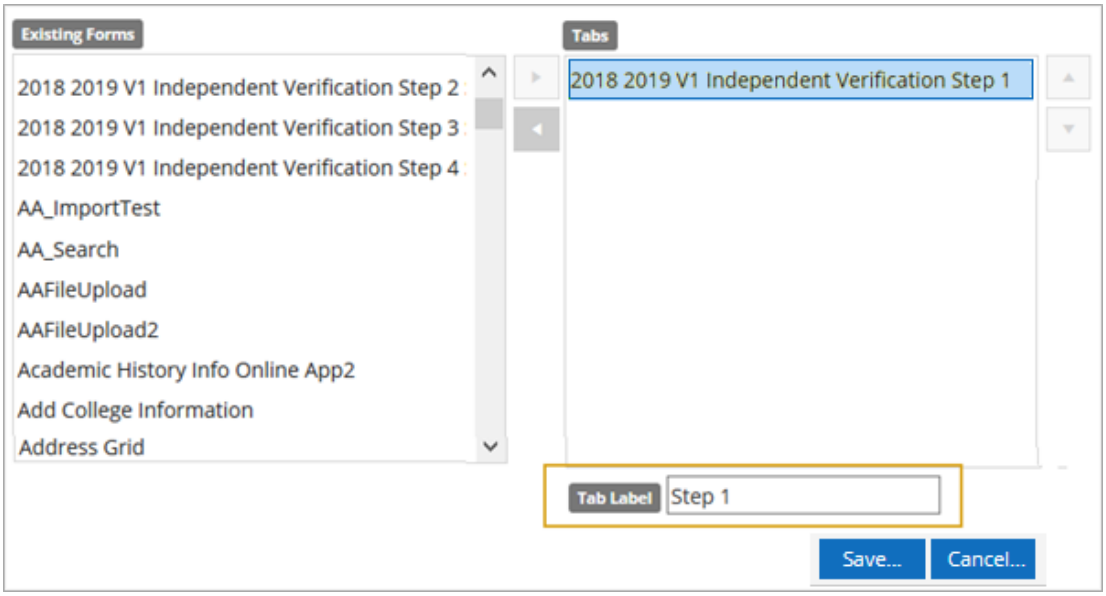
- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Collapsible** indicates whether the tab contents can be toggled between visible (expanded) and non-visible (collapsed). The default is collapsed when selected. This property is not bindable.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Tab Position** is the relative position of the tabs with respect to the content. Select a value from the drop-down list. The options are:
    - bottom
    - left
    - right
    - top

  The property is not bindable.
- **TabStrip Configuration** - Select the Edit button to select the forms to display in the TabStrip and to assign the tab labels. Refer to Creating a TabStrip.
- **Visible** makes the control visible or hidden.

- Can be bound to a workflow argument or another control's value. This property is dynamic.
- A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

Creating a TabStrip

1. Drag the TabStrip component into the Layout pane.
2. Select the **Edit** button in the Control Property Settings pane. The TabStrip Configuration window is displayed.
3. In the **Existing Forms** section, select a form and select the right arrow to move the form to the Tabs pane.
4. In the **Tab Label** field, specify a label for the tab.



5. Repeat steps 3 and 4 for any additional tabs.
   - To reorder tabs, select a form in the Tabs pane and select the up/down arrows.
   - To remove a form, select the form in the Tabs pane and select the left arrow.
6. After you have completed the tab configuration, select **Save**. The TabStrip Configuration window is closed.
7. Specify additional settings for the TabStrip in the **Control Property Settings** pane.
8. **Save** the form or form section.

Text Box

You can use the Text Box component to capture a single line of information. The input type for the Text Box component is defined by the **Type** property. You can choose from the following Type values: text (default), number, password, email, and url. The available properties depend on the Type:

- For **text**, you can specify the Min and Max Length values.
- For **number**, you can specify a minimum and maximum value.
- Type **password** allows input to be hidden with "*".
- Type **email** and **url** have built in validations for proper format.

Properties

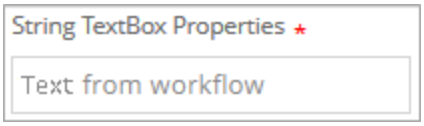
Example: Type = text

This example shows how you can dynamically bind the Text Box component itself (Model = vm.models.myTextBind) and several other fields: PlaceHolder, Required, RequiredMessage, and Tooltip.

Control Property Settings

Control Type	Text Box
Class	
Disabled	false
Id	42b660cd-4a80-25c1-70c0-247144fdb6d0
Label	String TextBox Properties
Max Length	50
Maximum Value	0
Min Length	1
Minimum Value	0
Model	vm.models.myTextBind
Placeholder	{{vm.models.myPlaceHolder}}
Read-only	false
Required	vm.models.myRequired
Required Message	{{vm.models.myReqMsg}}
Tab Index	
Tooltip	{{vm.models.myTextToolTip}}
Tooltip Duration	750
Type	text
Visible	true

Rendered Component



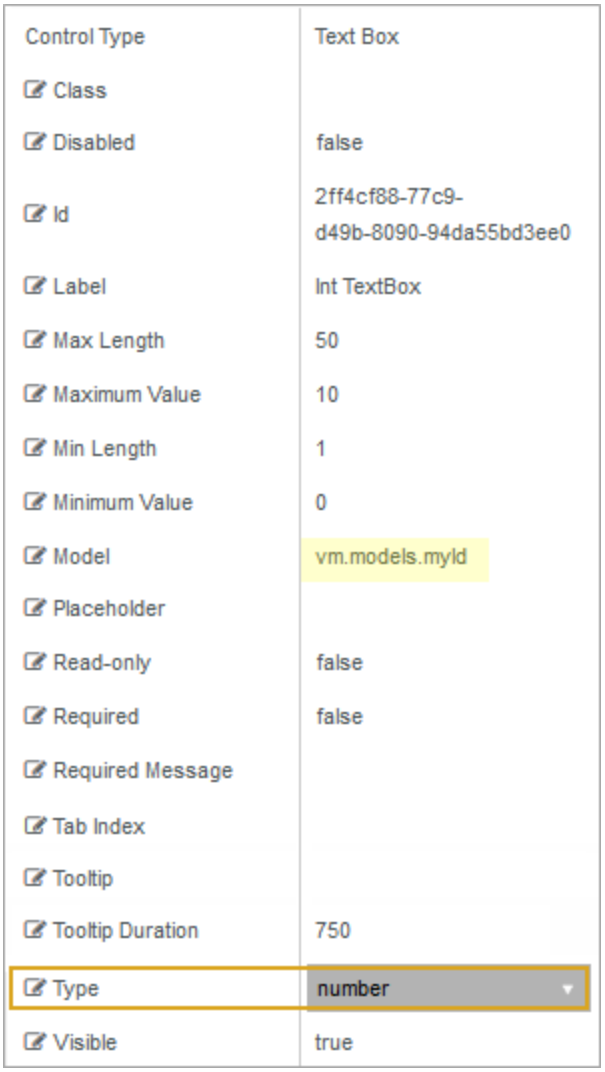
Workflow Arguments

Name	Direction	Argument type	Default value
myTextBind	In/Out	String	*Default value not supported*
myPlaceHolder	In/Out	String	*Default value not supported*
myRequired	In/Out	Boolean	*Default value not supported*
myReqMsg	In/Out	String	*Default value not supported*
myTextToolTip	In/Out	String	*Default value not supported*

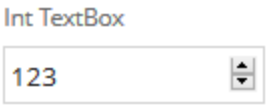
For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

Example: Type = Number

Control Property Settings



Rendered Component



The Text Box for Type=number is rendered with a selector that increments or decrements the typed number within the range of the Max Number and Min Number values. For example, if Max Number=10 and Min Number=0, the increments are 1, 2, 3, ...10.

Workflow Argument

Name	Direction	Argument type	Default value
myId	In/Out	Int32	*Default value not supported*

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Disabled** sets a control to disabled.
    - Must be true or false, or a binding beginning with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Label** is the value displayed in the label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Max Length** is the maximum number of characters in a text box.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessagel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Maximum Value** specifies the maximum value for a Text Box with `Type=number` or a Numeric Text Box. Leave blank for no maximum.
- **Min Length** is the minimum number of characters in a text box.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessagel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Minimum Value** specifies the minimum value for a Text Box with `Type=number` or a Numeric Text Box. Leave blank for no maximum.
- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.
    - The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
    - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
    - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
    - The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
    - If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

  If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

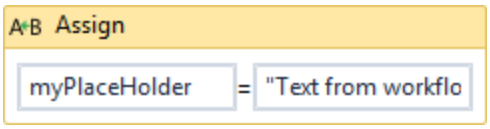
  If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

  Construction of the model in the workflow is done by assigning data from a provider.
- **Placeholder** is the prompt text displayed in a ghost style in an input box when nothing has been entered.
    - If this property is bound, it must start with {{vm.models. and end with }}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

  In the Text Box with Type = text example above, the Placeholder property is bound to the workflow through the `myPlaceHolder` argument. The syntax for the binding is as follows (note the double curly braces):

  ```
  {{vm.models.myPlaceHolder}}
  ```

  This argument allows the ghost text to be defined in the workflow as shown below using the Assign activity.

- **Read-only** makes the control read-only. It is set to false by default. If you want the component to be read-only, set the property value to true. It is typically used for an input box.

- Can be bound to a workflow argument or another control's value.
- A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
- If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

- **Required** makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.
  - Can be bound to a workflow argument or another control's value.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
  - If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)

  In the Text Box (Type = String) example above, the Required property is bound to the workflow through the `myRequired` argument. The syntax for the binding is as follows (note that curly braces are not needed here because 'Required' is a Boolean property):

  ```
  vm.models.myRequired
  ```

- **Required Message** is optional. It overrides the default "Required" message.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

  In the Text Box (Type = String) example above, the Required Message property is bound to the workflow through the `myReqMsg` argument. The syntax for the binding is as follows (note the double curly braces):

  ```
  {{vm.models.myReqMsg}}
  ```

- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
  - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.

- A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
- A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

  The tab index value should not match another control's tab index.
- A blank value (default) will not add a tab index in the HTML.

For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.

- **Tooltip** is the value to display when hovering over the control's label.

  Example

  Control Property Settings

Tooltip	The first name of the student or person.
Tooltip Duration	750
Type	text

  Rendered Component

  The property can be bound to the workflow through an argument, e.g., `myTextTooltip`. The syntax for the binding is as follows (note the double curly braces):

  `{{vm.models.myTextTooltip}}`

- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Type** is the input type for the component. Choose from the supported types. The directive for this attribute produces a standard <input> tag. The default for the Type value is set based on the control type (e.g., for the Text Box component, the default is `text`). Other Type values can be selected in the Value field: `password`, `email`, `number`, `url`.
- **Visible** makes the control visible or hidden.

- Can be bound to a workflow argument or another control's value. This property is dynamic.
- A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)
- When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

Textarea

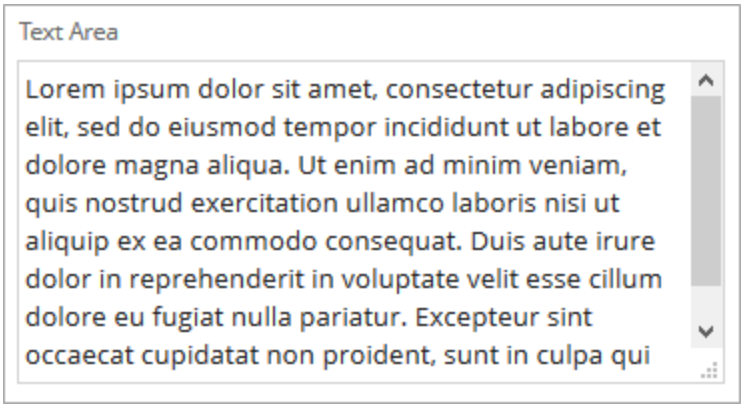
You can use the Textarea component to capture multiple lines of text.

Properties

Control Property Settings

Control Type	Textarea
Class	
Disabled	false
Id	id89a2269a-b0d2-ebd0-7b12-df67f2e7acb6
Label	Text Area
Max Height	30000
Model	vm.models.myTextArea
Read-only	false
Required	false
Required Message	
Tab Index	
Tooltip	
Tooltip Duration	750
Visible	true

Rendered Component



The rendered component is resizable; a scrollbar is added automatically.

Workflow Argument

Name	Direction	Argument type	Default value
myTextArea	In/Out	String	*Default value not supported*

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Disabled** sets a control to disabled.
    - Must be true or false, or a binding beginning with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Label** is the value displayed in the label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Max Height** is the maximum height in px, em, or %. The default is 30000 pixels. Binding is not supported for this property.

  This property allows you to set a specific height for the control so that a scroll bar is not shown on the form and all text entered by the user is properly printed or displayed in PDF. Specify only the numerical pixel value (do not add "px").

All existing sequences with a Textarea will get this new property with the default value and will still render initially as previously with about 3 lines of space for text.

- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.
    - The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
    - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
    - Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
    - The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
    - If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

  If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

  If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

  Construction of the model in the workflow is done by assigning data from a provider.

- **Read-only** makes the control read-only. It is set to false by default. If you want the component to be read-only, set the property value to true. It is typically used for an input box.
    - Can be bound to a workflow argument or another control's value.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Required** makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.

- Can be bound to a workflow argument or another control's value.
- A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
- If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

- **Required Message** is optional. It overrides the default "Required" message.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
  - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
  - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
  - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

    The tab index value should not match another control's tab index.
  - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.
- **Tooltip** is the value to display when hovering over the control's label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
    - When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

Time Picker

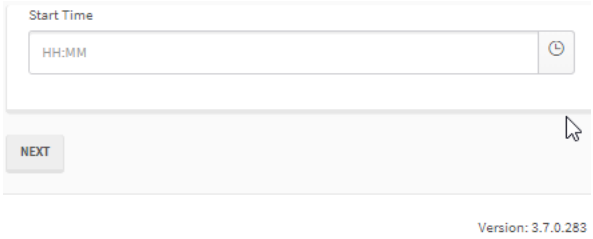
You can use the Time Picker component to allow form users to select a time in hours, minutes, and seconds. The time format is defined in the property settings for the component.

Properties

Control Property Settings

Control Type	Time Picker
Class	
Disable Input Text	false
Disabled	false
Format	HH:mm
Id	idf6990cb5-a22a-110e-08c2-0d8ccb891199
Interval	30
Label	Start Time
Model	vm.models.myTime1
Read-only	false
Required	false
Required Message	
Tab Index	
Tooltip	
Tooltip Duration	750
Visible	true

Rendered Component



Workflow Argument

Name	Direction	Argument type	Default value
myTime1	In/Out	DateTime	*Default value not supported*

The default time is 12:00 AM. To display a different default, set it in the workflow.

For workflow arguments used with the Calendar/Scheduler, refer to Default Argument Types for Components.

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Disable Input Text** disallows typing in the field. When true, dates must be selected from the date picker popup.
    - If this property is bound, it must start with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Disabled** sets a control to disabled.
    - Must be true or false, or a binding beginning with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Format** property settings depend on the control type used:
    - **Date Picker**:

      Specifies the format of the date and, when text input is allowed (Disable Input Text = false), the parsing of typed text.

      Examples: yyyy-MM-dd, MM-dd-yy (case sensitive: d for day of month, M for month, y for year)
    - **Time Picker**:

Specifies the format of the time and, when text input is allowed (Disable Input Text = false), the parsing of typed text.

Examples: HH:mm, hh:mm:ss tt (case sensitive: H or h for hour, m for minute, s for second)

Refer to Kendo UI documentation for more information.

- **Date Time Picker**:

  Combines the two above with both date and time.

- **Numeric Textbox**:

  Specifies format of the numeric value. e.g., n2 - 2 decimal places, c - currency with cents

> ⚠ The date parsing and formats for these controls changed in Forms Builder 3.5. If you used date formats that are no longer supported, you need to update and re-save affected forms.

- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
  - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
  - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
  - Binding is not supported for this property.
- **Interval** for the time picker. The default is 30 minutes.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myInterval}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Label** is the value displayed in the label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.
  - The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
  - This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).

- Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
- The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
- If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

Construction of the model in the workflow is done by assigning data from a provider.

- **Read-only** makes the control read-only. It is set to false by default. If you want the component to be read-only, set the property value to true. It is typically used for an input box.
  - Can be bound to a workflow argument or another control's value.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
  - If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)
- **Required** makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.
  - Can be bound to a workflow argument or another control's value.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
  - If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)
- **Required Message** is optional. It overrides the default "Required" message.

- If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
- Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
  - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
  - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
  - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

    The tab index value should not match another control's tab index.
  - A blank value (default) will not add a tab index in the HTML.

  For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.
- **Tooltip** is the value to display when hovering over the control's label.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Visible** makes the control visible or hidden.
  - Can be bound to a workflow argument or another control's value. This property is dynamic.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)
  - When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

This control has the capability to output an ISO 8601 String value which is converted to a DateTime or DateTimeOffset object (depending on the type of the workflow argument). For more information, refer to Date & Time Values and Offsets.

Tooltip

You can use the Tooltip component to add a customized tooltip to any page element. The tooltip can contain text, animation effects, and images. A unique CSS selector path in the Target Selector property identifies the element to which the tooltip applies.

Drop Tooltip components in a non-visible form section. Use as many Tooltip components as needed. This is a general purpose component that can be made to appear on any page element including form components, form sections, images, etc.

Most components already have a tooltip property, but for a complex element like the Grid, which may require the user to perform several operations (add, edit, delete), you may want to be more exact and only show a tooltip when, for example, the Add New Record button is selected.

Properties

Example: Tooltip with Text on a Grid Control

Control Property Settings

Control Type	Tooltip
Animation	{ "open": { "effects": "fade:in", "duration": 750 }, "close": { "effects": "fade:out", "duration": 1000 } }
Content	Add addresses that relate to student FA forms
Height	
Relative Position	bottom
Target Selector	#idadf80fb8-e4af-9e2d-5138-0e1b78cf6a74 .k-header.k-grid-toolbar .k-button
Tooltip Duration	2500
Width	

Rendered Component

Example: Tooltip with Image and Text on a Grid Control

Control Property Settings

Control Type	Tooltip
Animation	{ "open": { "effects": "fade:in", "duration": 750 }, "close": { "effects": "fade:out", "duration": 1000 } }
Content	 <p translate>Field is required</p>
Height	
Relative Position	bottom
Target Selector	#idadf80fb8-e4af-9e2d-5138-0e1b78cf6a74 .k-header.k-grid-toolbar .k-button
Tooltip Duration	2500
Width	

Rendered Component

- **Animation** — The value must be false or a valid JSON specification.

  (false must be all lowercase, true is not valid)

  Example

  { "open": { "effects": "fade:in", "duration": 750 }, "close": { "effects": "fade:out", "duration": 1000 } }

- **Content** is the text to display when hovering over the target element.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
  - Allowable suffix characters: starts with letter, then letters, numerals or underscore.
  - HTML is supported. If you want translatable HTML, make sure the element that contains the text has the "translate" attribute added. This will be picked when the POT file is generated and be available for translation. A string with no HTML does not need a "translate" attribute as it will be embedded in a div with the attribute.

  Example 1

  Content value for a tooltip with text:

  Only Add new addresses that relate to FA forms

  Example 2

Content value for a tooltip with an image and text that will be translated.

```
<div>
  <img src="Content/Custom/Images/MyCustomImage.png" />
  <p translate>My custom text</p>
</div>
```

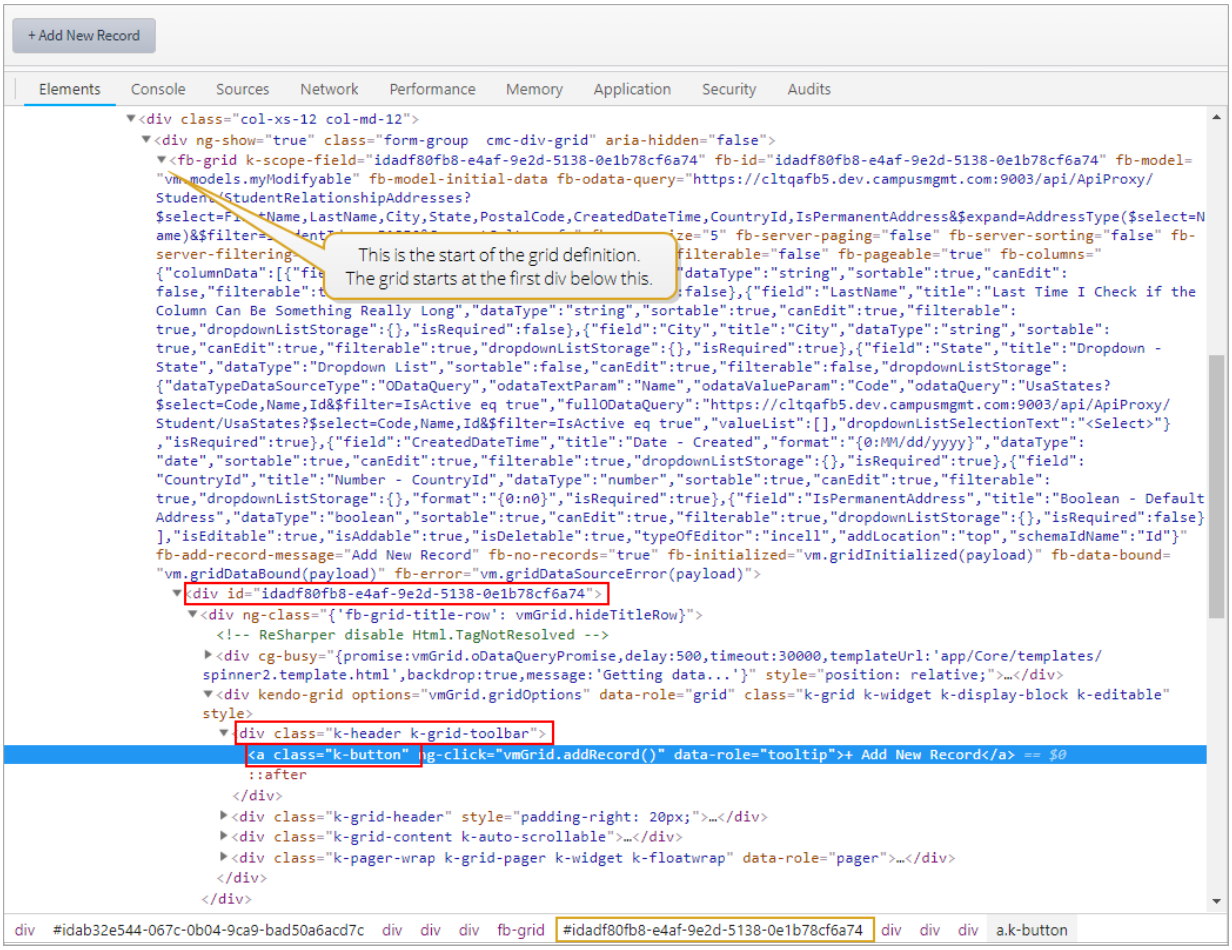
- **Height** is an optional property to override the default height.
    - Numeric values are treated as pixels.
    - This property cannot be bound.
- **Relative Position** is the position of the tooltip relative to the element. If there is not enough room in a container, a tooltip may override the chosen position.
- **Target Selector** is a CSS selector for the tooltip target. Choose a CSS selector which targets (usually a unique) container element for a tooltip. (Using an element would be too broad a selector.)

  For a class selector, use a dot in front of the class name: e.g., ".myClass"

  For an Id selector, use a # in front of the Id: e.g., "#myId"

  To find a unique CSS selector path to an element:

  1. Access the rendered form in a browser.
  2. Right-click the content of interest (e.g., **Add New Record** button) and depending on browser select the equivalent to the **Inspect Element** context item.
  3. On the **Elements** tab of the browser tools, locate the element's CSS class and Id.



The highlighted line shows the class name for the button. In our example it is:

- class="k-button"

The div above the button has the class name:

- class="k-header k-grid-toolbar"

The id for the div is found higher up in the hierarchy. In our example it is:

-

These three items are required to specify the Target Selector value in the Tooltip component. The format requirements are:

- Ids have a # prefix.
- Classes have a . (dot) prefix.
- Elements have no prefix.

In our example the Target Selector value is specified as:

- #idadf80fb8-e4af-9e2d-5138-0e1b78cf6a74 .k-header.k-grid-toolbar .k-button

Note that there is no space between .k-header and .k-grid-toolbar. This indicates that the CSS classes are in the same element (not in a hierarchy).

Also note that .k-button alone would not have been unique because there are other .k-button classes within the grid.

Lastly, note that we did not choose every element or class in the path hierarchy (we skipped some) and used only the ones that would make the path unique. (There are many combinations that would result in a unique path).

- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Width** is an optional property to override the default width.
    - Numeric values are treated as pixels.
    - This property cannot be bound.

Typeahead

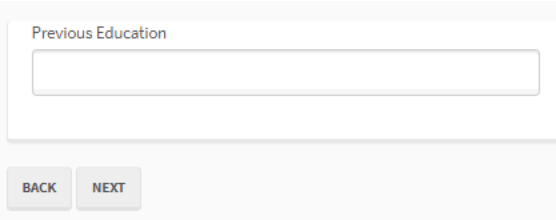
You can use the Typeahead component to perform a basic search on a list using a 'Contains' condition based on three characters. Auto-complete matches are displayed in a drop-down. This control type finds existing records and relates them to the entered characters.

Properties

Control Property Settings

Control Type	Typeahead
Class	
Disabled	false
Id	id9a99a220-6127-6021-b663-744e62b27733
Label	Previous Education
Lookup Display Member	Name
Lookup Query	PreviousEducationCodes?$select=Code,Name,Id&$filter=IsActive eq true&$orderby=Name
Model	vm.models.myTypeAhead
Product	Student
Read-only	false
Required	false
Required Message	
Tab Index	
Tooltip	
Tooltip Duration	750
Visible	true

Rendered Component



Workflow Argument

Name	Direction	Argument type	Default value
myTypeAhead	In/Out	String	*Default value not supported*

- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Disabled** sets a control to disabled.
    - Must be true or false, or a binding beginning with "vm.models.".
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Label** is the value displayed in the label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Lookup Display Member** is the name of the property in the OData query string to use for the display. .

  For example, if the query string for a list of PreviousEducationCodes contains the Code, Name, and ID fields, the Lookup Display Member value can be set to Code, Name, or ID.
- **Lookup Query** is the OData query string to retrieve values for the control.

  The following is an example of an OData query string that retrieves the Code, Name, and ID values from the PreviousEducationCodes table, where `isActive` equals true and the returned values are sorted by Name.

  ```
  PreviousEducationCodes?$select=Code,Name,Id&$filter=IsActive eq true&$orderby-y=Name
  ```
- **Model** is required for binding to a workflow argument or another control. If the Model property is not specified, the component will be displayed on the form, but any values the user enters on the form cannot be captured or used in the workflow.

- The Model value must always start with "vm.models.", e.g., vm.models.myArgument.
- This value may initialize the control, and may be updated by the control, and if matched to a workflow argument, is available in the workflow (readable or writable).
- Ensure your model argument is defined in your workflow for custom components if it is used in the workflow. Otherwise, a workflow argument is not required.
- The casing of an argument used in the workflow MUST match the "vm.models." suffix casing.
- If the model addresses CustomProperties or MultiValueCustomProperties, the property identifier string must be enclosed in single quotes, e.g., vm.models.myentity.CustomProperties['mycustomprop']

If an OData query is specified and this binding is specified, it will be overwritten with the value of the OData query results and thus be available read-only in the workflow.

If only "Model Data" is specified and the workflow variable is either not initialized or set to an empty array, this value will be initialized to the "Model Data" value.

Construction of the model in the workflow is done by assigning data from a provider.

- **Product** indicates the product from which OData query results are returned. Select from:
  - Student
  - CRM
  - Occupation Insight

The selected product must be configured in the <products> section of the Renderer web.config file.

The default Product value will be "Student" if "Student" is selected in the <Select Provider> list on the Fields tab.

The default Product value will be "CRM" if "CRM" is selected in the <Select Provider> list on the Fields tab.

Select "Occupation Insight" in the Product property if the source of the query will come from a different data source other than Student/CRM. For more information, refer to Build Queries for Occupation Insight.

A form can have multiple controls that retrieve data from different providers. For example, a form can have a control that is populated by a query to the Student database. The same form can have another control that retrieves data from Occupation Insight.

Specify the query to retrieve data from the selected provider using the Lookup Query or OData Query property (as applicable for the control). The query contains only the URL specific part of an OData URI. The Base URL and Product will be supplied by the configuration.

- **Read-only** makes the control read-only. It is set to false by default. If you want the component to be read-only, set the property value to true. It is typically used for an input box.

- Can be bound to a workflow argument or another control's value.
- A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
- If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

- **Required** makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.
  - Can be bound to a workflow argument or another control's value.
  - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
  - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
  - If comparing to a string, it must be in single quotes.
  - (true and false must be all lowercase)
- **Required Message** is optional. It overrides the default "Required" message.
  - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myMessage}}.
  - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Tab Index**allows you to specify the order of elements that are brought into focus when the user presses the Tab key on the rendered form. Allowed values are -1, 0, and positive numbers.
  - A value of "-1" removes the element from the sequential tab order preventing keyboard users from focusing on it.
  - A value of "0" means the element is ignored in the tab order, but that does not mean users cannot tab to and focus an element.
  - A value of "1" sets the element as the first item to gain focus when tabbing through the page followed by any higher numbered tab indices, followed by any other keyboard focusable elements such as buttons. required fields, and CAPTCHA.

    The tab index value should not match another control's tab index.
  - A blank value (default) will not add a tab index in the HTML.

For more information, refer to https://html.spec.whatwg.org/multipage/interaction.html#the-tabindex-attribute and https://www.alexlande.com/articles/cross-browser-tabindex-woes/.

- **Tooltip** is the value to display when hovering over the control's label.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myTooltip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.

  We suggest using the **Tooltip** property on the Typeahead component to instruct the user to enter search text to view all matching values.
- **Tooltip Duration** is the amount of time in milliseconds a tooltip is displayed (default=750). The value must be greater than 0. If it is set to 0, a form validation error will occur.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myToolTip}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
    - When the control is **not visible** (usually based on some condition) and is a required field, the required field validation will not be triggered.

Lookup Queries for CampusNexus CRM Metadata

For any drop-down or search controls that will be populated via a lookup query, the CampusNexus CRM user needs to enter values for the **Lookup Display Member** and **Lookup Sort Member** attributes. The **Lookup Query** and **Lookup Value Member** property settings should have default values (if applicable for the selected property) as these are currently specified in the metadata.

View Summary

You can use the View Summary component to capture the content of forms within a sequence before submitting the information that will be persisted to your system. The data displayed via the View summary component is retrieved from the durable instancing table instead of workflow tracking.

Important Notes

- In **e-signature** sequences, a form with View Summary component should be placed **before** the *Default-Frame* where the redirection to the e-signature provider occurs. This allows the user to review and, if necessary, change form input before proceeding with the redirection. Do not place a form with View Summary component after the *Default-Frame* form.
- Any form with View Summary component should set *Include in View Summary* to false.

Forms Builder 3.4 and later provides enhanced validation when saving a sequence. A sequence cannot be saved if at least one form in the sequence has a View Summary component, but the Include in View Summary property is not selected on any of the forms in the sequence.

The placement of the View Summary component within the sequence determines the content captured by the component. If View Summary is placed on the last form in a sequence, it captures the content of all preceding forms.

The data displayed by the View Summary component is not editable. After reviewing the data, the user may select the Back button to return to the form whose data is captured and make revisions. Therefore, the View Summary component must be placed on a form that contains a Back button, i.e., it should not be placed in an end form.

We recommend placing the View Summary component on a separate form within the sequence (just before the end form) along with HTML components that provide instructions to:

- Review the data
- Use the Back button to make revisions on the forms within the sequence

The form property "Include in View Summary" controls whether the content of a form is displayed by the View Summary component. "Include in View Summary" is selected by default. For more information, refer to Save Forms and Form Properties.

JSON Debug Info output is not rendered when the sequence contains a View Summary component or a PDF is created.

Properties

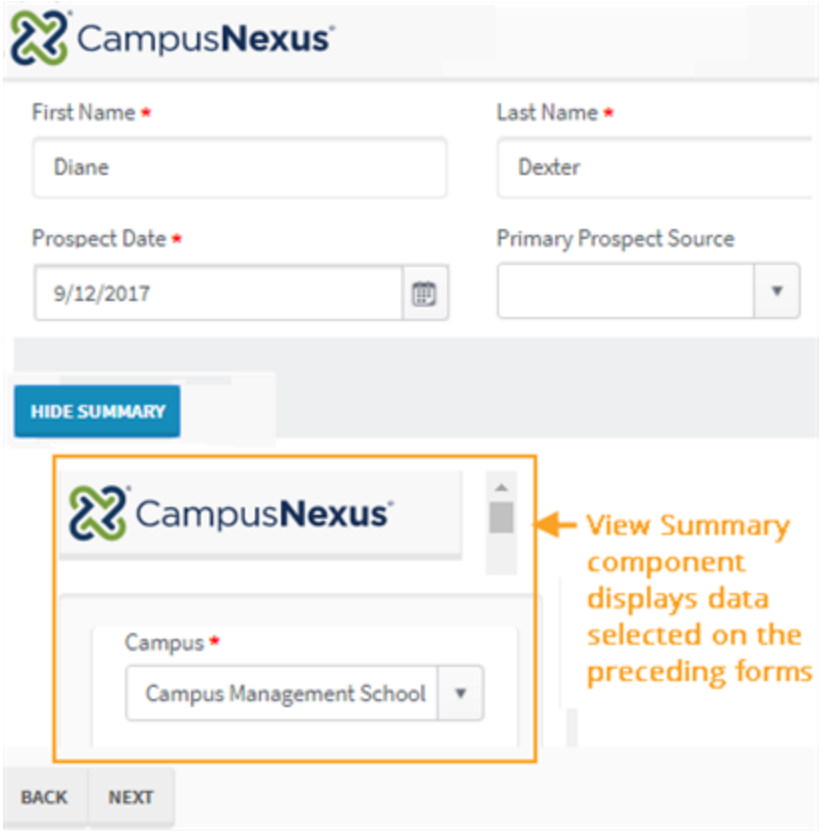
Control Property Settings

Control Type	View Summary
Button Class	btn btn-primary
Class	
Create PDF Button Text	Create View Summary PDF
Display As Optional	false
Hide Button Text	Hide Summary
Id	id15adbcc8-7e76-03ca-fad1-c79196a7f20a
Print Visible	false
View Button Text	View Summary
Visible	true

Workflow Argument

— NA —

Rendered Component



- **Button Class** is the CSS class for the button. The default is `btn btn-primary`.
    - The class `btn btn-primary` is a Bootstrap class that renders as a blue button with white text.
    - You can customize the CSS Class by selecting a different Bootstrap class or adding your own class in a custom CSS.
- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- **Create View Summary PDF** is the value displayed in a link or button used to create a PDF file of the sequence.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Display As Optional** is set to false by default. The value must be true or false (all lowercase) or a binding beginning with "vm.models.".
    - Set this to true if you would like the user to decide if they would like to view the summary or not. The View Summary button is displayed on the rendered form, but the summary itself is hidden. If the user selects the button, the summary is displayed, and the button label changes to "Hide Summary".
    - When set to false, the button is not displayed, and the summary is displayed.
- **Hide Button Text** is the value displayed in a link or button used to hide the sequence summary view.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Print Visible** makes the print button visible when set to true (default = false). When the print button is selected, Forms Builder creates a PDF that can then be saved or printed.

    - Can be bound to a workflow argument or another control's value.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
    - If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)
- **View Button Text** is the value displayed in a link or button used to display the sequence summary.
    - If this value is bound, it must be enclosed in double braces, e.g., {{vm.models.myLabel}}.
    - Allowable suffix characters: starts with letter, then letters, numerals, or underscore.
- **Visible** makes the control visible or hidden.
    - Can be bound to a workflow argument or another control's value. This property is dynamic.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].
    - An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=). If comparing to a string, it must be in single quotes.
    - (true and false must be all lowercase)

Notes:

- The View Summary functionality is not supported in Internet Explorer (IE) and Firefox browsers. When the View Summary button is selected in IE or Firefox, the following message is displayed: "*View Summary is not supported in IE or Firefox. Use Chrome or Edge.*"
- In the PDF file created by the View Summary control, if a form contains controls with long text strings in the Option Label property, the label may overwrite the label of the adjacent control. In the rendered form, the Option Label text will be truncated to fit in the space available to the control.

Control Property Settings

When a Field or Component is dragged to the Layout pane, the Property Settings pane is refreshed and the Control Property Settings are displayed.

The OData service exposes the metadata of the database Fields as defined in the Entity Data Model (EDM) for CampusNexus. Refer to OData Queries.

The Control Property Settings include the control type, which is read-only. Other properties can be modified. The list of editable properties (marked with ✎) varies depending on the control type.

Binding Properties

If you want to use a different control type for a property from the CampusNexus model than what the default is, you can create a custom component, bind it to the field using the **Model** property, and save the component. For more information, refer to Components.

When binding controls, **String** and **Integer** properties require the Model value to be enclosed in **double curly braces**, for example, `{{vm.models.myTooltip}}` for Tooltip or `{{vm.models.myMinValue}}` for a Text Box of Type Number.

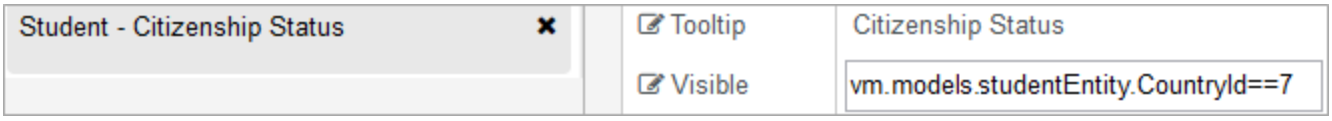
Boolean properties do not need the curly braces, for example, `vm.models.myRequired`.

You can also bind fields using dynamic **AngularJS** expressions in the property settings (refer to https://-docs.angularjs.org/guide/expression). For example, you could define the Visible property on the Citizenship Status field to be visible only when the following expression evaluates to true:

```
vm.models.studentEntity.CountryId==7
```

Where the CountryId value 7 represents United States, and Country is a field in the same sequence.

With this expression, if the user selects a country other than the United States (i.e., country==7), the Citizenship Status field will not be displayed on the form.



Notation for Array Variables

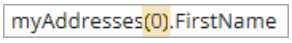
Array variables in the Property Settings pane of Form Designer use AngularJS notation with "square brackets" [].

Example



Array variables in Workflow Composer require VB.NET notation with "rounded brackets" ().

Example



AngularJS Expression Sandbox Security

When you use AngularJS expressions in the Property Settings pane of Forms Builder, be careful **not** to use arbitrary JavaScript in HTML template expressions. Use the expression sandbox only for data binding.

- If you dynamically generate AngularJS templates or expressions from user-provided content, you are at risk of cross-site scripting (XSS) attacks.
- If you do not generate your AngularJS templates or expressions from user-provided content, you are not at risk of these attacks.

Database Tables for Property Settings

The control types and their property settings are persisted as part of the JSON string stored when saving the form. The following database tables hold the control types and their properties:

- The *ControlTemplate* table stores the available control types and has one row per control type instance.
- The *ControlProperty* table stores all of the property settings that can have values specified for each control type. This table has one row per instance of each property setting.

Update of Properties

Prior to Forms Builder 3.4, when properties were added to controls (such as the Visible property added in Forms Builder 3.3), you had to drag the updated control into the Layout pane, fill out the properties again, and remove the original control from the Layout pane.

In Forms Builder 3.4 and later, when properties are added to the *ControlProperty* table, the properties are automatically updated in the Property Settings pane. Default values are assigned to any new properties. You no longer need to replace the original control in the Layout pane.

To save new properties in JSON/HTML format in the database, you still need to re-save the form.

Editable Properties

The following table lists common editable properties for database fields and components. For more details about component properties, refer to the Components topics.

For Boolean type properties, the literal values `true` and `false` must be entered in lowercase. Uppercase TRUE/FALSE will always evaluate to false.

Editable Properties

Property	Description	Required	Default	Data Type
Allow Multiple Files	File Upload component: Specify whether multiple files can be uploaded.		false	Boolean
Class	Optional CSS class specific to the control. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.	No		

Property	Description	Required	Default	Data Type
Disabled	Specify whether the field is disabled on the rendered form.		false	Boolean
Extensions Allowed	File Upload component: Specify the allowed extensions of files to be uploaded.		doc, docx, gif, jpg, pdf, png	String
Format	Display format for the Masked Text Box or Date Picker control types. Masked Text Box: For more information about the formats, refer to Kendo UI: MaskedTextBox. Date Picker: For more information about date formats, refer to Date Formats.			Refer to EDM Attribute: Format.
Grid Columns	Specify the grid columns displayed in Single-select Search control types. The "field" value must match the fields returned by the Lookup Query.		[{"field": "Name", "title": "Name"}]	
Hyperlink Target	Specify where to open the link. Possible values are _self (default), _blank, _parent, and _top.		_self	
Id	Id is a globally unique identifier (GUID) for the field instance within the form in which it is included. It is automatically created by Forms Builder.	Yes		String
Label	Specify the label assigned to the field on the rendered form.			String
Link Text	Specify the link text for a Hyperlink control.			String
Lookup Display Member	Specify the name of the column that is displayed in the control.	Yes	Name	String
Lookup Query	Specify the OData query associated with the property when a list control such as a drop-down is used to display the property. **Lookup Queries for CampusNexus CRM Metadata** For any drop-down or search controls that will be populated via a lookup query, the CampusNexus CRM user needs to enter values for the **Lookup Display Member** and **Lookup Sort Member** attributes. The **Lookup Query** and **Lookup Value Member** property settings should have default values (if applicable for the selected property) as these are currently specified in the metadata.	Yes	null	OData Query
Lookup Sort Member	Specify the name of the database column that is to be used to sort the values returned by the lookup.	Yes	Name	String

Property	Description	Required	Default	Data Type
Lookup Value Member	Specify the name of the column in the query that is used as the value of a selection.	Yes	Id	String
Max Size Allowed	File Upload component: Specify the maximum file size (in bytes) for files to be uploaded. The default is 0 (unlimited). If this control is bound to a model, depending on the product, the Max Size allowed may be limited by the service used to persist the attachment.		0	
Max (for number type)	Specify the maximum value for an Input Type of number. The default is 0 (unlimited).			String
Max Length	Specify the maximum length of a string property.		field specific	String
Min (for number type)	Specify the minimum value for an Input Type of number. The default is 0 (unlimited).			String
Min Length	Specify the minimum length of a string property.		1	String
Model	Specify the model of the component in the Entity Data Model (EDM) for CampusNexus for a bound control (e.g., vm.models.<entityname>.<propertyname>) or any unbound control (e.g., vm.-models.CustomTextbox). All Model bindings defined or used in a workflow must be **arguments** (not variables). Any sequences that are bound to variables will no longer work.			String
Option Label	Specify the label for the options in a Drop-Down List control.		<Select>	Enum
Page Size	Specify the size of the page when paging is true.		100	Int32
Placeholder	Specify the ghost prompt text in an input box before anything is typed.			String
Product Name	Specify "Student" or "CRM" depending on whether Anthology Student or CampusNexus CRM is used with Forms Builder.	Yes	Student	String
Read-only	Specify whether a control is read-only.		false	Boolean
Required	Specify whether input for a field or component is required on the rendered form.		false	Boolean
Required Message	Specify the validation message displayed on the rendered form.			String
Tooltip	Specify the text to be displayed when the cursor is placed over the component.			String

Property	Description	Required	Default	Data Type
Type	Applicable only to the Text Box control. Specify the input type for the component. The directive for this property produces a standard <input> tag. The default for the Type value is `text`. Other Type values that can be selected from a drop-down list in the Value field are: `password`, `email`, `number`, `url`.			Enum
Url	Specify the URL for a Hyperlink control.			String
Visible	Specify whether a field or component is visible on the rendered form. Like most properties, this property can be also bound to a variable in a workflow or another control's value.		true	Boolean

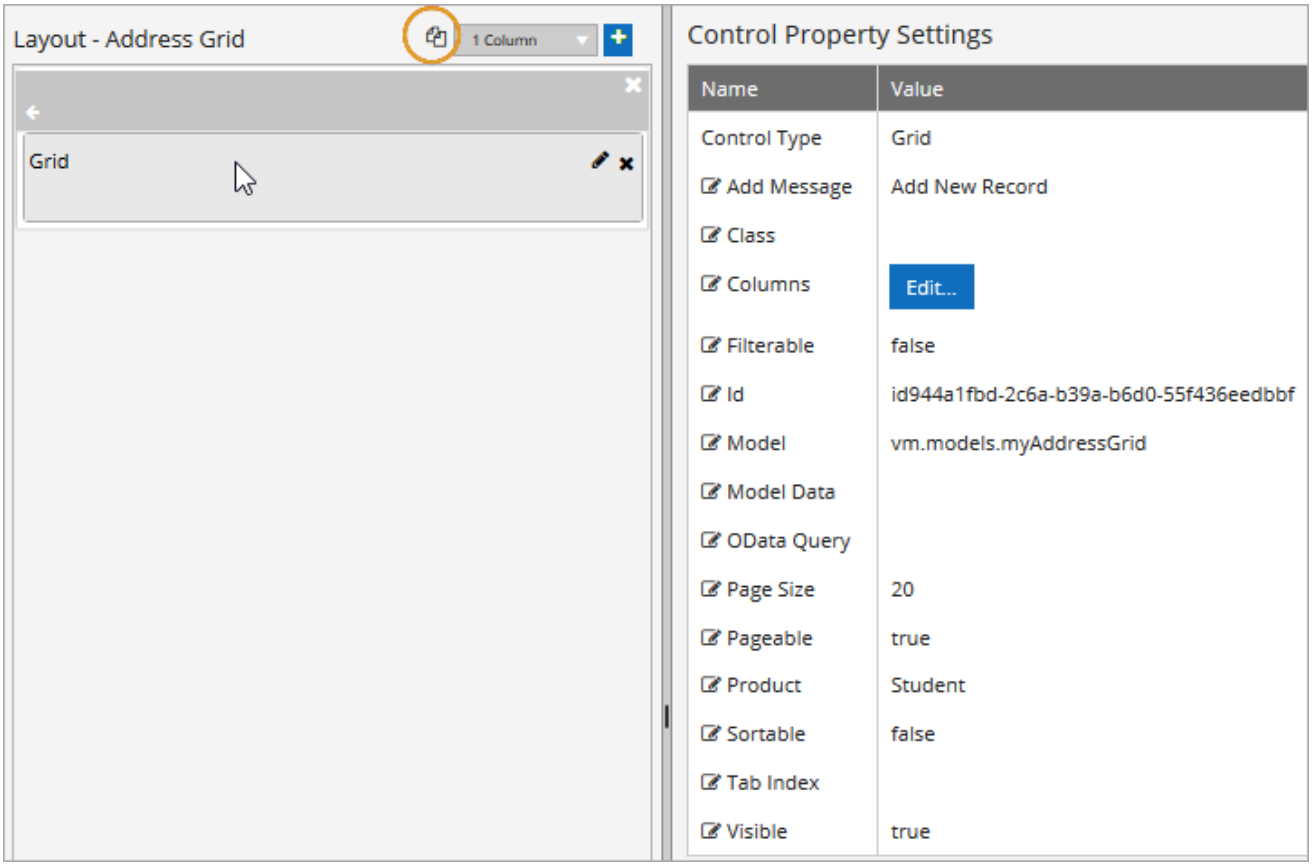
EDM Attribute: Format

Value	Description	Example
d	Short date	6/15/2009
D	Long date	Monday, June 15, 2009
F	Full date/time	Monday, June 15, 2009 1:45:30 PM
G	General date/time	6/15/2009 1:45:30 PM
M	Month/day	June 15
u	Universal sortable date/time	2009-06-15 20:45:30Z
Y	Year/month	June, 2009
t	Time	13:45:30
n0	Number (0 precision)	123
n1	Number (1 precision)	123.4
n2	Number (2 precision)	123.45
c0	Currency (0 precision)	$123
c1	Currency (2 precision)	$123.45
p0	Percent (0 precision)	100 %
p2	Percent (2 precision)	100.00 %

Copy and Paste Controls

You can copy and paste individual controls in the Layout pane of Form Designer. You can copy and paste a control from one form to another form or from one form section to another form section in the same form. You can copy and paste custom controls (Components) and controls that originate from the Fields tab.

1. Open a form and select a control in the Layout pane. The icon appears above the Layout pane.

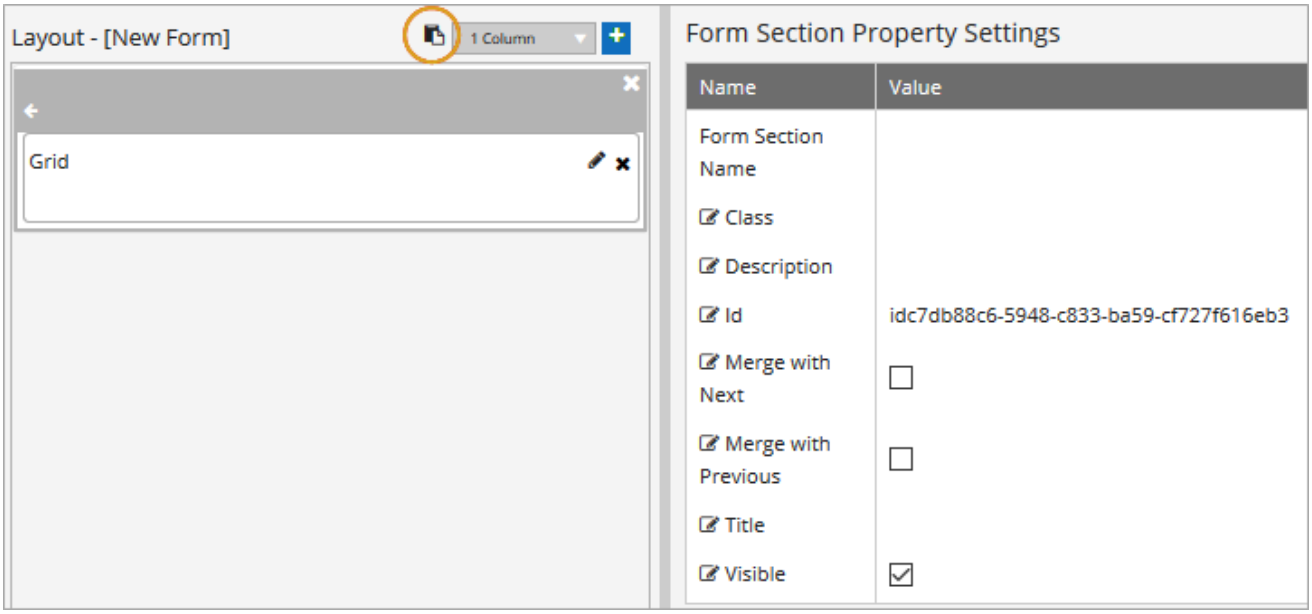


2. Select the icon. The selected control is copied to the clipboard.

   If you select a different control and select again, only the latest item copied will be saved to the clipboard.

3. Open another form or start a new form and select a form section. The icon appears above the Layout pane.

4. Select the icon. The control is pasted into the form section.



The pasted control is assigned a unique Id value. All other properties are carried over from the copied control.

Limitations

- You can copy only one control at a time. If you need to copy multiple controls, you have the option to create, save, and reuse Form Sections.
- You must stay within Form Designer to copy and paste a control. If you browse back to the Designer home page, the in-memory copy is lost.
- You cannot copy a control across browser sessions or browser tabs.

Custom Styles

Forms Builder enables you to customize the styling of forms and form elements using cascading style sheets (CSS). Style sheets determine the look and feel of your forms including logos, controls, backgrounds, font, etc.

It is not the purpose of this topic to teach you CSS. Many references can be found on the web on how to create style sheets. One good example is at w3schools.com/css. There are also free CSS editors with extra features, among them Microsoft Visual Studio Community Edition, but any text editor can be used as well.

There are 4 ways to style sequences:

1. **Apply styles globally for all sequences**

   Create a CSS file and then select the Custom Content tile on the home page to upload the file to the CustomContent table in the database. In the Custom Content tile, select the check box in the Global column, the uploaded CSS file will apply globally to all sequences. For more information, refer to the section **Applying a Global CSS** in Custom Content.

2. **In the Sequence Designer**

   Add CSS files and campus logos to the CustomContent table in the database and reference them by selecting **Apply Themes Dynamically** in the Sequence Properties area in Sequence Designer. This option enables institutions to configure form sequences to be displayed at different URLs with different branding using a single instance of Forms Builder (Designer and Renderer).

   For more information, refer to Sequence Designer and Multiple Renderer URLs.

   OR

   Upload custom css files to the database and then select a custom css file in the **Theme-Custom Content** drop-down list in the Sequence Properties. The selected css file will be applied to the entire sequence. For more information, refer to Sequence Designer and the section **Applying a CSS to a Sequence** in Custom Content.

   > You cannot save a sequence with selections for both **Apply Themes Dynamically** and **Theme-Custom Content**.

3. **Add the style to an HTML component on a form.**

   This can be set to Visible=false so it does not reserve space on the form. The style will apply to the form it is contained in and all remaining forms in the sequence. Once it is loaded, it will apply also if you go back in the sequence (which is why it is best in the first form in a sequence). Use the style tags like this:

   ```
   <style>
       Styles here
   </style>
   ```

4. **Use Custom Content**

Upload a file and refer to it with a simple directive tag in an HTML component. Refer to the **CSS Files (Style Sheets)** section in Custom Content for details on how to reference the uploaded file.

Create a Style for a Label Control

Each control has a Class property. When you fill in this property with your own class name (or multiple class names), your class (or classes) will be added to a top level HTML `<div>` element for a control. The class name you specify will need to be defined in a custom style sheet file and will then need to be uploaded to the database. For information on uploading files, refer to Custom Content.

Unlike most other controls in Forms Builder, the **Label** control does not use AngularJS directives to define custom HTML tags which are then translated to "regular" HTML/CSS/JS when they are rendered in a browser.

1. In the Property Settings on Form Designer, fill in the class name for a Label control with the name `my-custom-style`. (The convention is lowercase with hyphenation ("snake case"), but any unique name would work).

   When a Label is rendered, it will now look like the following:

   ```
   <div class="form-group my-custom-style cmc-div-label">
   <span id="my-id-13232" class="label label-default">This is my label</span>
   </div>
   ```

2. To apply a style to this class, create a style sheet file (`.css`) which contains your style.
3. Upload the custom style sheet using the steps described in Custom Content.
4. Let's say we want to make this label red with a light blue background and force it to wrap if it is too long.

   Since you want to style the `<span>` within the `<div>`, the following contents of your `.css` file will do this. The class at the top level (refer to step 1) has a 2nd selector which tells it to drill down to the `span` element and apply the style in the braces to it.

   ```
   .my-custom-style span {
   white-space: normal;
   display: inline-block;
   background-color: lightblue;
   color: red;
   }
   ```

   For control types that use directives, e.g., **input** (Text Box), the custom style must be defined similar to the following example to override the default style. The `!important` rule forces the override of the default style.

   ```
   .my-custom-style input {
   white-space: normal;
   display: inline-block;
   background-color:lightblue !important;
   color: red !important;
   }
   ```

   This style only affects the Text Box input control itself not the label above control.



5. On other controls (look at the HTML generated in the browser), you may have a different 2nd selector like `label`.
6. Notice that there is another class above: `cmc-div-label`. This fixed class is in all labels. If you create a style for it, all labels on your site will have this style.

```
.cmc-div-label span {
white-space: normal;
display: inline-block;
background-color: lightblue;
color: red;
}
```

7. Similarly, `form-group` is in all controls, so a style (if appropriate) can be applied to all controls.

One caveat is that multiple definitions for the same element have a priority scheme. The `label` and `label-default` classes in the `span` are a default Bootstrap style and produce white text with a gray background. Your style may not override a higher priority style. If your style doesn't seem to take effect, most browsers have a development mode where you can examine the styles being applied to each element on the page.

Replace the Logo

You can designate image files as logo files and associate a logo with a sequence. For more information, refer to Navigation Bar Logos.

Video: Combine Form Sections on a Form

The following remains as an example of custom styling techniques, however, Forms Builder 3.6, this particular style can be accomplished with the new Merge with Next and Merge with Previous properties on two successive Form Sections. Refer to Merged Form Sections for information.

This short video demonstrates how you can modify form sections using a custom class name. It includes the following steps:

- Modifying and capturing a style using browser developer tools
- Creating a custom CSS file using form section Ids as CSS selectors

  To link the custom CSS file to a form, upload the file using custom content and then link it to a form using the functionality of the Theme-Custom Content property.
- Not shown in the video, a 3rd way is to add an HTML component to the page with <style>….</style> tags and set property Visible=false.

Download the video (wmv file; 26 MB; 4:30 min.)

Exclude a Component from a PDF

All components in Forms Builder have a Class property. Class is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file.

For example, the class on a component could be named "pdfNoCheckBox". In a CSS file, using one of the four techniques listed above for adding CSS to sequences, use the following CSS:

```
<style>
@media print {
   .pdfNoCheckBox {
      display: none;
   }
}
</style>
```

This will remove the component from the PDF created from HTML which loads this CSS. One needs to be aware of what will happen to the layout of the rest of the components when this is done and perhaps arrange the layout so that there is minimal impact.

Date Formats

In Forms Builder the input format for date values is controlled by the settings on the **Format** property. The text typed in a date field on a rendered form is parsed and displayed based on the specified format. The format is case sensitive: **d** for day, **M** for month, **y** for year, for example, MM/dd/yyyy or dd-MM-yy. The **Disable Input Text** property must be set to false to enable text input on a field.

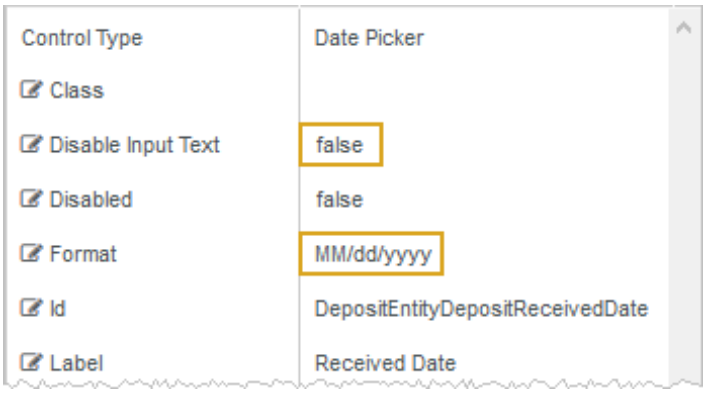
If you are entering dates elsewhere in the product that are not displayed to the user such as dates in properties, queries or expressions, the preferred format is the ISO 8601 format. This format is universal and is always parsed correctly.

Time:	**Date**:
2016-03-09T10:20	2016-03-09
2016-09-09T22:23	2017-04 – year and month only
2016-09-10T22:23-05:00 – with time zone offset	
2016-09-10T22:23:00Z - UTC time	

Example 1: Admissions Deposit - Received Date Field

The Property Settings for the Received Date Field specify the Format as **MM/dd/yyyy**. The Disable Input Text property is set to **false** (default: true).

Control Property Settings - Received Date Field



Rendered Field

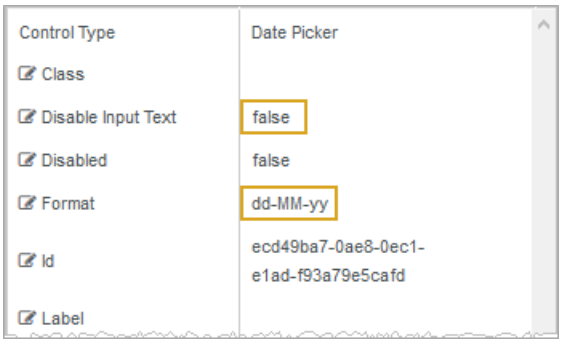
After text input, the rendered form displays the Received Date as follows:



Example 2: Date Picker Component

The Property Settings for the Date Picker were edited to specify the Format as **dd-MM-yy**. The Disable Input Text property is set to **false** by default.

Control Property Settings - Date Picker Component



Rendered Component

After selecting a date, the rendered form displays the date as follows:



Regardless of how the date format is defined in Forms Builder, any date values stored in the database use the `<DateTime>` data type. Workflows also use the `<DateTime>` data type.

Date & Time Values and Offsets

The Date Picker, Date Time Picker, and Time Picker controls have the capability to output an ISO 8601 string value which is converted to a **DateTime** or **DateTimeOffset** object in a workflow (depending on the type of the workflow argument).

Example

Assume one is using a control on March 8 at 10AM in Florida during Eastern Standard Time:

- **Date Picker**

  The Date Picker control should have **Ignore Time** set to `true` and will only output the date portion of the ISO 8601 string.

vm.models output example:	“2018-03-08”
DateTime object will show:	Mar 8, 2018 at midnight, no offset
DateTimeOffset:	same and likewise no offset

- **Date Time Picker**

vm.models output example:	“2018-03-08T10:00:00.000-05:00”
DateTime object will show:	Mar 8, 2018 10 AM, offset is lost
DateTimeOffset:	same but will have Offset property set to 5 hours

- **Time Picker**

vm.models output example:	2018-03-08T10:00:00.000-05:00”
DateTime object will show:	same as Date Time Picker, but typically you will ignore the date and use only a formatted String as the time
DateTimeOffset:	same but will have Offset property set to 5 hours

When binding vm.models values for these controls directly to an entity argument property, if the associated database `Datetime` column is **not** nullable, the entity data type is `DateTime` instead of `Nullable<DateTime>`. This means a direct model map to an entity argument property must have Required set to `true` so that client-side validation will prevent submittal of a value if it is null (no value). Failure to do so would cause an error in the workflow and a subsequent abort of the workflow.

If the entity property is `Nullable<DateTime>`, you still might not want to store a null value in certain circumstances (since it might be invalid for the situation). In that case you could mark it required in the client, or conditionally in the workflow, check your argument with the property `myDate.HasValue` and create a descriptive server validation error if it is false.

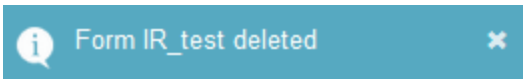
Delete Forms

You can delete a form in Form Designer. When the delete option is selected, Forms Builder checks if the form is included in any sequences. If the form is included in at least one sequence, a validation message is displayed, and the delete process is not allowed to continue.

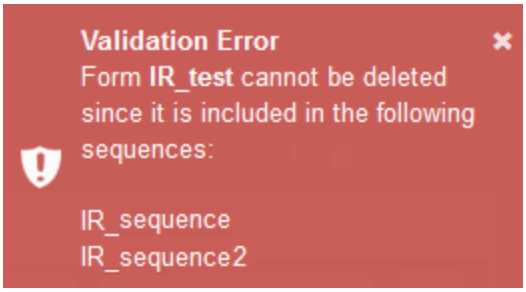
Forms can easily be modified to fit any new requirements. It may not be necessary to delete them. Just keep in mind that any additional entities or properties that were not included in the initial sequence must be manually added to workflow definition. For more information, refer to Update a Form After Creation of a Sequence.

1. In Form Designer, select the form you want to delete.
2. Select **Delete Form** in the action bar.

   The following message is displayed: *"Are you sure you want to delete <form name>?"*
3. To proceed, select **OK**.
4. If the form is **not** included in a sequence, the form is deleted and a confirmation message is displayed.

5. If the form is included in one or more sequences, a validation error is displayed.

Fields

The Fields tab in Form Designer enables you to find the database fields from the CampusNexus object model that can be used when building forms.

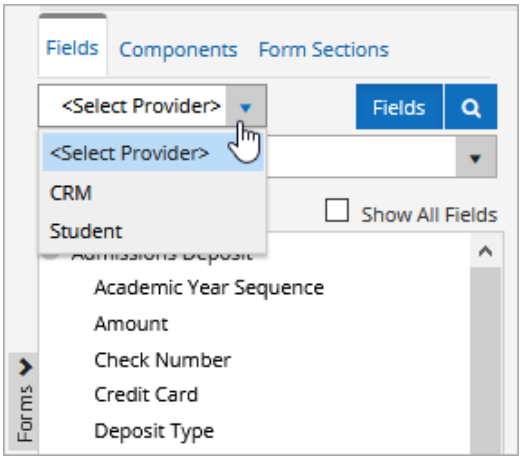
To use fields:

1. Select a database provider. Refer to Select the Database Provider.
2. Select an entity. Refer to Find Fields in an Entity.
3. Locate the desired field in the Field Properties pane. Refer to Search for Fields and Show All Fields.
4. Drag the field into the Layout pane.

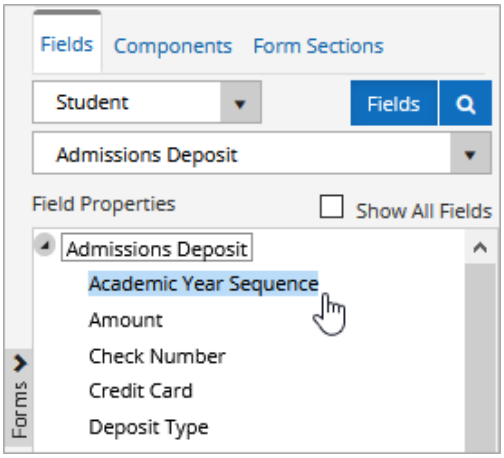
   You must drag a control/field over another control. You cannot drag a control over an empty space in the Layout pane. If you drag a control over the top half of an existing control, it will be inserted before the existing control; if you drag it over the bottom half, it will be inserted after.
5. Edit the property settings as needed. Refer to Control Property Settings.

Select the Database Provider

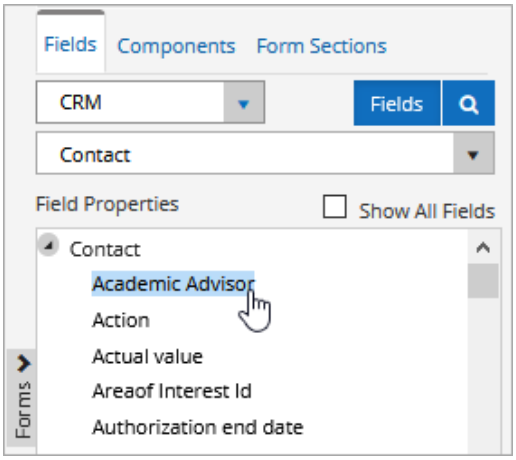
If Forms Builder is used with multiple database providers, the first step is to select the provider. Select **CRM** for CampusNexus CRM or **Student** for Anthology Student. The database provider selection determines which fields are available for use when building forms. If only one database provider is available, the <Select Provider> list will not be displayed.



When **Anthology Student** is the database provider, the list of entities on the Fields tab shows entities such as Admissions Deposits, Applicant Areas of Study, Applicants, and so on.



When **CampusNexus CRM** is the database provider, the list of entities on the Fields tab shows entities such as Address Type, Area, Area of Interest, and so on.



The list of entities displayed for the Student and CRM databases is configured using the **Entity and Entity Properties Visibility** option. Refer to Settings.

Find Fields in an Entity

1. Select the **Fields** button. The Entities drop-down list is activated (replacing the Search field).
2. Select an entity from the **Entities** drop-down list. The Field Properties pane displays the fields associated with the entity.

   Depending on the selected entity, navigation properties may be available. Navigation properties signify a relationship between the selected entity and another entity in the data model. Select ⌄ to expand the navigation property entity.



Search for Fields

1. Select  next to the Fields button. The Search field is activated (replacing the Entities drop-down list).
2. Enter the first 3 characters of a field name in the **Search** field. The Field Properties pane displays all instances of the field grouped by entities.

   Depending on the selected entity, navigation properties may be available. Navigation properties signify a relationship between the selected entity and another entity in the data model. Select  to expand the navigation property entity.

Show All Fields

The Show All Fields check box controls whether a subset or the entire list of fields (categorized by entities) in the CampusNexus data model is exposed.

The Show All Fields check box is cleared by default. When cleared, a prescribed list of entities is exposed. The prescribed list of entities is aligned as much as possible with what is exposed in Forms Builder 2.x.

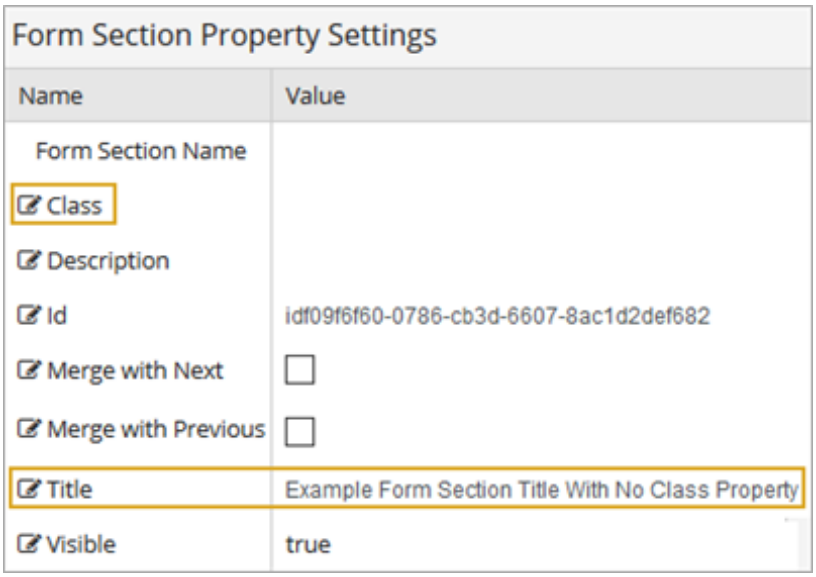
The list of entities displayed for the Student and CRM databases is configured using the **Entity and Entity Properties Visibility** option. Refer to Settings.

Form Sections

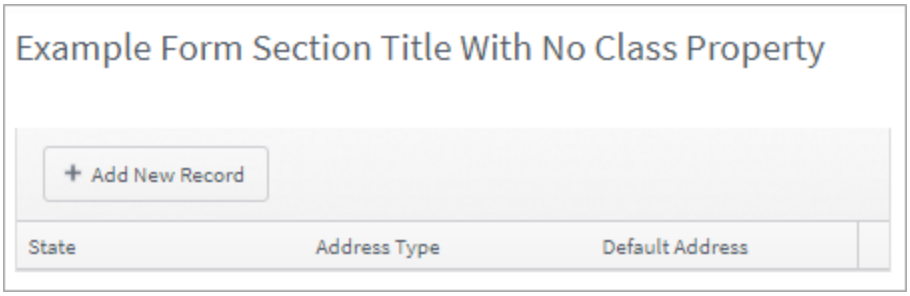
A Form Section has a set of fields and section dividers in the Layout pane of Form Designer. Form sections can be styled as distinct segments within a form, and they can be saved to reusable units.

Style Form Sections

For any Form Section, you can use the **Class** and **Title** properties to change the rendered display of the Form Section.



Default Rendering of a Form Section Title without Class Property



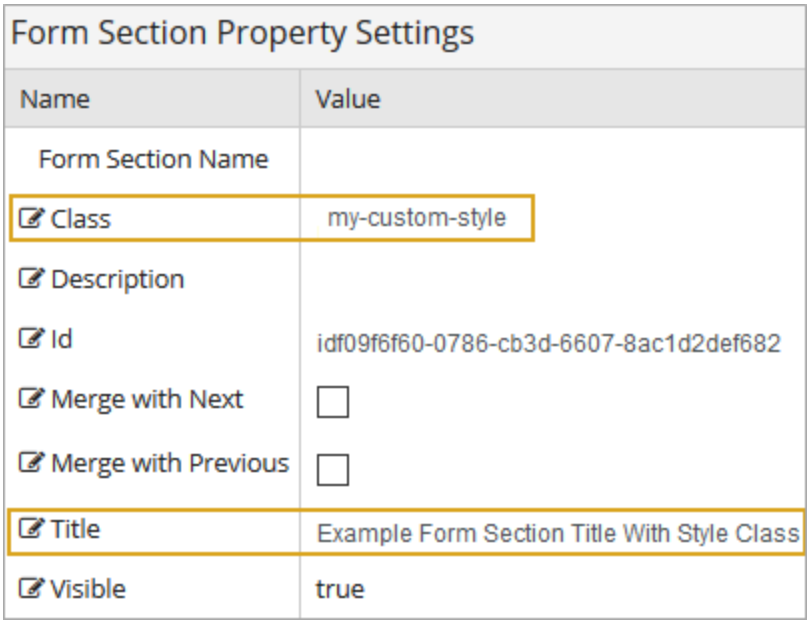
Apply a Style to a Form Section

When a custom style defined in a .css file and specified in the Class property is applied to a Form Section, it modifies entire Form Section, not just the Title.

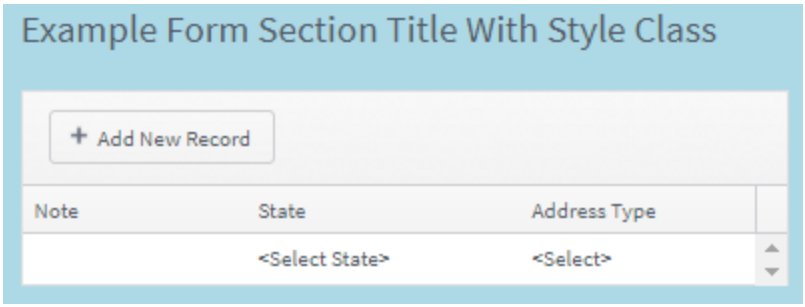
The following "my-custom-style" definition is rendered as shown below.

```
.my-custom-style {
   white-space: normal;
```

```
background-color:lightblue !important;
}
```



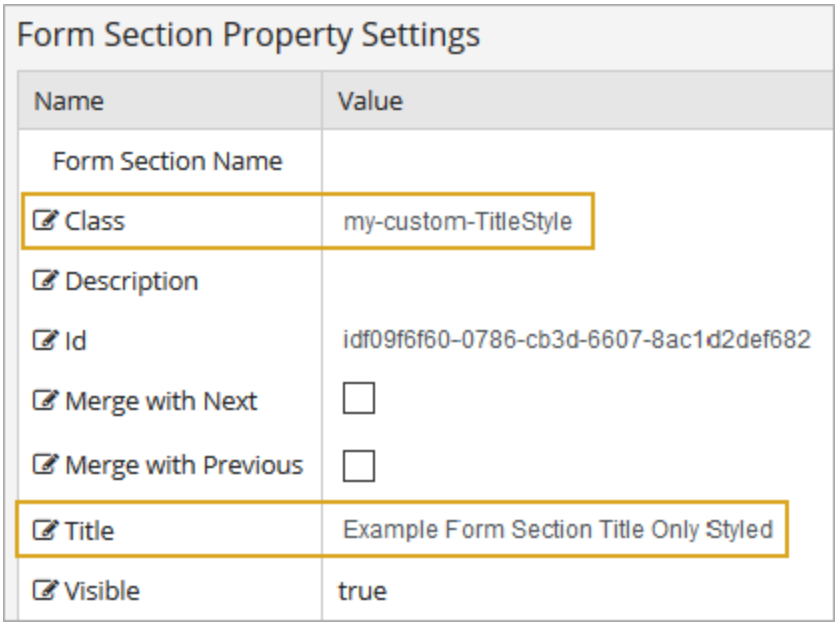
Rendering of a Form Section Title with Class Property Applied to Entire Form Section



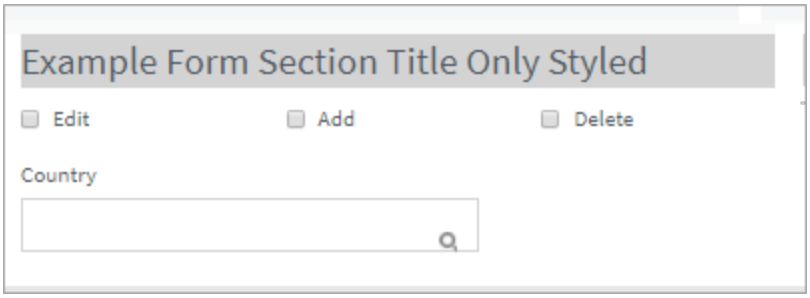
Apply a Style to a Form Section Title

To apply a class style only to the Form Section Title, use a .css definition similar to the following example:

```
.my-custom-TitleStyle > span.fb-form-section-title {
    width: 100%;
    white-space: normal;
    display: inline-block;
    background-color:darkgray !important;
```



Rendering of a Form Section Title with Class Property Applied Only to Title



Hide Empty Space for a Form Section with E-Sign Components

When you design a form with hidden fields or Form Sections, you may need to hide the empty space for e-sign components to ensure that the form is rendered as intended. To hide the space for e-sign components:

1. Place the e-sign components in their own **Form Section** (usually at the bottom of a form).
2. In the Form Section Property Settings, specify the Class name **hideDocuSignWhiteSpaceInFormSection**.
3. Save the form. Forms Builder will not render the Form Section on the screen but will allow it to render when it is printed to a PDF.

Visible Property

The Visible property is available for Form Sections in Forms Builder 3.4 and later. If set true (default), all contents of the Form Section will be displayed when the form is rendered. If set to false, all contents of the Form Section will not be displayed at render time.

This property can be dynamically bound using a vm.models. value or an expression that evaluates to true or false. The default is true.

The Visible property can be used for many reasons, e.g.:

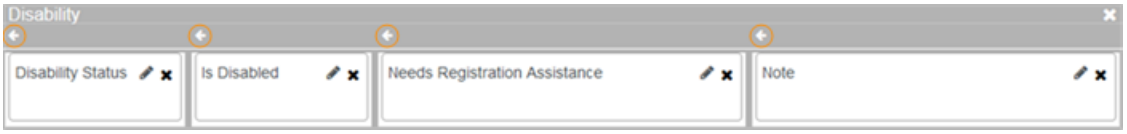
- You have added a custom script to an HTML component and stored it as a Form Section. You want the ability to drop this custom script on any page. However, if the Form Section cannot be hidden, there will be an empty panel on the page.
- You have workflow logic bound to an argument or just bound variable logic on a page which makes a given Form Section conditionally visible. This makes it possible to hide a group of controls programmatically instead of hiding each control and being unable to hide the panel that contains them.
- You specify an argument (e.g., vm.models.showSection) in the Model property of a Checkbox control and then use the same argument in the Visible property of a Form Section. The Form Section will be displayed only if a user selects the check box.

Layout Enhancements

The column layout options for Form Sections provide additional flexibility. You can select from 1-12 columns, set up columns with different widths, and you can merge Form Sections using the 'Merge with Next' and 'Merge with Previous' properties.

Flexible Column Layout

The Form Section below has 4 fields with different widths. The first 2 fields take up 2 columns each; the other 2 fields take up 4 columns each. The maximum width is 12 columns. The left pointing arrows above the controls indicate that the width of each field can be decreased, but not increased.



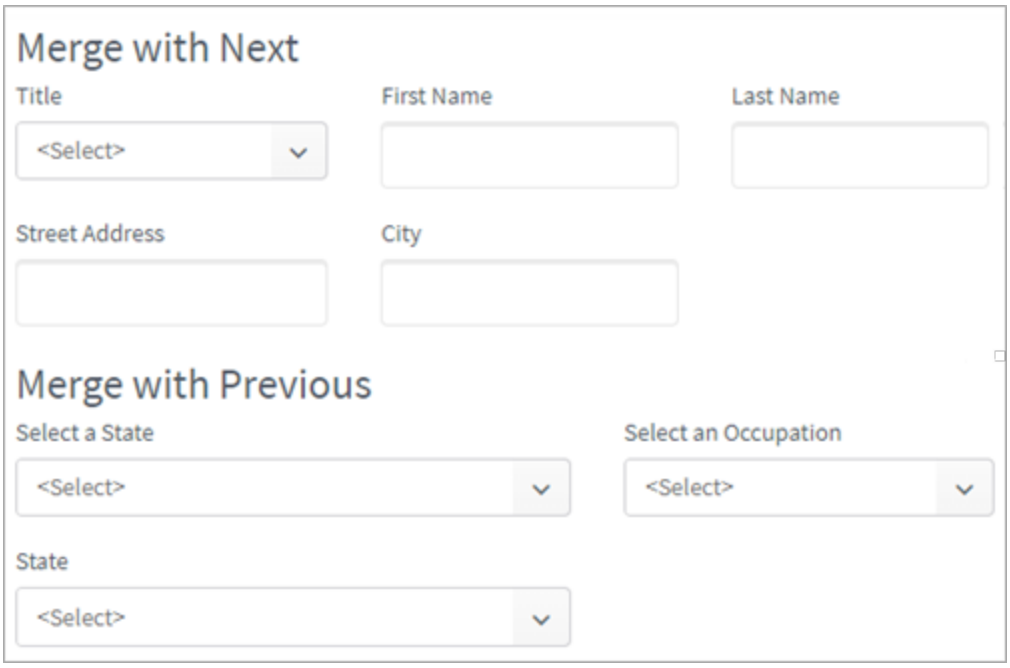
After reducing the width of the Note field, right pointing arrows appear above the fields indicating that there is space available to increase the width of any field.



Merged Form Sections

To render Form Sections without the horizontal rule that divides Form Sections by default, you can use the "Merge with Next" and "Merge with Previous" properties. These properties must be paired, i.e., when "Merge with Next" is selected for a Form Section, the next Form Section must have "Merge with Previous".

The image below shows how two Form Section are rendered when the first Form Section has the "Merge with Next" property and the following Form Section has the "Merge with Previous" property. Note that there is no visible section divider.



Reusable Form Sections

Forms Builder provides ability to create and save Form Sections as independent reusable units. Once saved, Form Sections are accessible from the Form Sections tab in Form Designer and can be dropped on to other forms. A Form Section in a form is saved with all of the included form properties as well as any applicable properties that exist for the Form Section itself.

Create and Save a Form Section

1. In Form Designer, select **New** to activate the Layout pane.

   — OR —

   Select the vertical **Forms** tab to the left of the Field Properties area and select a form. Continue with step 4.

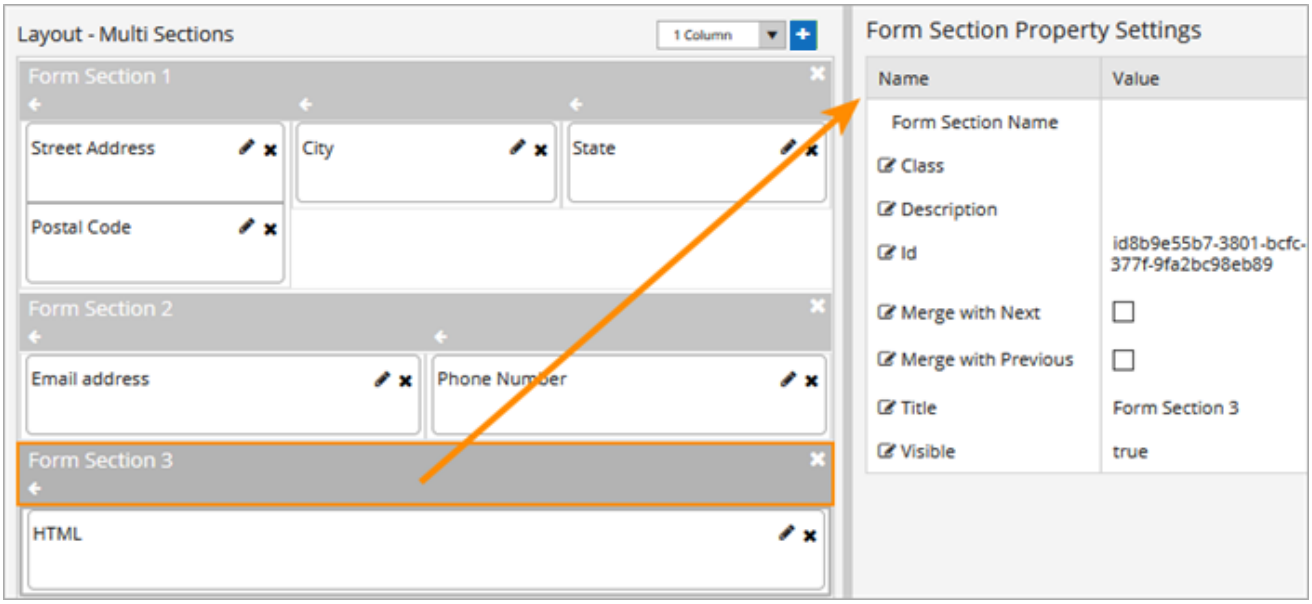
2. Select a **Column** layout (1, 2, or 3 columns) and select  to add a section divider.

   In Forms Builder 3.6 and later, the Column layout options include 1-12 columns and the Merge With Next / Merge With Previous properties.

3. From the Fields panel, drag a **Field** into the Layout pane, and/or select a **Component** from the Components panel and drop it into the Layout pane. Specify any applicable values in the **Property Settings** pane.

   Repeat this step as necessary to assemble the desired Form Section in the Layout pane.

4. Select the section divider in the Layout pane and specify the **Form Section Property Settings**.

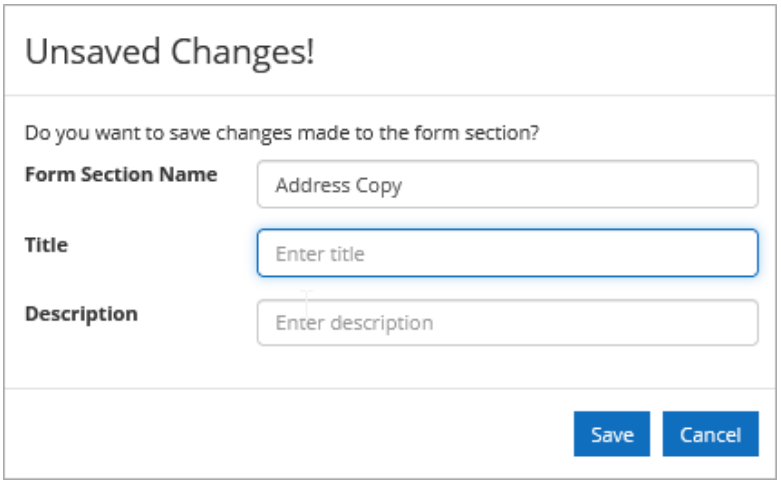


- **Class** is an optional CSS class (or space separated classes) added to the top level of the control. CSS specific to the control can be applied. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
- Description is an optional field to describe the Form Section.
- **Id** is required. It can be any valid JavaScript id attribute value. (Must start with a letter followed by 0 to 9, a to z, dash, or underscore characters).
    - Using a globally unique identifier (GUID) from GuidGen or GuidGenerator prefixed by at least one letter prevents a clash with any other id.
    - Id should contain only a to z (uppercase or lowercase), 0 to 9, dash, or underscore. It should not have spaces.
    - Binding is not supported for this property.
- **Merge with Next** and **Merge with Previous** enable you to merge sections on a form. For more information, refer to Merged Form Sections above.
- **Title** is displayed at the top of the Form Section when the form is rendered.
- **Visible** makes the Form Section visible or hidden. This property can be dynamically bound using a vm.models. value or an expression that evaluates to true or false.

Forms Builder validates the Forms Section properties. Invalid properties will be marked with an icon ( ). The Layout pane will display a red border on the affected Form Section. If you try to save a Form Section with errors, an error message will be displayed.

5. Select the **Save Section** button. The section divider in the Layout pane now indicates the Title of the Form Section and, in parentheses, the Form Section Name.

6. In the Unsaved Changes dialog specify the **Form Section Name** (required), **Title** (optional), and **Description** (optional). Select **Save** to continue.



The Form Section tab indicates the Form Section Name after the Form Section is saved.

All applicable data for the Form Section is persisted to the database.

Edit a Form Section

1. In Form Designer, select **New** to activate the Layout pane.
2. On the **Form Sections** tab, select a Form Section, and drop it into the **Layout** pane.
3. Edit the Form Section by adding or removing Fields and/or Components and/or editing the Form Section Property Settings.
4. Select the section divider for the Form Section.

   Select **Save Section** to retain the Forms Section Name, Title, and Description.

   — OR —

   Select **Save Section As** to change the Forms Section Name, Title, and Description.

When a Form Section is edited as described above after the Form Section has already been used in one or more forms, the edits are **not** propagated to the form instances where the Form Section is used.

Add a Form Section to a Form

1. In Form Designer, create a new form.
   a. Select **New** to activate the Layout pane.
   b. Select a **Column** layout (1, 2, or 3 columns) and select  .
   c. From the Fields panel, drag a **Field** into the Layout pane, and/or select a **Component** from the Components panel and drop it into the Layout pane. Specify any applicable values in the **Property**

Settings pane.

d. Repeat this step as necessary to assemble the desired form in the Layout pane.

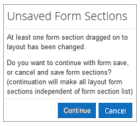
— OR —

a. Select the vertical **Forms** tab to the left of the Field Properties area and select a form.

2. Select the **Form Sections** tab and drag a Form Section to the **Layout** pane.

Edit a Form Section in a Form

1. Select the vertical **Forms** tab to the left of the Field Properties area and select a form that contains a Form Section.
2. Edit the Form Section.
3. Select the section divider of the Form Section and select **Save Section** or **Save Form Section As**.



The "Unsaved Form Sections" dialog is displayed:

- If you select *Save* instead of *Save Section* or *Save Form Section As* after editing a Form Section within a form.

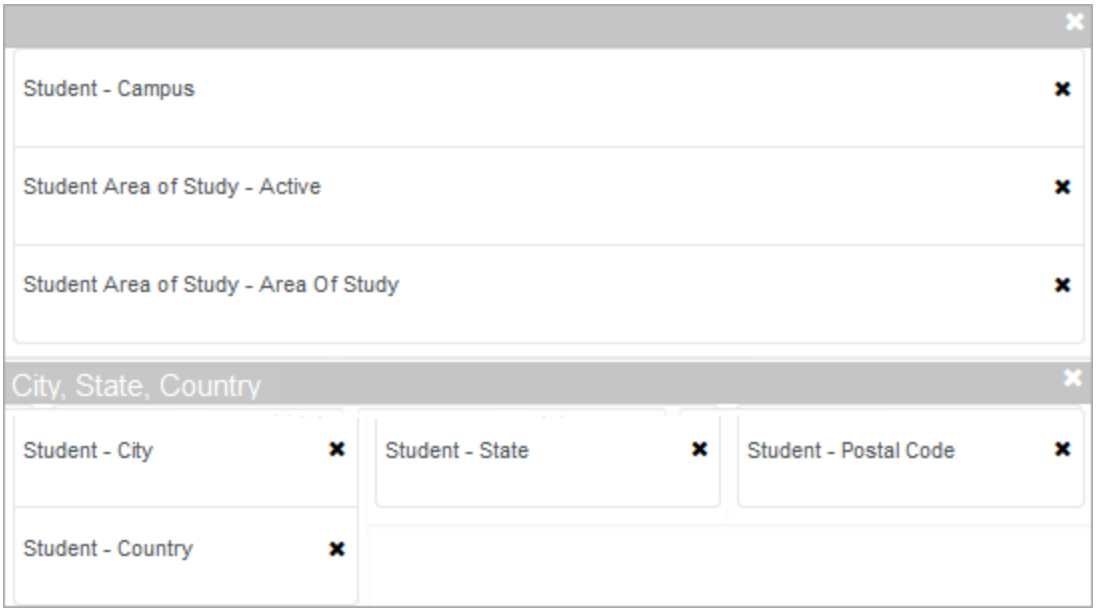
  — OR —

- If you select  to navigate back to the home page.

When this dialog is displayed, you have the following options:

- Select **Cancel** and then select **Save Section** to save the changes to the Form Section. In the Unsaved Changes dialog specify the **Form Section Name** (required), **Title** (optional), and **Description** (optional). Select **Save** to continue. The revised Form Section is saved to the database and listed on the Form Sections tab.
- Select **Continue** to save the form. In the Unsaved Changes dialog, specify the **Form Name** (required), **Title** (optional), and **Description** (optional). Select **Save** to continue.

  Notice that the section divider in the Layout pane no longer indicates the *Form Section Name* after the form is saved.



Changes made to a Form Section in a form are not saved to the Form Section in the database. The Form Section in the database will retain the original Form Section fields.

Delete a Form Section

On the **Form Sections** tab, select a Form Section, and select **Delete Section**. The Form Section is removed from the Form Sections tab.

If the deleted Form Section is used in a form, the *Form Section Name* is removed from the section divider, but the fields of the Form Section remain on the form.

Delete a Form Section from a Form

In the Form, select a Form Section, and select the icon below the title bar of the **Form Section**.

Multiselect for Single Property Collections

There are several instances within the Anthology Student and CampusNexus CRM models where multiple selections of a single property are needed. For example, when specifying student demographic data, a student's ethnicity may require selecting multiple ethnicity values. Another example is a Prospect Inquiry where a user can select multiple programs.

Currently, the ability to achieve this capability by dragging the applicable properties from the data model into the Layout pane is not supported. Instead, the Multiselect control from the Components tab in Form Designer must be used to achieve this capability.

Ethnicities List

1. In **Form Designer**, select **New** or access a previously created form that requires an Ethnicity List field.
2. Select the **Components** tab and drag the **Multiselect** control to the Layout pane.
3. In the **Control Property Settings** pane for the Multiselect, specify the following values:

Label	=	Ethnicity List
Lookup Display Member	=	Name
Lookup Query	=	Ethnicities?$select=Code,Name,Id&$filter=IsActive eq true&$orderby=Name
Lookup Sort Member	=	Name
Lookup Value Member	=	Id
Model	=	vm.models.**prospectInquiryEntity.Student.**EthnicitiesList (Use this Model value if you are saving data for a prospect.)
— OR —		
Model	=	vm.models.**studentEntity.**EthnicitiesList (Use this Model value if you are saving data for a student.)

Accept the defaults for the remaining values.

Control Property Settings	
Name	Value
Control Type	Multiselect
Class	
Disabled	false
Filter Type	contains
Header Template	
Id	idcead06a7-619d-216b-fee2-e8a1f3034325
Item Template	
Label	Ethnicity List
Lookup Display Member	Name
Lookup Query	Ethnicities?$select=Code,Name,Id&$filter=IsActive eq true&$orderby=Name
Lookup Sort Member	Name
Lookup Translation Members	
Lookup Value Member	Id
Model	vm.models.prospectInquiryEntity.Student.EthnicitiesList
Option Label	<Select>
Product	Student
Read-only	false
Required	false
Required Message	
Tab Index	
Tag Template	
Tooltip	
Tooltip Duration	750
Value List	Edit...
Visible	true

4. Access the **Sequence List** and view the rendered form.
5. Select the control labeled **Ethnicity List** and verify that:

- The Ethnicity values from your database are listed.
- Multiple values can be selected.
- Your selections are saved to the appropriate table (*AmElectronicLeadsAmRace* for the prospectInquiryEntity or *SyStudentAmRace* for the studentEntity).

Programs List

1. In **Form Designer**, select **New** or access a previously created form that requires a Programs List field.
2. Select the **Components** tab and drag the **Multiselect** control to the Layout pane.
3. In the **Control Property Settings** pane for the Multiselect, specify the following values:

Label	=	My Programs
Lookup Display Member	=	Name
Lookup Query	=	Programs?$select=Code, Name, Id&$filter=IsActive eq true
Lookup Sort Member	=	Name
Lookup Value Member	=	Id
Model	=	vm.models.**prospectInquiryEntity.Student.**ProgramList (Use this Model value if you are saving data for a prospect.)
— OR —		
Model	=	vm.models.**studentEntity.**ProgramsList (Use this Model value if you are saving data for a student.)

Accept the defaults for the remaining values.

Control Property Settings

Name	Value
Control Type	Multiselect
Class	
Disabled	false
Filter Type	contains
Header Template	
Id	id2208cd60-db58-3e60-7266-b0b8cf437b9e
Item Template	
Label	My Programs
Lookup Display Member	Name
Lookup Query	Programs?$select=Code,Name,Id&$filter=IsActive eq true&$orderby=Name
Lookup Sort Member	Name
Lookup Translation Members	
Lookup Value Member	Id
Model	vm.models.prospectInquiryEntity.Student.ProgramsList
Option Label	<Select>
Product	Student
Read-only	false
Required	false
Required Message	
Tab Index	
Tag Template	
Tooltip	
Tooltip Duration	750
Value List	Edit...
Visible	true

4. Access the **Sequence List** and view the rendered form.
5. Select the control labeled **My Programs** and verify that:

- The Program values from your database are listed.
- Multiple values can be selected.
- Your selections are saved to the appropriate table (*AmProspectProgram* for the prospectInquiryEntity or *SyStudent* (column AdProgramID) for the studentEntity).

Save Forms and Form Properties

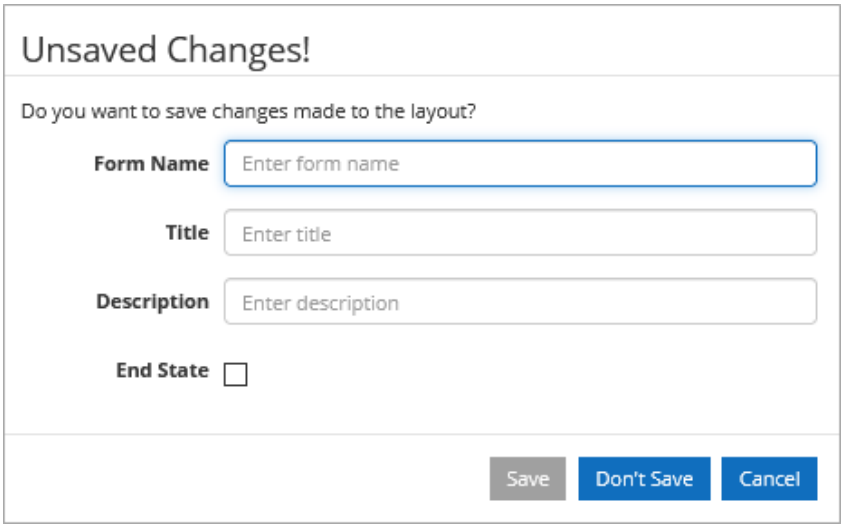
In Forms Builder individual Fields and/or Components are assembled in the Layout pane and saved as a Form. Forms are then grouped into sequences. End users view the saved sequences.

To create a form:

1. Access the Form Designer workspace.
2. Select the **New Form** button.
3. Select the number of **Columns** you want to use for each row on the form.
4. Drag one or more **Fields** or **Components** into the Layout pane. Position the Fields/Components in the selected column/row grid.
5. Select or specify values in the **Control Property Settings** pane for each Field/Component. For more information, refer to Fields and Components.
6. Select or specify values in the **Form Properties** pane. For more information, refer to Form Property Settings.
7. Select **Save Sequence**. The Unsaved Changes dialog is displayed.

Unsaved Changes Dialog

Each form in a sequence has a Form Name and optional properties such as Title, Description, and End State. You specify these properties in the Unsaved Changes dialog when saving a form.



1. Specify a **Form Name** (required).
2. Specify the **Title** (optional) on the webpage when the form is rendered.
3. Specify a **Description** (optional) that helps you identify the form.

The Form Name, Title, and Description are displayed in the Sequence List and can be used as search and filter criteria.

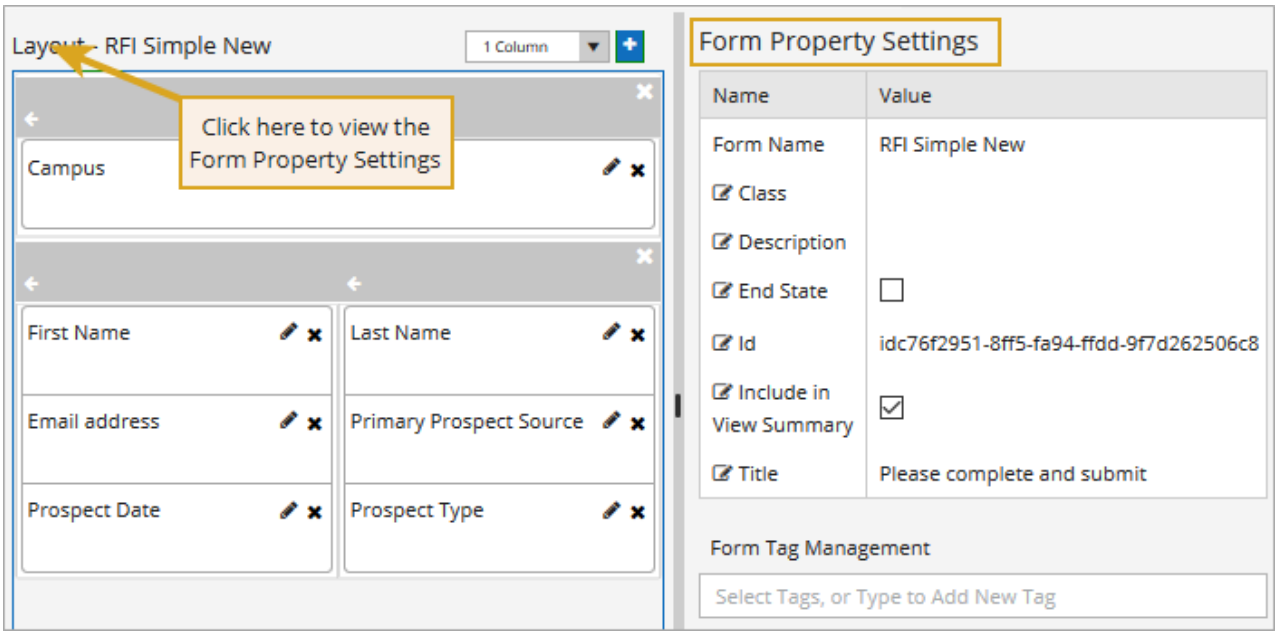
4. Select the **End State** (optional) check box if the form is used as a Confirmation form in a sequence. Refer to Welcome and Confirmation Forms.
5. Select **Save**.

The form is validated before it is saved. For more information, refer to Validation on Form Save.

Form Property Settings

You can assign additional form properties in the Form Property Settings pane which is displayed when:

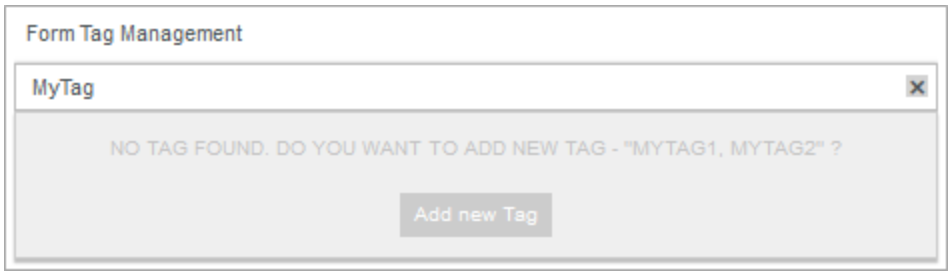
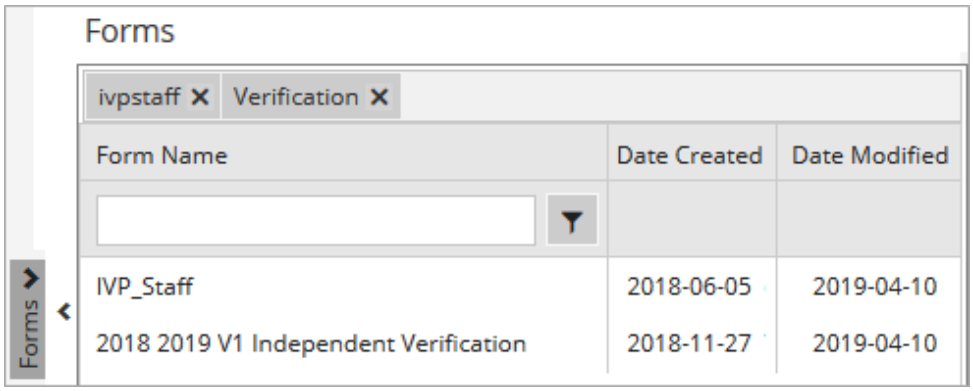
- A form is loaded from the Forms fly-out tab.
- A form title is selected above the Layout pane.



Form Property Settings

Property	Required	Description
Form Name	Yes	The Form Name will be the name of the State in a workflow. The Form Name is not editable after the initial save. Specify the value in the Property Settings pane or in the Unsaved Changes dialog. **Allowed special characters**: ! * ( ) - _ ~ . [ ] { } **Restricted special characters**: ` @ # $ % ^ & = < > ? : ; " ' +
Id	Yes	Globally unique identifier (GUID) for the form. It is automatically created by Forms Builder.

Property	Required	Description
Title	No	Displayed at the top of the form when the form is rendered. Specify the value in the Property Settings pane or in the Unsaved Changes dialog.
Class	No	CSS class (or space separated classes) specific to the form. The class must be defined in a Renderer CSS file. For more information, refer to Custom Styles.
Description	No	This is a convenience field to describe a form. Specify the value in the Property Settings pane or in the Unsaved Changes dialog.
End State	No	Set to false (cleared) by default. Specify the value in the Property Settings pane or in the Unsaved Changes dialog. If you select this check box, the form will be available in the list of End State forms for a sequence. Clearing the check box does not affect sequences where the form is defined as an End State form. This only adds or removes it from the End State form list in Sequence Designer. For more information, refer to Welcome and Confirmation Forms.
Include in View Summary	Yes	Set to true (selected) by default. If the View Summary component is present in a sequence, the form will be included in the forms rendered by the component. Clear this check box if you do not want to include this form in the View Summary. Forms Builder 3.4 and later provides enhanced validation when saving a sequence. A sequence cannot be saved if at least one form in the sequence has a View Summary component, but the Include in View Summary property is not selected on any of the forms in the sequence.

Property	Required	Description
Form Tag Management	No	Use this field to create a tag or associate an existing tag with the form. The assigned tags are available for selection in the Forms flyout and make it easier to locate forms. To create a tag, type the tag name in the Form Tag Management field, select **Add Tag**, and save the form.  To find a tagged form, on the Forms fly-out menu, select the **Select Form Tags to Search** field, and select a tag. The tag will be used to filter the list of forms. 

School Defined Fields

Your institution may need data that the generic version of Anthology Student has not provided. If so, you can define fields in Anthology Student, also known as School Defined Fields (SDFs), that are unique to your operations. Once SDFs have been created in Anthology Student, you can use Forms Builder and create forms that contain these fields.

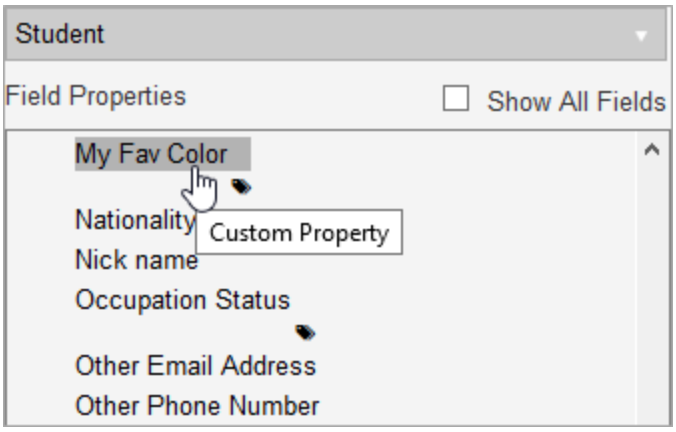
1. Create the SDFs in Anthology Student under **Setup > School Defined Fields > Student**. SDFs configured in Anthology Student are stored in the SyUserDict table.

   Forms Builder supports SDFs only for the StudentEntity.

   > ⚠ After creating new SDFs, restart the Anthology Student app service and then restart the Form Designer app service to regenerate the metadata to include the new SDFs.

2. In Form Designer, select **New** to enable the Layout pane.

3. In the Entities list on the Fields tab, select the **Student** entity. SDFs that are configured in the SyUserDict table will appear when the Student entity is selected. SDF properties are marked with a icon and display the text "Custom Property" when you hover over a field.



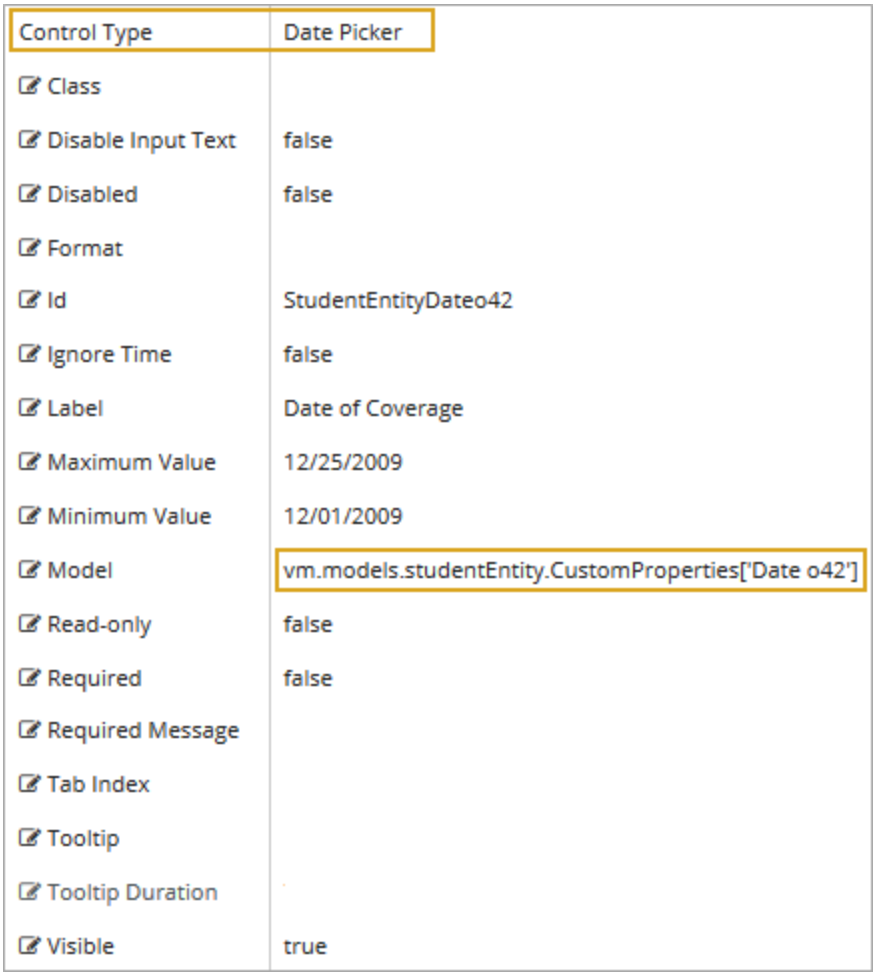
4. Select the SDF you want to use in your form and drag it into the **Layout** pane.

5. Select the SDF in the Layout pane and edit the values in the **Property Settings** pane.

   The binding for SDFs in the Model property uses the following format (refer to Text Box and Date Picker examples):

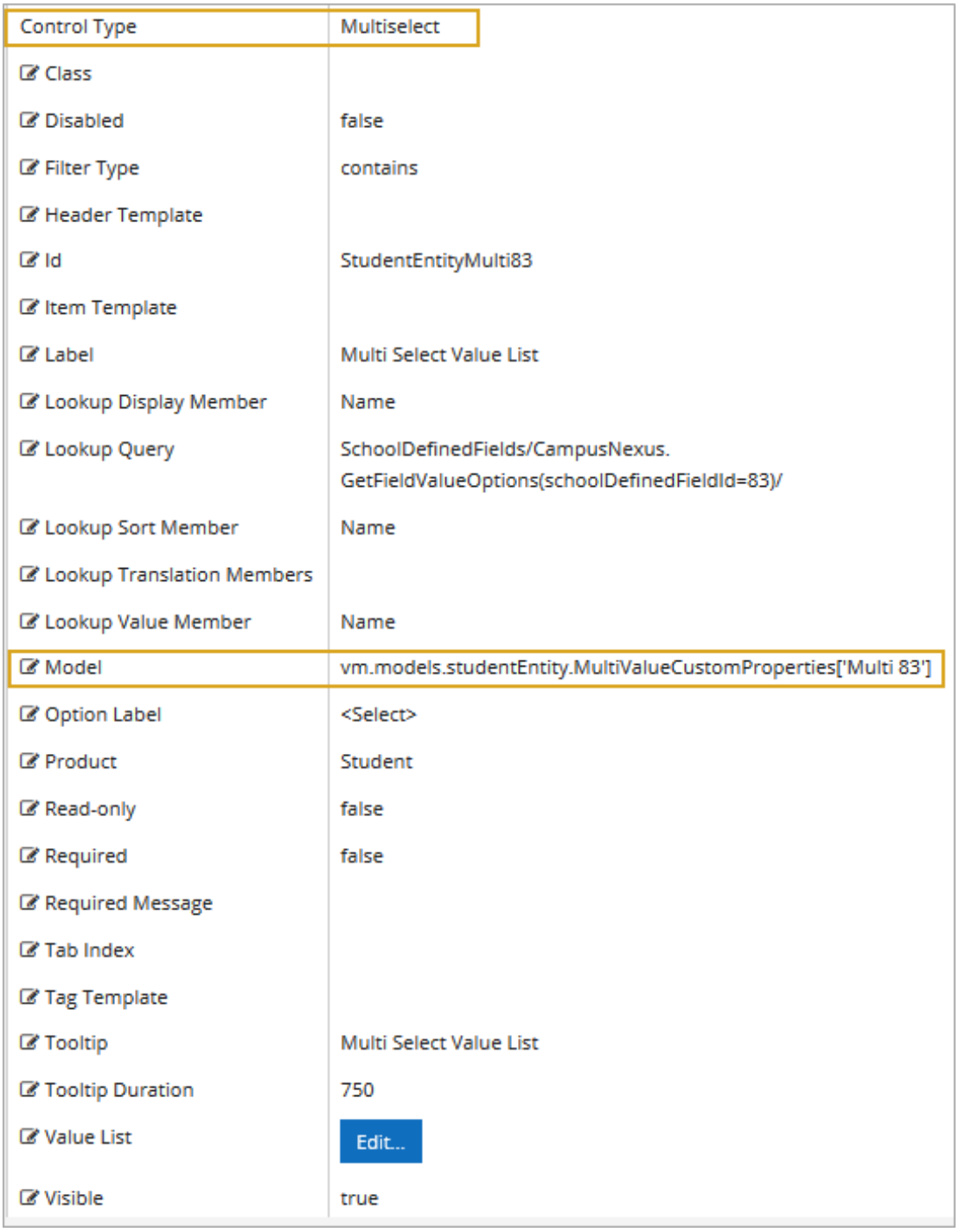
   ```
   vm.models.studentEntity.CustomProperties['propertyName']
   ```

Control Type	Text Box
Class	
Disabled	false
Id	StudentEntityMAXCH70
Label	MAX CHECK
Max Length	4000
Maximum Value	
Min Length	1
Minimum Value	
Model	vm.models.studentEntity.CustomProperties['MAX CH70']
Placeholder	
Read-only	false
Required	false
Required Message	
Tab Index	
Tooltip	MAX CHECK LENGTH ACCEPTED
Tooltip Duration	750
Type	text
Visible	true



The binding for multiselect SDFs in the Model property uses the following format (refer to Multiselect example):

```
vm.models.studentEntity.MultiValueCustomProperties['Multi 83']
```



6. **Save** the form.
7. Navigate to Sequence Designer, add the form to a sequence, and select **Save**.
8. In Workflow Composer, open and edit the workflow. For more information, refer to Open the Workflow for a Sequence.

To persist the data entered for an SDF in the rendered sequence, add a SaveEntity<StudentEntity> activity to the workflow.

SerializableDynamicObject

When a component in Forms Builder contains properties, which are not defined in a known type, the Model value needs to be bound to an argument of type SerializableDynamicObject[] which is an array of SerializableDynamicObject objects.

Example

Calendar/Scheduler Initialized by Model Data

What exactly is a serializable dynamic object array? Let's break it down.

Dynamic Objects

Dynamic objects expose properties at **run time**, instead of at compile time. This enables you to create objects with property names that are not previously defined.

For example, you may have data such as XML or JSON where the members aren't known ahead of time. The dynamic object type enables you to work with the objects and access their properties at run time.

Serializable Objects

An object can contain named properties with values that can themselves be objects or arrays. When an object is serializable, it can be converted to a string representation of the object. A string, as a sequence of characters, is easily transported across the Internet through web services. At its destination, it can be deserialized into the original object. The most efficient string representation of an object is a JSON string and is the format mainly used in Forms Builder.

Dictionary Objects

A Dictionary object is used to store information in name/value pairs (sometimes referred to as key and item value).

Since a dynamic object has properties whose names are not known until run time, a dictionary allows us to store the property name as a string (the key) and its value. Its value must be a known primitive type, like a string, integer, byte, etc., because these are easily deserialized. A SerializableDynamicObject has a Dictionary property called “DataDictionary”.

As an example, consider an object that has the following two properties with values.

“Property1” : “Value1”,

“Property2” : 234

Assuming an argument with name “mySerializableDynamicObject”, the values of the original object from the form web page can be accessed in a workflow as:

mySerializableDynamicObject.DataDictionary(“Property1”) => “Value1” (a string)

mySerializableDynamicObject.DataDictionary(“Property2”) => 234 (an Int32)

Values can also be set (with an Assign activity) and sent back to the form web page.

mySerializableDynamicObject.DataDictionary(“Property2”) = 987 (In this case an Int32 integer has been assigned.)

Validation on Form Save

To alert users about invalid property settings before a form is saved, Form Designerdisplays error ( ) and warning ( ) icons along with tooltips for invalid property settings. The indeterminate ( ) icon indicates that a property's validity could not be determined.

When a form with incorrect settings is saved, the errors/warnings are retained and displayed upon opening the form so that you can make the necessary corrections. However, not all items are validated such as formats and any properties in popups (e.g., Grid Columns).

If you try to save a form with errors, the font title above the Layout pane will turn red.

Boolean Properties

- All tooltips for Boolean properties indicate that the values "true" and "false" must be all lowercase.
- On many of these properties, a binding starting with "vm.models." is allowed.
- Where a binding is allowed, Forms Builder allows the input of AngularJS expressions.

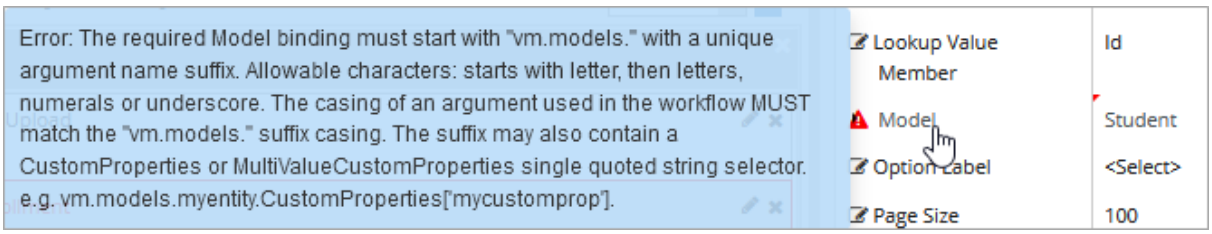
  For example, you can change the default value (true) of the Visible property on the SSN field to a condition so that the field is visible only if the condition evaluates to true:

  ```
  vm.models.StudentEntity.CountryId==7
  ```

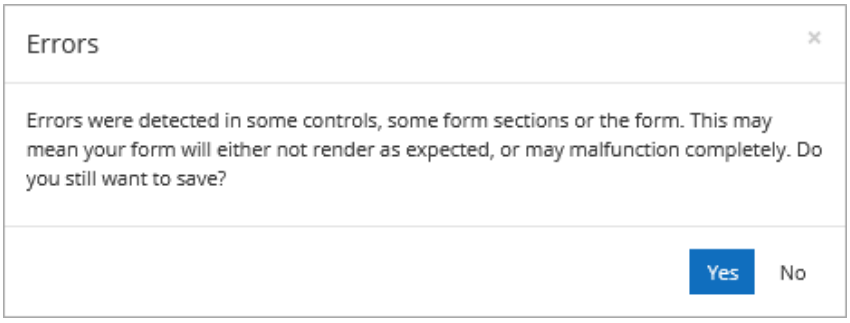
  Where the CountryId 7 represents United States, and Country is a field in the same form sequence.

Model Property

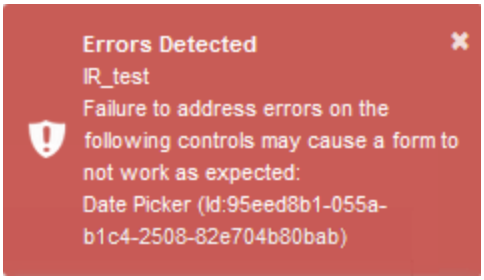
- All tooltips for the Model property include a reminder that the model argument needs to be defined in the workflow for custom controls.
- Forms Builder validates that the Model property is populated and starts with "vm.models.", for example vm.models.myArgument
- If the syntax for the Model value is incorrect, an error icon ( ) appears next to the Model property, and an error message is displayed:



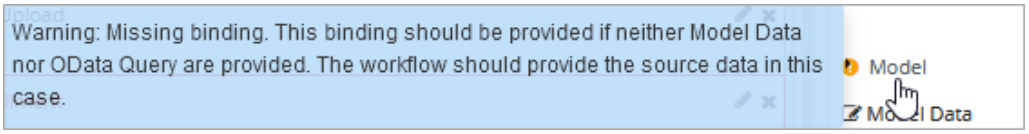
- If the Model value is required and is incorrect or missing, an error icon ( ) appears next to the Model property, and the following error message is displayed when the form is saved.



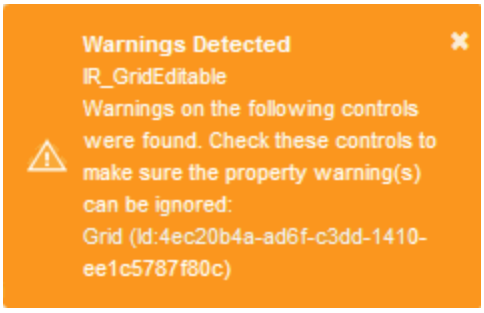
If you ignore the error and save the form, the next time the form is opened, a message similar to the following appears:



- If the Model value is optional and is incorrect or missing, a warning icon ( ) appears next to the Model property and a warning message is displayed.



If you ignore the warning and save the form, the next time the form is opened, a message similar to the following appears:



Validation Errors for School Defined Fields

In forms created with Forms Builder 3.2 or earlier, School Defined Fields that contain spaces in the field name (e.g., "Date of Coverage") will show validation errors for the Id property. These errors can be ignored since they do not impact the functionality of the forms (invalid JavaScript).

To clear the errors, simply:

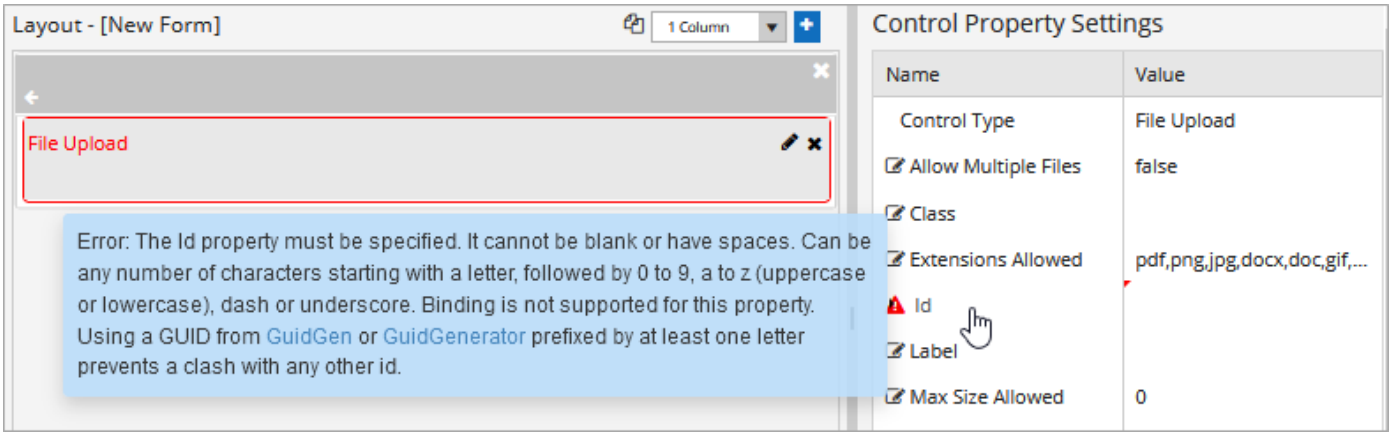
- Re-drag the school defined field into Layout pane.

  — OR —

- In the Property Settings pane, delete the spaces on the Id value (e.g., "DateofCoverage").

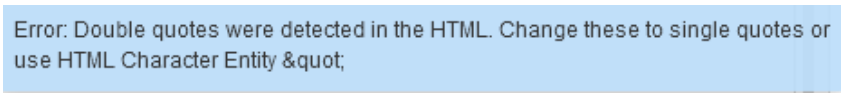
Validation Error for Id Property on File Upload

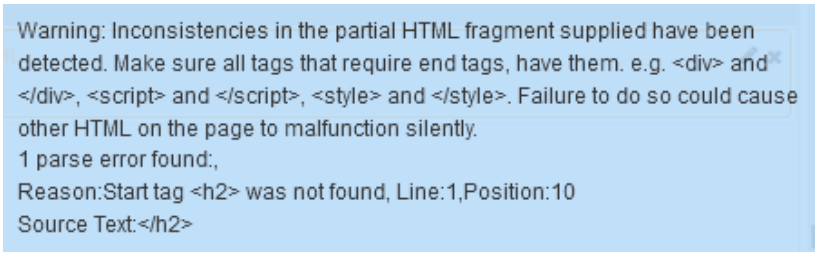
If a File Upload control was added in an earlier version of Forms Builder, a warning will be displayed on the Id property because the Id must start with a letter. To clear the warning, add a letter to the beginning of the Id value.



HTML Syntax Checking

Forms Builder will perform as much HTML syntax checking as possible. Correct syntax errors or warnings as indicated by the tooltips.





Indeterminate Flags

Form Designer displays icons along with tooltips for indeterminate property settings. OData query properties such as Lookup Display Member, Lookup Query, Lookup Sort Member, and Lookup Value Member will display this icon if:

- There is no main select statement.
- An expand, filter, or orderby is found before a select statement.
- A property with dot notation is present, and each part is somewhere in the query so it did not produce a warning.

Queries should always be tested externally in browsers (we recommend Chrome or Edge) to determine if they actually will produce results (with bound Model values, if any, substituted with real values).

Internationalization

With the enhancements for internationalization and localization, you can use Forms Builder to create form sequences in multiple languages.

Definitions

Internationalization (or globalization) refers to the design and development of a culture-neutral and language-neutral product that enables easy localization for target audiences in all parts of the world.

Internationalization requires:

- Enabling the code to support language, local, or cultural preferences (e.g., use of Unicode internally, support for right-to-left and vertical text direction, time zone awareness, local date and time formats, local calendars, local number formats, and so on).
- Separating localizable content from the source code so that localized alternatives can be loaded based on the user's selection.

Internationalization is often written as i18n, where 18 is the number of letters between i and n.

Localization refers to the adaptation of a product to meet the language, cultural and other requirements of a specific target market (a locale). A **locale** is a collection of specific numeric, date and time formats, currency, symbols, and other requirements.

The localization process involves translating text and selecting locale-specific components. Translatable text, including labels and validation messages, is extracted from the source code, translated, and rendered based on the user's language/culture choice.

Localization is sometimes written as l10n, where 10 is the number of letters between l and n.

Internationalization and Localization in Forms Builder

The **Gettext** library of tools was added to the Forms Builder 3.5 code. The Gettext utilities cover all phases of the translation process:

a. Gettext scans the code and places the translatable text strings (case sensitive) in a **.pot** (portable object template) file. The .pot file has all the translation strings (the `msgstr` parts) left empty, for example:

```
msgid "Hello world"
msgstr ""
```

b. You use the .pot file to create one **.po**(portable object) file per locale (e.g., fr.po, de.po, es.po). The .po files have all the translation strings (the `msgstr` parts) filled in, for example:

```
msgid "Hello world"
msgstr "Bonjour le monde"
```

In components that are shared with other CampusNexus products (core UI components), an attribute that marks translatable text was added to properties that can contain text, such as Label, Required Message, Selection Text, Tooltip, etc. The Gettext tools extract the marked text strings.

All custom Forms Builder and Kendo components that have properties with translatable text run the **`angular-gettext`** tools. The text to be translated is marked with a `translate` directive. The Gettext tools extract the marked text strings.

In addition to form text, the .pot file includes non-form text so that text strings for the following items can be translated using .po files:

- Button labels:
  - Next/Back
  - View Summary/Hide Summary (View Summary page)
  - Submit (internal page displayed after the DocuSign process)
  - Show Filter Row and Select & Cancel (in Single-select Search components)

  If button labels are changed in workflows, you will have to manually add the translations to the .pot file, unless the labels are already in the file and translated.

- Version information
- Form Titles
- Form Section Titles
- Messages generated by Forms Builder Settings
- Error messages from Forms Builder (Error messages from CampusNexus CRM or Anthology Student are not captured in the .pot file.)
- Error messages, warnings, and notifications triggered by form validations
- Values of Custom Value Lists in Drop-down and Multiselect components
- Model Data in Grid and Calendar/Scheduler components
- Grid Column validations

Credit Card Payment forms using PayPal, ACI, IATS, or TouchNet payment gateways will not be translated as these vendors do not support localization at this time.

Culture Scripts for Kendo Components

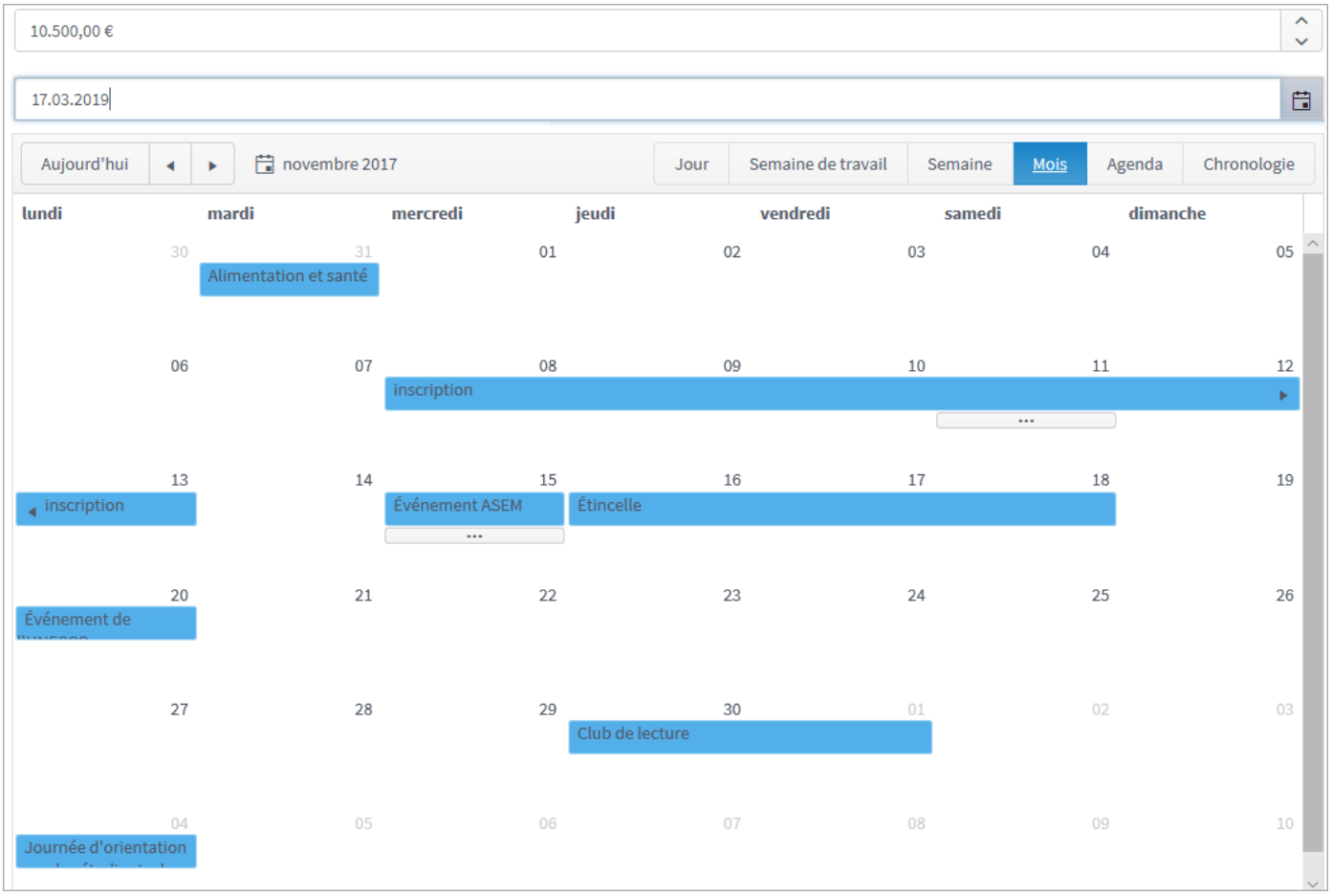
Forms Builder uses a set of Kendo components that support localization around formats of dates and currency. The Kendo components in Forms Builder are:

- Calendar/Scheduler
- Date Picker
- Date Time Picker
- Masked Text Box (globalized mask literals)
- Numeric Text Box
- Time Picker

When a Locale component is included on the first form of a sequence, all Kendo components within the sequence will run the culture scripts to apply the setting selected in the Locale component.

Example:

A Locale component offering a drop-down list with multiple languages, including French, is placed on the welcome form of a sequence. The user selects French and proceeds to a form that includes a Numeric Text Box with Euro currency symbol, a Date Picker, and a Calendar/Scheduler.



For more information, refer to https://docs.telerik.com/kendo-ui/framework/globalization/overview.

The translation of labels and validation messages in Kendo controls is handled by the Gettext tools.

Localization of DocuSign Forms

Users can set the display language for DocuSign from the selection menu at the bottom of the DocuSign Iframe page.



Steps to Localize Sequences

1. Add the Locale component to the first form in a sequence. Use this component to set up the language and culture selections supported in a sequence.
2. Re-save all forms that were created in Forms Builder 3.4 and earlier.

   In forms created prior to Forms Builder 3.5 with File Upload component, the File Upload button will not be translated. If the component is replaced with the new File Upload component created in 3.5, the button is translatable.

   Any additional tasks depend on the types of components used in the sequence.

   a. For HTML components, apply the **`translate`** attribute to the text you want to localize. The attribute can be added to any HTML element. For example:

      ```
      <div translate>Translate me!</div>
      ```

      If the HTML component has a <p> or element (or any other valid HTML element) that contains text **and** a link, enclose the text in a element and add the `translate` attribute to the . Do not include non-alphabetic character at the end of sentences (e.g., !, ..., ?) into the span with translate tag. For example:

      ```
      <p style="color:green; word-wrap:break-word;"><span translate>For more info, please visit</span>: <a href=" http://anthology.com ">Campus Tech Website</a></p>
      ```

      Do not use the **&** symbol within HTML text because it will not be properly pulled into .pot file or translated. Replace & with the word "and".

   b. If the optional Header Template property is used for Multiselect components, apply the **`translate`** attribute to the Header Template property. For example, to translate the Header Template property value `<div><b>Degrees</b></div>`, specify:

      ```
      <div><b translate>Degrees</b></div>
      ```

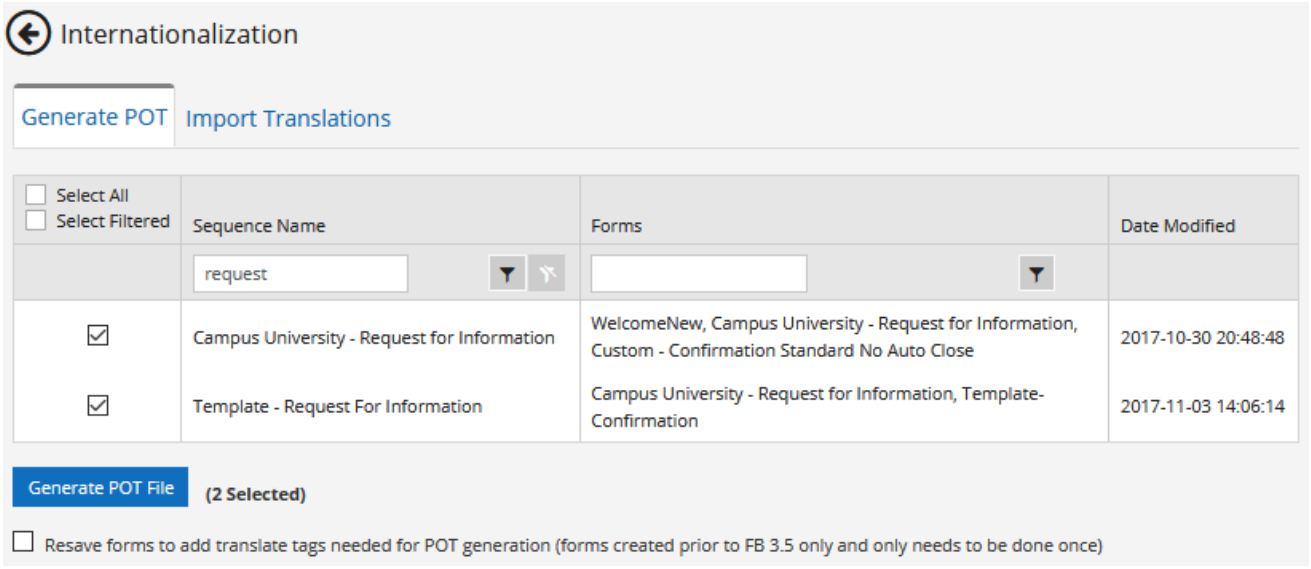
   c. If your sequences include text that originates in a workflow (e.g., custom validation items), perform the steps needed to add TranslateText activities.

   d. If your sequences include Drop-down List, Multiselect, Single-select Search, or Calendar/Scheduler components, use the **Lookup Translation Members** or **OData Translation Members** (on Calendar/Scheduler) property to extract translatable text from an OData query (e.g., names of table columns). This property is a comma separated list of items for a Select statement in an OData query. The query results will be included in the POT file generation.

   Avoid using text in your forms that can be interpreted as an HTML tag by a browser (i.e., text wrapped in < and >, for example, <Select>). This text may not be translated properly in the current version (2.4.1) of angular-gettext with Internet Explorer and Microsoft Edge browsers.

3. Go to the Forms Builder home page and select the **Internationalization** tile.

The Generate POT tab is displayed by default. It contains a grid listing sequence names, forms contained with the sequences, and date modified.



4. Use one of the following options to select sequences. The number of selected sequences will be displayed next to the Generate POT File button.

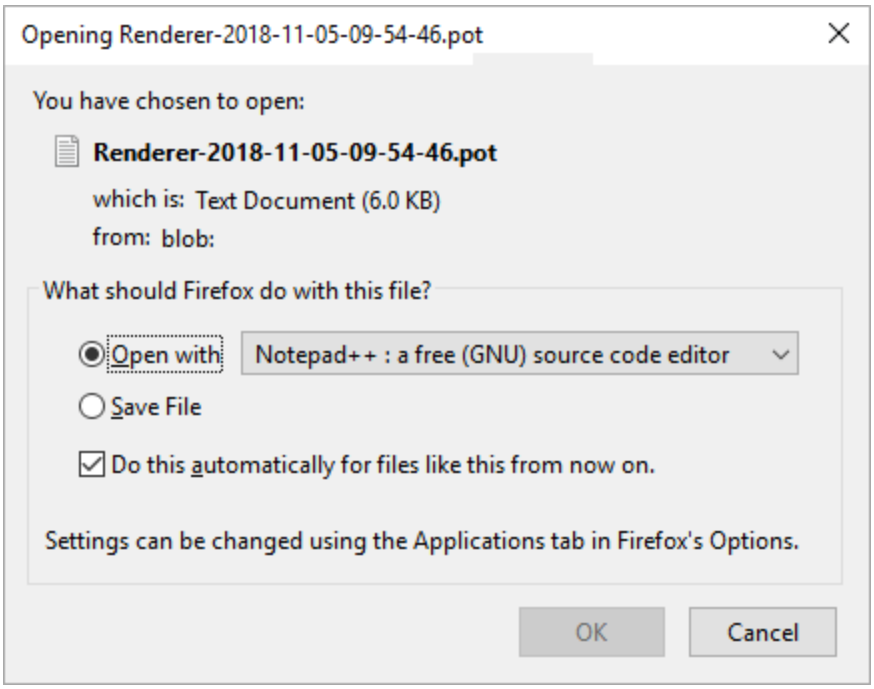
    - Select **Select All**.
    - Use the column filters to narrow your search and select **Select Filtered**.
    - Clear "Select All" and "Select Filtered" and select **individual sequences**.

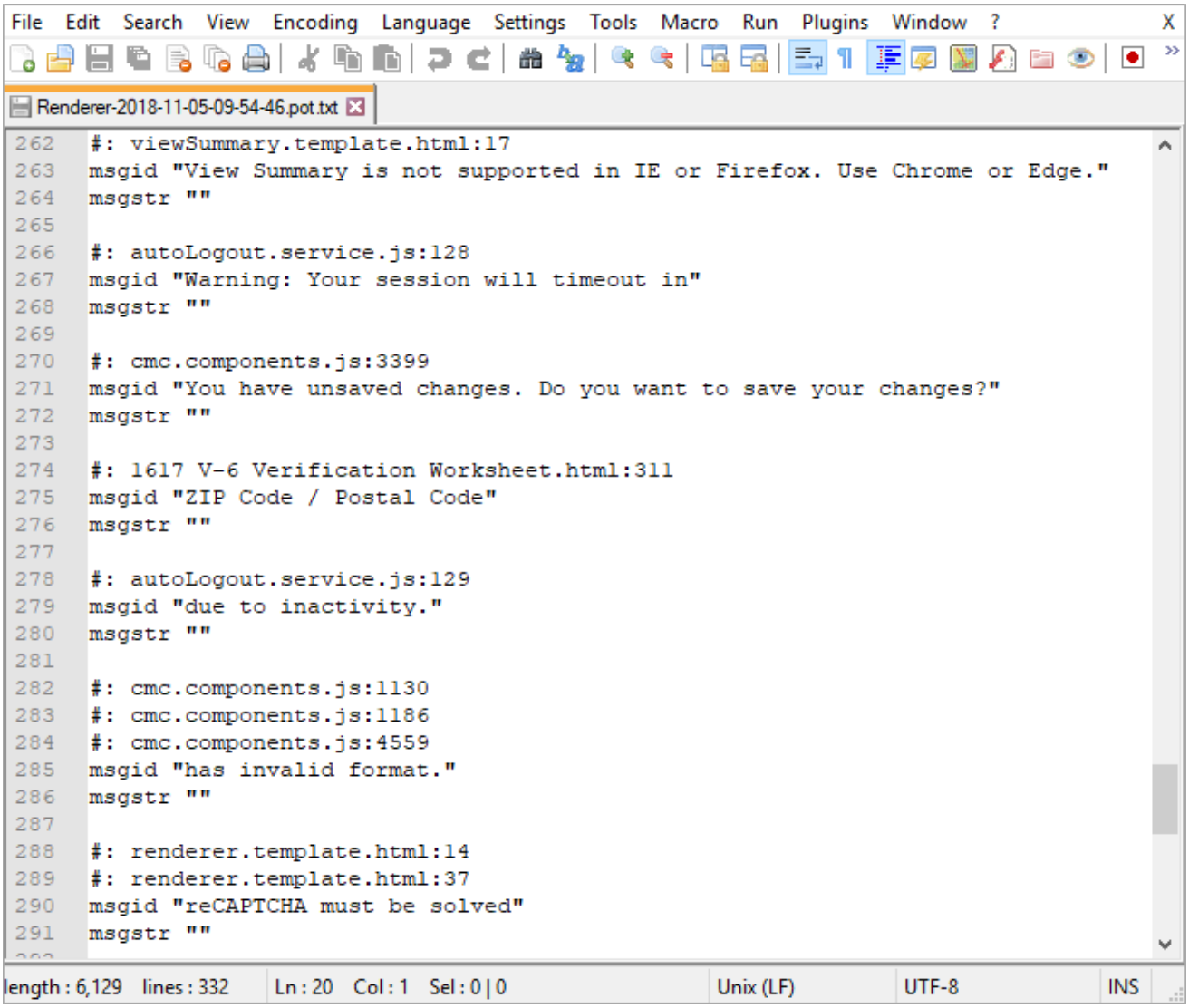
    Most often you would Select All sequences to extract all text in all forms. The text will be translated outside of Forms Builder and stored in .po files for the supported languages. If sequences are added or updated, select all sequences again. The generated .pot file will be updated to include the changes. The .po files will retain all previously translated text and just need to be updated to include the changes.

    In cases where an institution uses different forms for different locales/campuses, you might want to generate .pot files with a subset of sequences.

5. If you selected forms that were created prior to Forms Builder 3.5, select the check box below the Generate POT File button to **resave the forms**. This only needs to be done once to add the translate tags needed for the POT generation to the older forms.

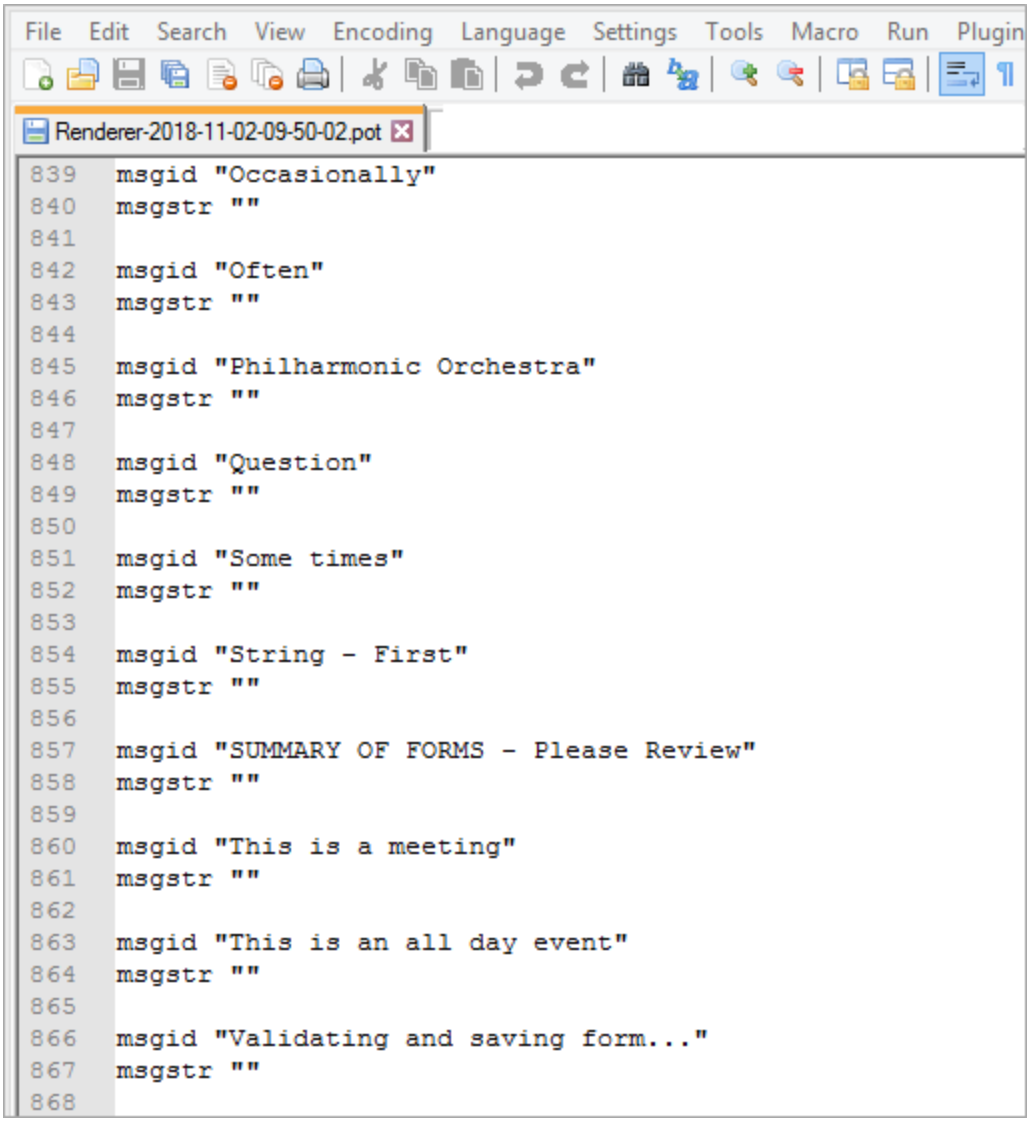
6. Select **Generate POT File**. When processing is complete, a message with the name of the .pot file is displayed. We recommend using **Notepad++** to open the file.


Opening Renderer-2018-11-05-09-54-46.pot
You have chosen to open:
Renderer-2018-11-05-09-54-46.pot
which is: Text Document (6.0 KB)
from: blob:
What should Firefox do with this file?
Open with
Notepad++ : a free (GNU) source code editor
Save File
Do this automatically for files like this from now on.
Settings can be changed using the Applications tab in Firefox's Options.
OK
Cancel



The .pot file will contain the names of forms where the text was found, the line numbers, and the `msgids` with the text strings to be translated. If the same text appears multiple times in different forms, the .pot file will capture a single `msgid` so that only one text string needs to be translated.

If your sequences contain components with **Lookup Query** properties, manually edit the .pot file in Notepad++. For the selection list values, add text strings enclosed in quotes to the msgid and msgstr fields. To obtain the values in the selection lists, render the sequences or run the OData queries directly in a browser. Be sure to include any trailing spaces on the text strings in the .pot file. Strings have to match exactly for translations. After adding the strings to the .pot file, update the .po files and import them into Forms Builder.



The Lookup Query property is found in the following components:

- Calendar/Scheduler
- Drop-down List
- Grid with Query on a Column
- Multiselect
- Single-select Search
- Typeahead

The Typeahead component does not have a Lookup Value Member property. Therefore, any values in the Typeahead selection list that are translated have to be saved to the database in the original language.

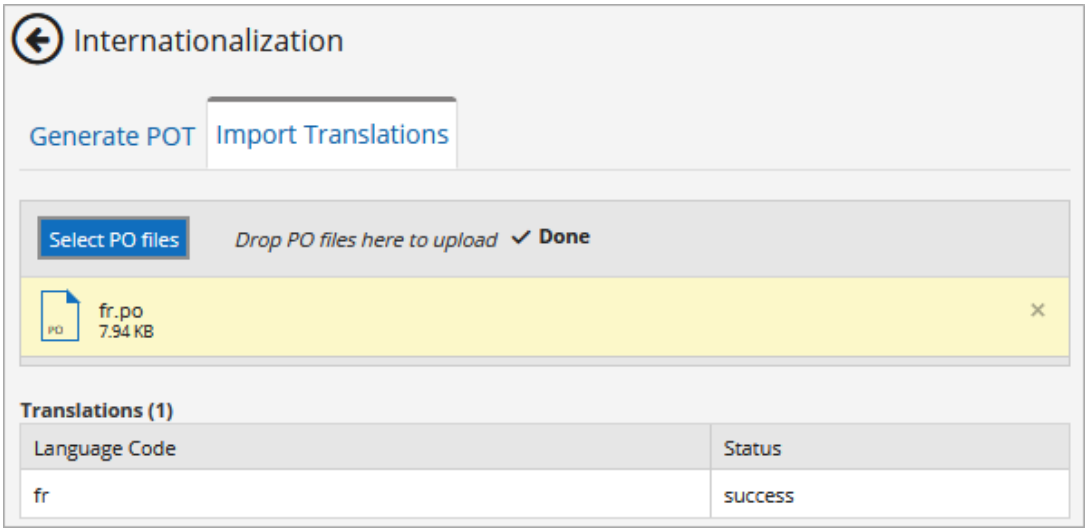
7. Outside of Forms Builder, use a translation tool such as *Poedit* (refer to https://poedit.net/) to create and save .po files for each supported locale.

The .pot generation is case sensitive. For example, if the text strings include the words "campus" and "Campus", you will need translations for both words. If only "Campus" is translated in the .po file and a page renders with "campus", the page will indicate that a translation was not found.

You can review the .pot files using a tool like NotePad ++. However, we do not recommend that you manually update the .pot files.

Do not change the file names of the .po files because the locale is part of name and has to be in a specific format. Only 1 .po file per locale is allowed.

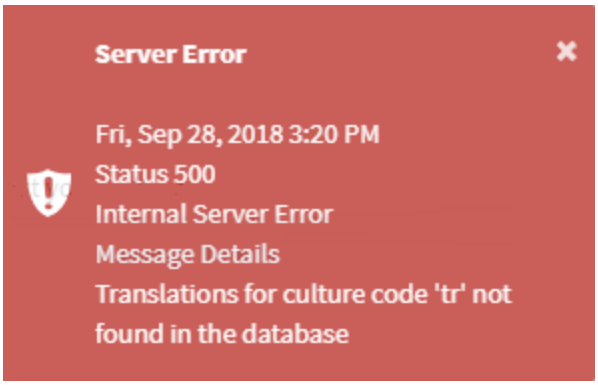
8. In Forms Builder, select the **Internationalization** tile and select the **Import Translations** tab.
9. Select **Select PO** files and navigate your .po file(s). The imported file(s) will be listed on the Import Translations tab. The Language Code column indicates the locale(s). The Status column indicates the import success/failure.



The imported translations are stored in the *formsbuilder.translation* database table. The table will contain all translations found in the .po files. Text strings without translations will not be imported.

10. After completing the POT Generation and Import Translations steps, you can view rendered sequences with translated text based on the locale selection.

    If POT Generation and Import Translations have not been completed for a locale, a warning message is displayed, and the translatable text is displayed in the default language set in the Locale component (if used).



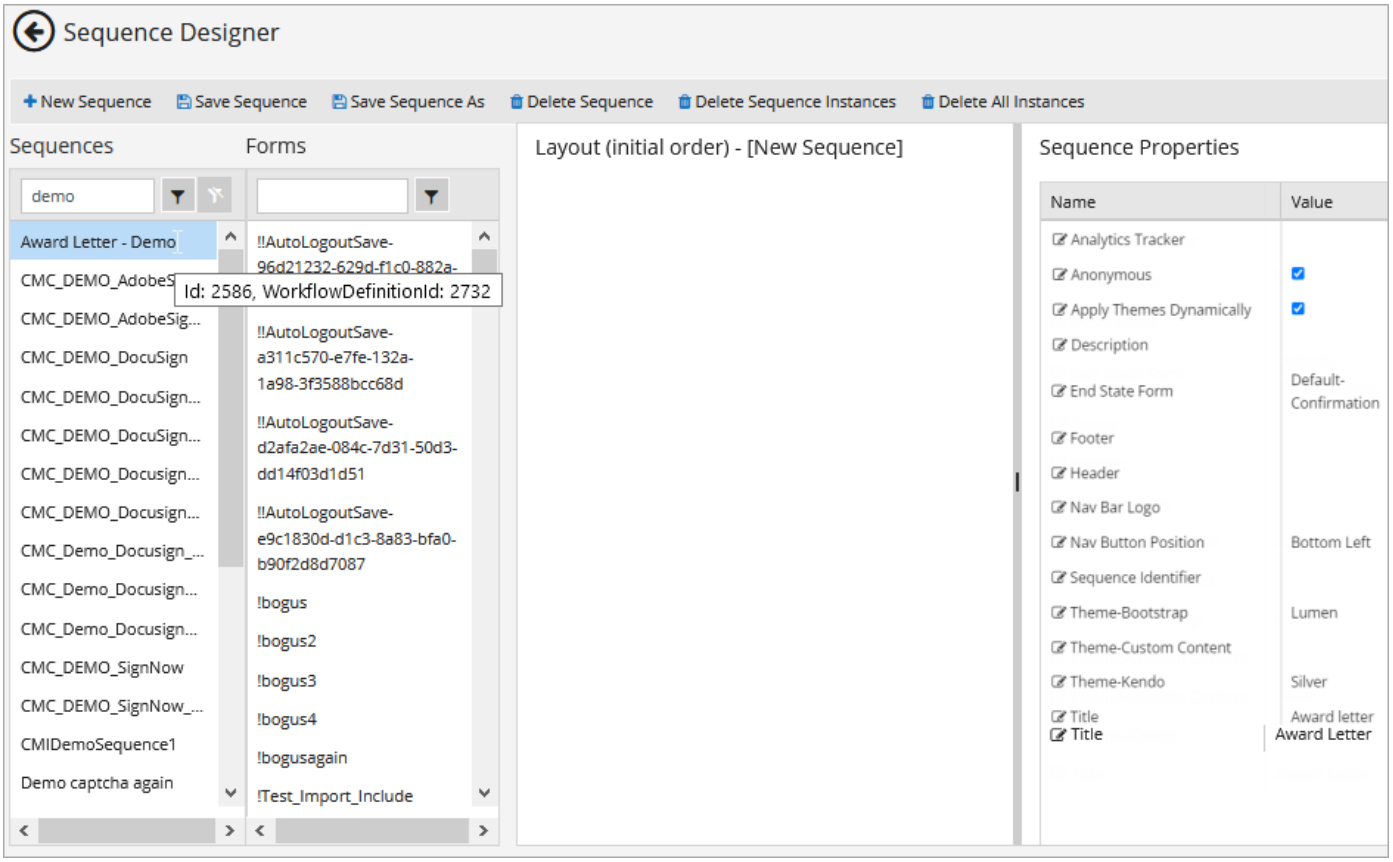
You can enable the Debug Translations option in the Settings workspace to help troubleshoot translated sequences. When this option is selected, text processed by the translation engine on rendered forms will be wrapped with '[ ]' markers and untranslated text will be prefixed with `[MISSING]:`.

The Login Locale setting can be used to add a drop-down list for locales on the Azure AD login page. For more information, refer to Login Locales.

If the "Create Account" option is selected on the Azure AD login page, the user is directed to the "New Account Creation" page in Portal. The header bar in Portal has a "Choose language" drop-down that is not linked to the Login Locales setting.

Sequence Designer

The Sequence Designer workspace is displayed when you select the Sequence Designer tile in the home page. This workspace enables you to create and edit form sequences.



Sequence Designer UI Elements

Element	Description
	Return to the Forms Builder home page.
New Sequence	Create a new sequence and clear the Layout pane. On the initial entry to the page, the Layout pane is disabled until New Sequence is selected.
Save Sequence	Save a sequence. This button is enabled only if a sequence is selected. For more information, refer to Save Sequences.
Save Sequence As	Save a sequence with a new name. This button is enabled only if a sequence is selected. For more information, refer to Save Sequences.
Delete Sequence	Delete a sequence. This button is enabled only if a sequence is selected. For more information, refer to Delete Sequences.

Element	Description
Delete Sequence Instances	Delete all persisted workflow instances of a sequence. This button is enabled only if a sequence is selected. For more information, refer to Delete Sequence Instances.
Delete All Instances	Delete all persisted workflow instances of all sequences. For more information, refer to Delete All Instances.
Sequences	
	Use the search tool to find a saved sequence or scroll through the list. Select to filter the list. When you select a sequence, the properties pane displays the metadata of the sequence. When you hover over a sequence name, the sequence Id and workflow definition Id are displayed.
Forms	
	Use the search tool to find a saved form or scroll through the list. Select to filter the list. Drag forms into the Layout pane to build a sequence. When you hover over a form name, the form Id is displayed.
Layout (initial order)	
Use the Layout pane to assemble and organize forms into a sequence. Drag forms from the Forms pane into the Layout pane. To change the order of forms in a sequence, drag forms within the Layout pane. Sequence Designer does not allow you to change the forms and their order once the sequence is saved. However, you can change the order of forms and transitions between them in Workflow Composer.	
✖	Remove a form from the Layout pane.
Sequence Properties	
When you select a sequence, the metadata of the sequence are displayed in the Sequence Properties pane. Except for the Name property, you can edit and save any sequence properties.	
Analytics Tracker	If you have purchased form analytics from Formisimo, specify the Zuko slug obtained from Formisimo. The slug is a text string that identifies the form in the Zuko platform. For more information, refer to Analytics Tracker.
Anonymous	When a sequence is rendered, authentication is required or not required based on the Anonymous setting for the sequence. The default is: Anonymous = false (check box is cleared, editable), i.e., by default the sequence will be authenticated.
Apply Themes Dynamically	Enables administrators to design form sequences to be displayed at different URLs with different branding while using a single instance of Forms Builder (Designer and Renderer). For more information, refer to Multiple Renderer URLs. You cannot save a sequence with selections for both **Apply Themes Dynamically** and **Theme-Custom Content**.

Element	Description
Authentication Product	When Forms Builder is installed in an environment that uses both CampusNexus CRM and Anthology Student as database providers, the Authentication Product field is displayed. You must select the Authentication Product (CRM or Student) before saving an authenticated sequence. The Authentication Product field is not displayed: • If only one product database is available • When the Anonymous field is selected and authentication is not applicable The default Authentication Product value in the database will be CRM if both CRM and Student databases are available.
Auto Logout when Complete	Select this property if you want the user to be automatically logged out when the sequence is completed. If selected, the logout is delayed by the number of seconds specified under Auto Logout Delay in Settings. The value must be 0 or greater and there is no max. The default is 10.
Description	Description of the sequence (optional, editable).
End State Form	The End State Form must be selected before saving a sequence, i.e., there is no default. For more information, refer to Welcome and Confirmation Forms.
Footer	Specify an arbitrary HTML fragment for a footer, including scripts and styles that will appear on all forms in the sequence. For more information, refer to Footer and Header.
Header	Specify an arbitrary HTML fragment for a header, including scripts and styles that will appear on all forms in the sequence. For more information, refer to Footer and Header.
Nav Bar Logo	Use the drop-down list to associate a custom logo with a sequence. The image file must have been uploaded to the Custom Content workspace. For more information, refer to Navigation Bar Logos.
Nav Button Position	Select the position of the navigation buttons (Back/Next) on the rendered page. For more information, refer to Nav Button Position.
Role	Select the role of the user who will be completing the sequence. The options are Student (default) and Staff. The Student role is also used for CampusNexus CRM Contacts. For more information, refer to Role.
Sequence Identifier	Identifier for a Renderer URL (optional, editable). For more information, refer to Sequence Identifier.

Element	Description
Theme-Bootstrap	• Bootstrap themes are used to style layouts, containers, and general UI elements. Select from the default themes that are available out-of-the-box. • Theme-Custom Content is used to select a CSS file that was previously uploaded to the CustomContent table in the database. Refer to Custom Content. • Kendo themes are used to style Kendo controls (e.g., buttons, check boxes). Select from the default themes that are available out-of-the-box. For more information, refer to Themes.
Theme-Custom Content	
Theme-Kendo	
Title	Specify the title of the sequence (optional, editable).

In Forms Builder 3.5.2 and later, the 'Open Workflow' button is no longer available. It enabled users to launch Workflow Composer directly from Sequence Designer. Users can launch Workflow Composer directly from the machine or site on which it's installed. For more information, refer to Open the Workflow for a Sequence.

Analytics Tracker

Forms Builder 3.7 and later supports form analytics via 3rd party scripts that are inserted at the sequence level. Anthology Inc. partnered with Formisimo to leverage the Zuko platform for form analytics. Forms Builder clients who request form statistics and usage data are referred to Formisimo to purchase the Zuko product. Zuko is a SaaS platform.

Zuko tracks form events, visualizes data, enables clients to configure alerts, and helps designers optimize forms. The Zuko platform provides a dashboard and multiple views where clients can review, for example:

- Number of form visits, starts, and completions
- Average form completion time
- User interactions with specific fields
- User navigation paths
- Field metric trends over a period of time
- Field return data
- Fields that caused users to abandon the form

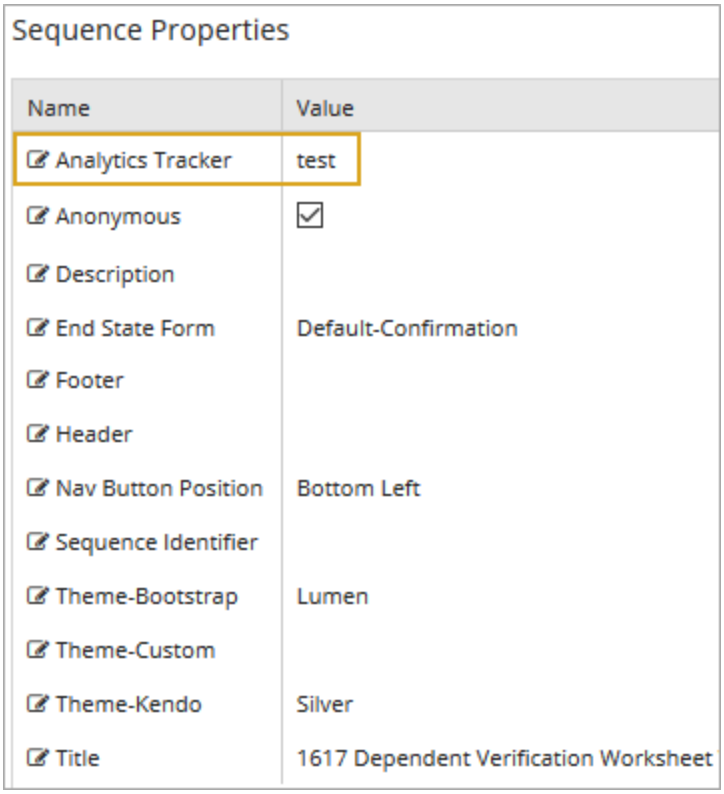
For more information, refer to https://www.zuko.io/formisimo-and-anthology-inc.

To implement the Zuko analytics tracker in your form sequences, log into at https://support.anthology.com/ and create a new partner request. For detailed instructions, refer to Analytics Tracker - Quick Reference.

Enable Form Analytics

If you have purchased a Zuko license, specify the slug provided by Formisimo in the Analytics Tracker property of the form sequence. The slug is a text string that identifies the form in the Zuko platform. Form events sent to the Zuko platform will include the slug and domain from where the event originates.

In the example below, the slug is "test".



Map Forms Builder Fields to Zuko

Refer to https://www.zuko.io/formisimo-and-anthology-inc for details about mapping your Forms Builder fields to the Zuko platform.

The quick reference guide for field mapping is also available Analytics Tracker - Quick Reference.

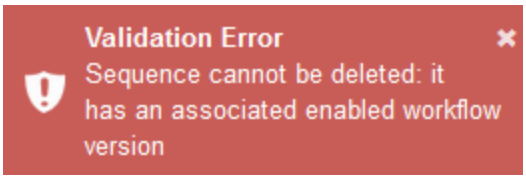
Delete Sequences

You can delete a sequence. When the delete action is executed, the sequence, the corresponding workflow definition, and all associated workflow versions will be deleted. The forms referenced in the sequence will not be deleted.

1. In Sequence Designer, select the sequence you want to delete.
2. Select **Delete** in the action bar. The following message is displayed: *"Are you sure you want to delete <sequence name>?"* Select **OK**.
3. If the sequence is **not** associated with enabled workflow versions or an entry in the durable instancing table, the sequence is deleted. A confirmation message is displayed.

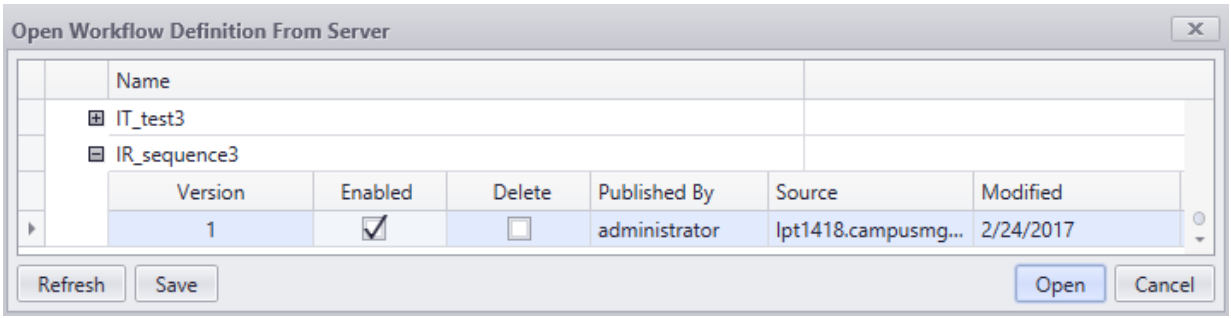


4. If the sequence is associated with enabled workflow versions, an error is displayed.



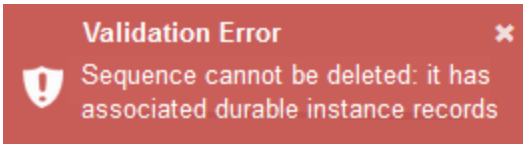
To delete a sequence that has an enabled workflow, perform the following steps:

a. In Workflow Composer, select **Open from Server** and locate the sequence.



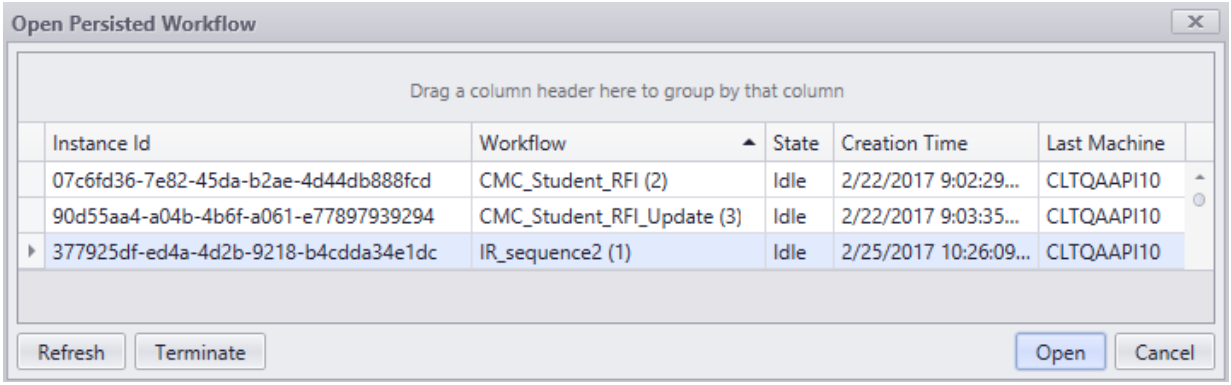
b. Clear the **Enabled** check box and select **Save**.

5. If the sequence cannot be deleted because it is associated with durable instance records, an error is displayed.



To delete a sequence associated with durable instance records, perform steps a and b or refer to Delete Sequence Instances.

a. In Workflow Composer, select **Open Persisted Workflow** and locate your sequence.



b. Select the workflow instance associated with the sequence to be deleted, select **Terminate**, and select **Yes**.

6. After you have disabled the workflow and/or terminated the persisted workflow, return to Sequence Designer, select the sequence again, select **Delete**, and select **OK**.

   A confirmation message is displayed.



Delete Persisted Workflow Instances

In Forms Builder 3.6 and later, you can delete persisted workflow instances of sequences from the Sequence Designer workspace instead of having to use Workflow Composer to accomplish this. Sequence Designer provides two options:

- **Delete Sequence Instances** removes all persisted workflow instances of a specific sequence
- **Delete All Instances**removes all persisted workflow instances of all sequences.

Delete Sequence Instances

1. Select a sequence in Sequence Designer.
2. Select **Delete Sequence Instances**. The following message is displayed:

   Are you sure you want to delete all persisted instances for sequence <sequence name>? This will abort any workflows in progress for this sequence.

3. Select **OK** to proceed. All persisted workflow instances of the selected sequence are deleted.

   The sequence itself is still available in Sequence Designer.

4. To remove the sequence from Form Designer, select the sequence and select **Delete Sequence**. The following message is displayed:

   Are you sure you want to delete sequence <sequence name>? This will abort any workflows in progress for this sequence.

5. Select **OK** to proceed.

   This deletes persisted instances, disables, and deletes the workflows, and removes the sequence from Sequence Designer.

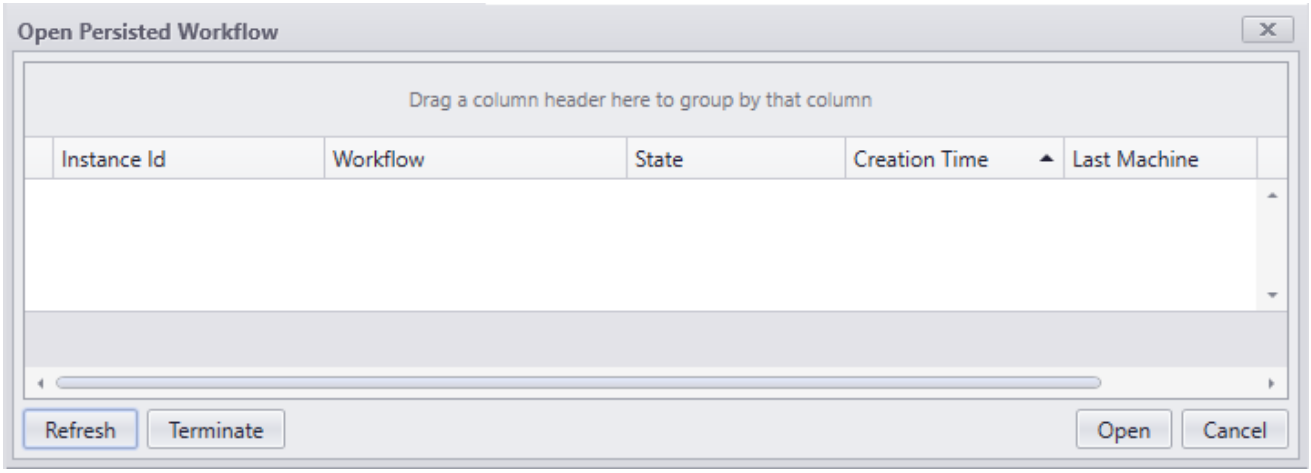
Delete All Instances

1. Select **Delete All Instances** in Sequence Designer. The following message is displayed:

   Are you sure you want to delete all persisted instances for all sequences? This will abort any workflows in progress.

2. Select **OK** to proceed. All workflow instances of all sequences are deleted.

   You can confirm this in Workflow Composer. The persisted workflow grid is cleared.



   The sequences are still available in Sequence Designer.

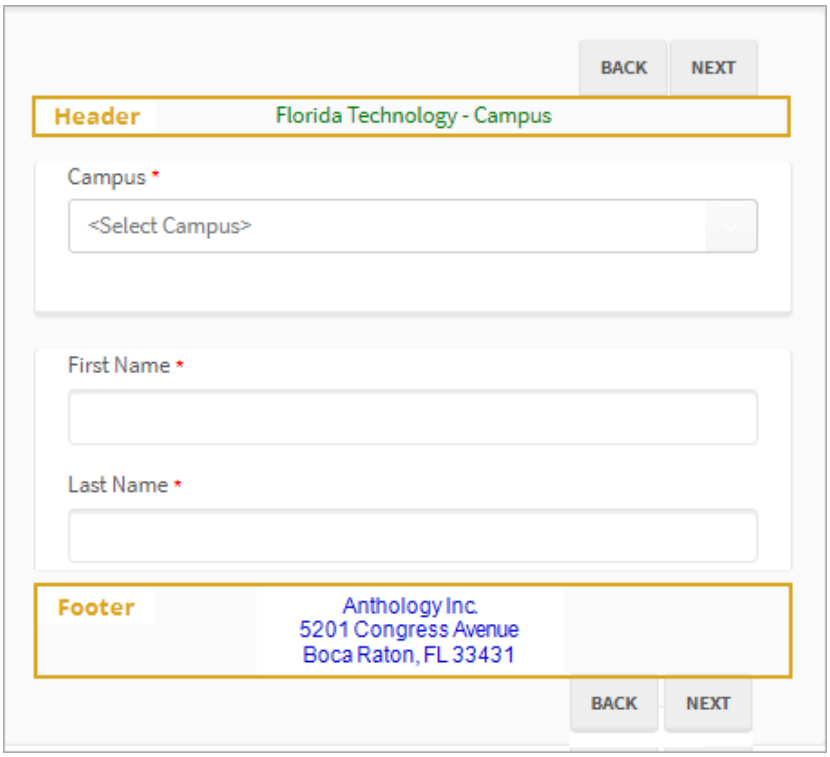
Footer and Header

In the Sequence Properties pane of Sequence Designer, you can use the Footer and Header properties to specify content, styles, and scripts for the bottom and top sections of all forms in a sequence.

Sequence Properties

Name	Value
Analytics Tracker	
Anonymous	☑
Description	
End State Form	Default-Confirmation
Footer	<span translate> Anthology Inc. 5201 Congress Ave Boca Raton FL 33431
Header	<span translate>Florida Technology - Campus
Nav Bar Logo	
Nav Button Position	Top and Bottom Right
Sequence Identifier	header
Theme-Bootstrap	Lumen
Theme-Custom	
Theme-Kendo	Silver
Title	KL3.6 Header and Footer

Rendered Sequence



Footer

The value for the Footer property is an arbitrary HTML fragment, including scripts and styles. Observe security guidelines for active content. If not set, no space is used.

Note that <HTML>, <HEAD>, <BODY> or <FORM> tags should not be put in an HTML fragment. Normally, you would use <DIV> tags to start and end the fragment.

Example 1

```
<div style="text-align:center"><p style="color:blue; word-wrap:break-word;"><span translate>Anthology Inc.<br>5201 Congress Ave <br>Boca Raton, FL 33431</span></div>
```



Example 2

```
<div style="text-align:right"><span translate>Page {{vm.models.progress}}</span></div>
```

This footer has a dynamic binding. The page numbers get updated for every page in a sequence.



Footers will appear on all forms in the sequence. They will be located below the form content and above navigation buttons in bottom position. For more information, refer to Nav Button Position.

Header

The value for the Header property is an arbitrary HTML fragment, including scripts and styles. Observe security guidelines for active content. If not set, no space is used.

Note that <HTML>, <HEAD>, <BODY> or <FORM> tags should not be put in an HTML fragment. Normally, use <DIV> tags to start and end the fragment.

Example

```
<div style="text-align:center"><p style="color:green; word-wrap:break-word;"><span translate>Florida Technology - Campus</span></div>
```

Florida Technology - Campus

Headers will appear on all forms in the sequence. They will be located above the form content and below navigation buttons in top position. For more information, refer to Nav Button Position.

Nav Button Position

In the Sequence Properties pane of Sequence Designer, you can select the position of the navigation buttons (Back/Next) on the rendered page.

The position options are:

- Bottom Left (default)
- Top Left
- Bottom Right
- Top Right
- Top and Bottom Left
- Top and Bottom Right
- Float Bottom Left
- Float Top Left
- Float Bottom Right
- Float Top Right

Float positions stay in the same position as the page is scrolled. Normal positions are part of the layout and scroll with the page.

The default CSS for the button(s) position is shown below. You can create a custom style with new values:

```
<style>
  .cmc-form-navigation-buttons {
    margin-top: 0;
  }
```

```
    .cmc-form-navigation-buttons > input {
      padding-top: 9px;
    }
  </style>
```

Notes

- When the top right position is used, error messages will hide the buttons until the message is closed. To change this behavior, you can add the following CSS style in a non-visible HTML component on the first page of the sequence where error messages can be shown (or globally, refer to 3 ways to style sequences in Custom Styles). The defaults are shown here.

  ```
  <style>
    .toast-top-right{
      top: 12px;
      right: 12px;
    }
  </style>
  ```

  You can change where the error message pops up by changing the values. For example, to move it 50px down, change the top to 62px.

- When float positions are used, validation error messages will appear above the floating navigation buttons.

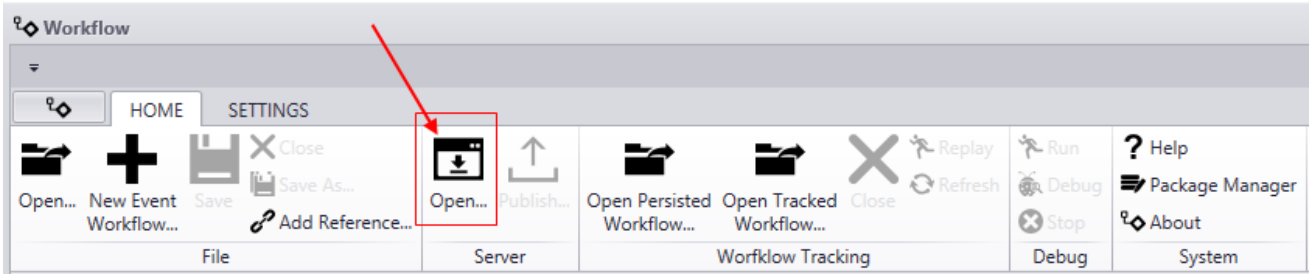
Open the Workflow for a Sequence

Perform the following steps to open the workflow for a sequence:

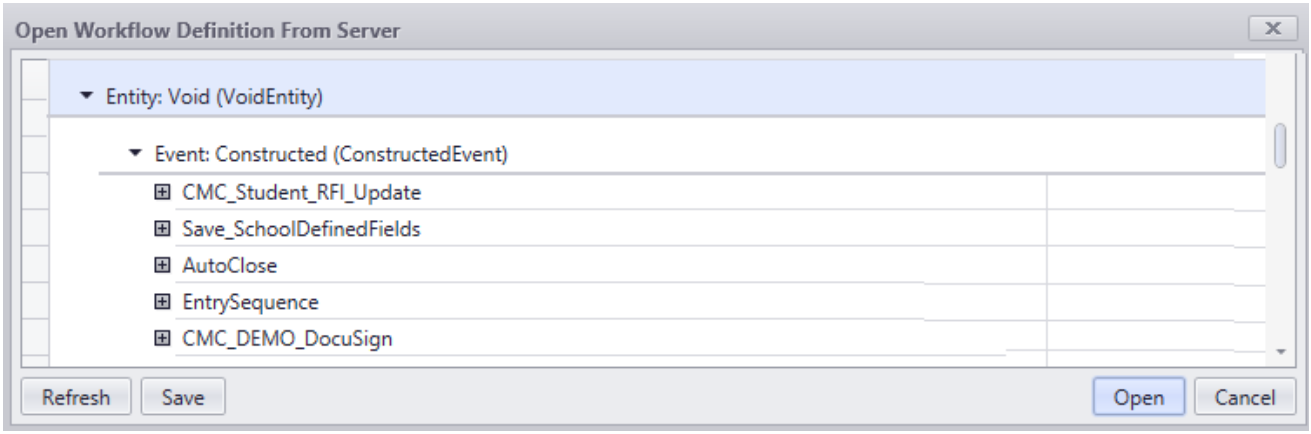
1. Create and save a sequence in Sequence Designer. Note the name of the sequence.
2. Depending your environment:
   - Launch your local installation of Workflow Composer.

     — OR —

   - Access your instance of the hosted Workflow Composer and launch the application.
3. In Workflow Composer, select **Open** in the Server section of the ribbon.



4. In the *Open Workflow Definition From Server* window, select the **Name** filter above the grid to sort the workflows alphabetically, scroll down to the *Entity Void* section, and locate your sequence.

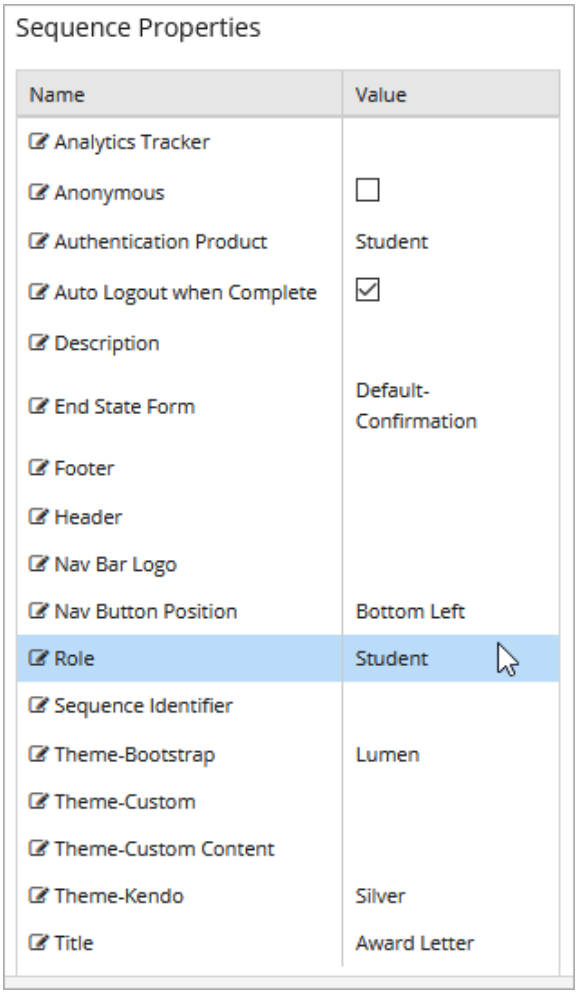


5. Expand the sequence, select a workflow definition version, and select **Open**. The initial workflow definition for the sequence is displayed.
6. Edit, publish, enable, and save the workflow in Workflow Composer.
7. In Forms Renderer, reload the webpage for the sequence and verify that the changes made in the workflow are displayed as expected.

Role

In the Sequence Properties pane of Sequence Designer, you can select the role of the user who will be completing the sequence. The options are **Student** (default) and **Staff**. The Student role is also used for CampusNexus CRM Contacts.

Forms Builder recognizes the current user's role and only allows the user to execute sequences that have a matching role, i.e., if the user is a Student, the user will not be allowed to execute sequences that have a role of Staff.



Role Configuration

The Student and Staff roles are configured during the installation of Forms Builder 3.5 and later in the web.config files of Forms Renderer and Staff STS.

The web.config file of CMCFormsRenderer_V3 provides authentication and mapping of Staff and Student roles to products:

```
<section name="authenticationConfigSection" type-
e="Cmc.Nexus.FormsBuilder.Helpers.AuthenticationConfigSection, Cmc.Nexus.FormsBuilder" />

<!-- Mapping of realms to issuers -->
<mappings>
  <!-- <mapping realmKeys="Comma separated realm URL keys or * for wildcard match"
        product="Student, CRM or * for wildcard match"
        role="Student or Staff"
        issuerKey="URL key of the issuer" /> -->

  <mapping realmKeys="*" product="Student" role="Student" issuerKey="Student STS"/>
  <mapping realmKeys="*" product="CRM" role="Student" issuerKey="CRM STS"/>
  <mapping realmKeys="*" product="*" role="Staff" issuerKey="Staff STS"/>
</mappings>
```

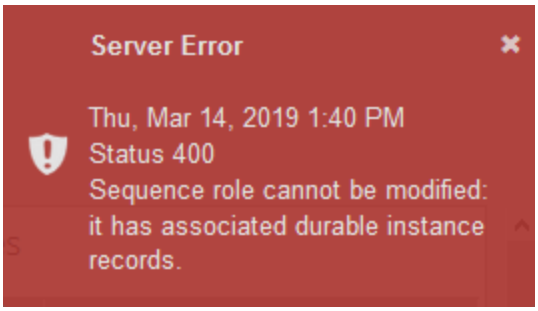
The web.config file of Staff STS uses the following key under `<appSettings>` to accept claims from Renderer:

```
<add key="FormsBuilder.Renderer.WsFed" value="http://<server>:<port>/" />
```

If you are using Forms Builder 3.5 with Anthology Student 19.0 or earlier, add the `FormsBuilder.Renderer.WsFed` key manually to the web.config for the Staff STS.

Notes

- An update script sets the value of the Role property to Student/Contact for all sequences created in Forms Builder 3.4 and earlier.
- If a staff sequence is accessed via cloud services (Azure), you must include a LookupUser activity with UserType=Staff in the workflow to ensure proper authentication and authorization for the Staff role. For more information, refer to Azure AD Authentication.
- When the Anonymous field is selected for a sequence, the role property is not applicable and the Role field is hidden. The default Role value in the database will be Student.
- When the Role value in an existing sequence is modified and a persisted instance of the workflow exists, a Save/Update of the sequence is not allowed. Forms Builder displays the following error:



Save Sequences

In Forms Builder individual forms are grouped into sequences and each sequence is saved. End users view the saved sequences.

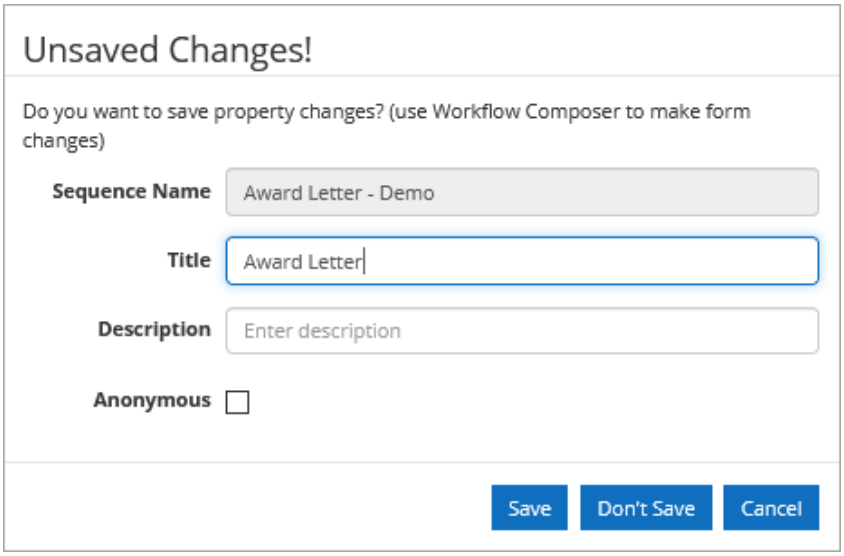
To create and save a sequence:

1. Access the Sequence Designer workspace.
2. Select the **New Sequence** button.
3. Drag one or more **Forms** into the Layout pane. Position the forms in the order you want them to appear in the sequence.

   Note that the first and last form in a sequence have special significance. For more information, refer to Welcome and Confirmation Forms.
4. Select or specify values in the **Sequence Properties** pane. For more information, refer to Sequence Properties.
5. Select **Save Sequence**. The Unsaved Changes dialog is displayed.

Unsaved Changes Dialog

The Unsaved Changes window prompts you to specify the Sequence Name, Title, Description, and Anonymous property. All four attributes are displayed in the Sequence List and can be used to search or filter sequences.



1. Specify a **Sequence Name** (required).
2. Specify the **Title** (optional) .
3. Specify a **Description** (optional) that helps you identify the sequence.

The Form Name, Title, and Description are displayed in the Sequence List and can be used as search and filter criteria.

4. Select the **Anonymous**(optional) check box as needed.

   The Anonymous property setting determines if a user will be authenticated before accessing the sequence in Renderer (refer to Renderer Authentication)

   - If Anonymous is 'true', the user will not be authenticated.
   - If Anonymous is 'false' (default), the user will be authenticated.

5. Select **Save**.

When a sequence is saved, the settings selected in the Sequence Properties pane are saved along with the settings specified in the Unsaved Changes window.

The Save operation also creates a **workflow definition** for the sequence. The initial workflow is enabled by default. Once a sequence is saved, all subsequent edits needed for the flow of the sequence must be done by editing the workflow in Workflow Composer. Refer to Open the Workflow for a Sequence.

Save Sequence As

Select this button to save a sequence with a new name. This button is enabled only if a sequence is selected.

When the 'Save As' option is chosen, the workflow definition version that is used from the sequence being copied will be the version that is enabled. If no version is enabled, the most recent version will be used.

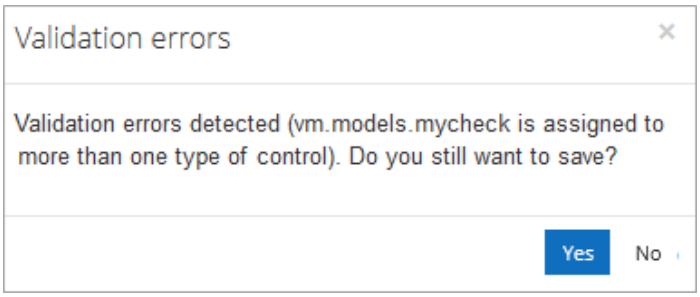
The 'Save As' option provides a means to easily create a copy of a complicated workflow definition such as the multi-signer e-signature functionality. The copied workflow definition can be modified using Workflow Composer, for example, to modify the state machine names so they align with the forms that need to be rendered for the sequence.

Validation Checks

Sequence Designer performs validation on Save and displays messages if the **Authentication Product** or **End State Form** are not selected.

In Forms Builder 3.6 and later, Sequence Designer performs additional validation checks and displays a validation error or warning.

- Sequence Designer checks the **Model bindings** on all forms in the sequence. A validation error indicates duplicate that bindings are found. The validation error gives you the option to proceed or cancel the Save operation.



- On **initial** Save for any sequence containing a form with a **Grid** or **Calendar/Scheduler** component, the following warning is displayed just prior to the Save Sequence window:

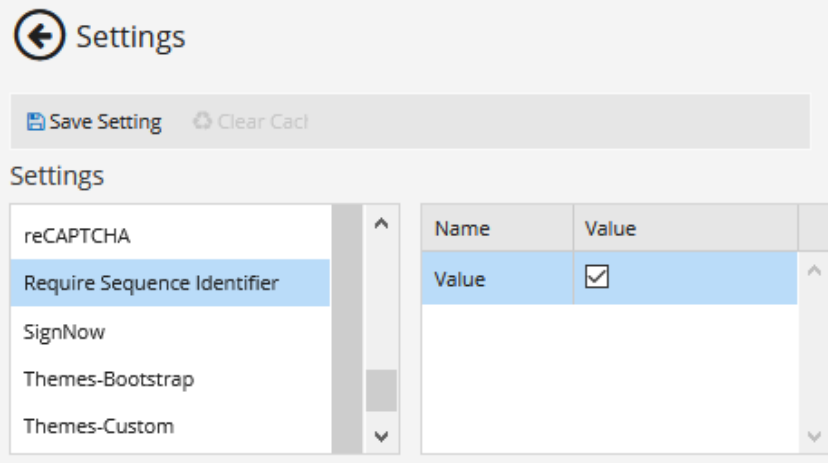
  WARNING! Unable to determine type for initial argument creation for some Calendar and/or Grid components. You must open workflow and update the argument type for these components after saving the sequence.

  This warning appears only for OData query or workflow initialized Grid or Calendar/Scheduler components bound to entities that Designer cannot determine on create. If either component has Model Data, the correct array of type SerializableDynamicObject is created and no message is displayed.

Sequence Identifier

The Sequence Identifier property enables you to specify the URL that will be used by students to access a Forms Builder sequence. This property defines an identifier for a Renderer URL. It is appended to the sequence URL (http:/mymachine:myport/#/renderer/) instead of the numeric identifier (WorkflowDefinitionId) generated by Forms Builder.

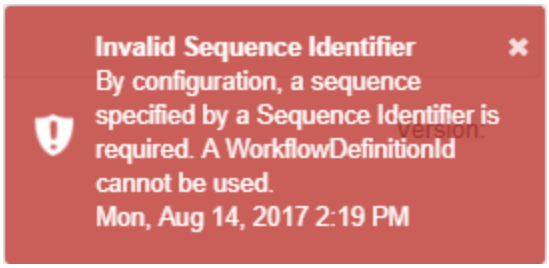
The Forms Builder Settings pane provides an option that controls whether the Sequence Identifier is required for all sequences.



- When "Require Sequence Identifier" is selected (true), sequences ...

  ... with Sequence Identifier can be accessed only using the Sequence Identifier. The WorkflowDefinitionId cannot be used.

  ... without Sequence Identifier cannot be accessed. The following message is displayed in Forms Renderer.

- When "Require Sequence Identifier" is not selected (false), sequences...

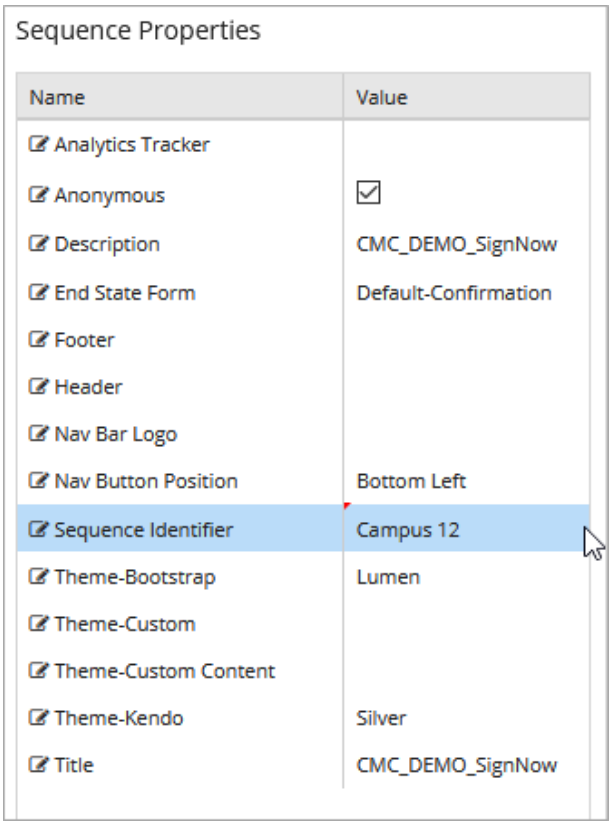
  ... with Sequence Identifier can be accessed using the Sequence Identifier or the WorkflowDefinitionId.

  ... without Sequence Identifier can be accessed using the WorkflowDefinitionId.

The `Formsbuilder.Sequence` table stores the sequence properties including WorkflowDefinitionId and Sequence Identifier (Url).

Assign a Sequence Identifier to a Sequence

1. In Sequence Designer, select the sequence to which you want to assign a Sequence Identifier.
2. In the Properties pane of the sequence, specify a **Sequence Identifier** and save the sequence.



The Sequence Identifier value...

... must be unique (refer to Create a Unique Sequence Identifier).

... is not case sensitive.

... can be any name valid in a URL. (Characters like spaces will be encoded on save, e.g., Campus+12+RFI.)

... must not be a number (which will be interpreted as a WorkflowDefinitionId).

3. In the Sequence List, use the copy icon to retrieve the final URL.
4. Paste the final URL into a browser window or select to access the sequence. Note that the Sequence Identifier value is appended to the Renderer URL, e.g.,

   http://<server>.<domain>:<port>/#/renderer/Campus+12+RFI

Create a Unique Sequence Identifier

To ensure that your Sequence Identifiers are unique and hard to guess, you could use GUIDs (128-bit integer numbers) generated by tools such as http://guidgen.com or http://guidgenerator.com.

Themes

A theme controls the presentation of content in a website. A theme is a collection of files that work together to control how content is displayed. The files in a theme can include style sheets (.css), images, fonts, color schemes, scripts, templates, and text files. The name of a theme maps to a .css file.

Forms Builder pre-populates a set of Bootstrap and Kendo themes.

- **Bootstrap** themes are primarily used for layouts, containers for elements, and general webpage elements.

  The following Bootstrap themes are available out-of-the-box:

  Bootstrap Themes

Common Name	File Name
Cerulean	bootstrap.cerulean.min.css
Cosmo	bootstrap.cosmo.min.css
Cyborg	bootstrap.cyborg.min.css
Darkly	bootstrap.darkly.min.css
Default	bootstrap.default.min.css
Flatly	bootstrap.flatly.min.css
Journal	bootstrap.journal.min.css
Lumen	bootstrap.lumen.min.css
Paper	bootstrap.paper.min.css
Readable	bootstrap.readable.min.css
Sandstone	bootstrap.sandstone.min.css
Simplex	bootstrap.simplex.min.css
Slate	bootstrap.slate.min.css
Spacelab	bootstrap.spacelab.min.css
Superhero	bootstrap.superhero.min.css
United	bootstrap.united.min.css
Yeti	bootstrap.yeti.min.css

- **Kendo** themes are used for Kendo controls (e.g., buttons, check boxes). Kendo themes override Bootstrap elements that are used in rendering Kendo controls. Kendo themes do not just override colors and fonts, they also override borders, spacing, padding, and margins to make the controls look better.

  The following Kendo themes are available out-of-the-box:

Kendo Themes

Common Name	File Name
Black	kendo.black.min.css
BlueOpal	kendo.blueopal.min.css
Bootstrap	kendo.bootstrap.min.css
Default	kendo.default.min.css
Fiori	kendo.fiori.min.css
Flat	kendo.flat.min.css
HighContrast	kendo.highcontrast.min.css
Material	kendo.material.min.css
MaterialBlack	kendo.materialblack.min.css
Metro	kendo.metro.min.css
Metro Black	kendo.metroblack.min.css
Moonlight	kendo.moonlight.min.css
Nova	kendo.nova.min.css
Office365	kendo.office365.min.css
Silver	kendo.silver.min.css
Uniform	kendo.uniform.min.css

For more information, refer to Kendo UI ThemeBuilder.

> Both Bootstrap and Kendo themes apply CSS based on media. That is, the size, type, and host operating system of the browser that is used. Thus, a complex interaction results where one may apply a different CSS set for a browser size, type, or host OS, while the other does not switch to different CSS.

Themes can be associated with individual sequences within the property settings in Sequence Designer.

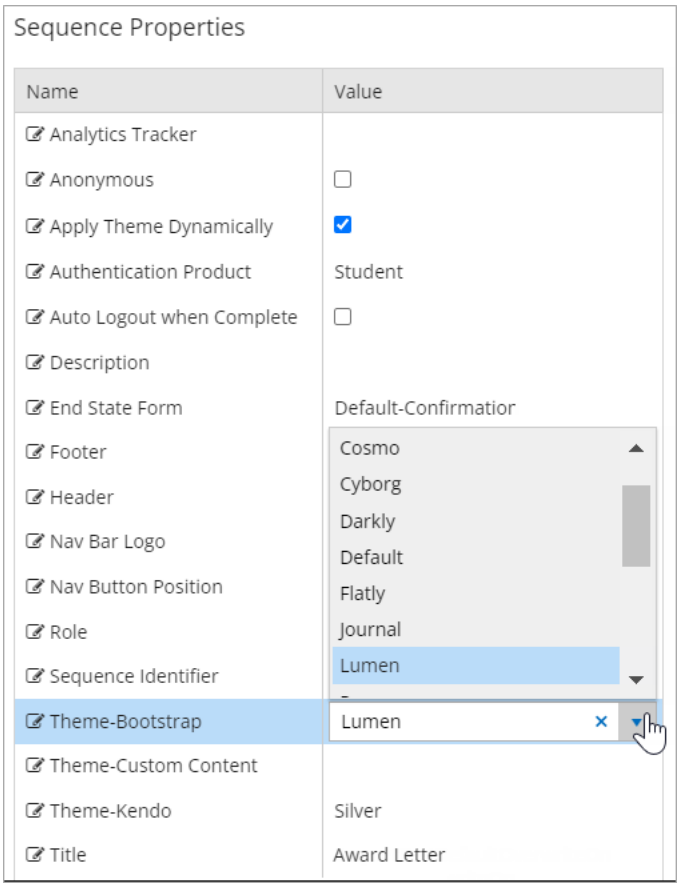
Note:

- Custom bootstrap and custom Kendo themes selected in version 4.0 (or in prior versions) in the Sequence Designer will no longer be available. The following default values will be set for such sequences:
    - Theme-Bootstrap: Lumen
    - Theme-Kendo: Silver
- Customizations requires all custom files to be uploaded to the CustomContent data table. Refer to Custom Content.

You can customize the style of controls using the Class property. For more information, refer to Custom Styles.

Apply a Theme to a Sequence

1. Select the **Sequence Designer** tile on the home page of Form Designer.
2. Select a sequence in the Sequences pane.
3. In the Properties pane:
    - Select the **Name** of a theme group, e.g., Theme-Bootstrap, Theme-Custom Content, or Theme-Kendo.
    - Select the **Value** field and select a theme from the drop-down list.



4. **Save** the sequence.
5. Review the rendered sequence. If necessary, refresh the browser cache by pressing **Ctrl+F5**. Note the changes in the content presentation based on the selected theme.

    To remove a previously selected theme from a sequence, select the blank option in the Value field.

Welcome and Confirmation Forms

The first form in a form sequence is often designed as a **Welcome** form. It contains only one navigation button (Next) and is the first state in a workflow. When it is added to a form sequence, the Form Names column in the Sequence List includes the Welcome form.

The last form in a sequence is the end state in a workflow. It is the form chosen as the **End State Form** in the Sequence Properties pane in Sequence Designer. You must choose an End State Form (no default). In Forms Builder 3.5 and later, the names of forms with 'End State' property appear in the Form Names column of the Sequence List. Previously, they were not listed.

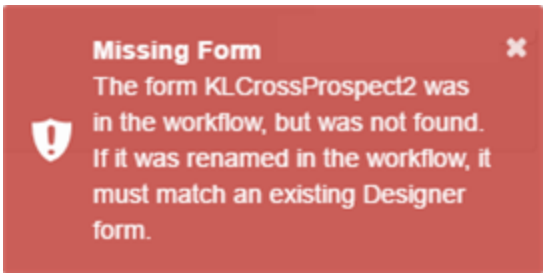
A Default-Confirmation form is available; however, this form will be overwritten when Forms Builder is updated. Either create a new form or make a copy of this form in Form Designer and update it as appropriate (branding, links to other sites or sequences, etc.). Save it and select “End State” in the Save dialog. This will make it appear in the “End State Form” property list for a sequence. It will not appear in the Forms list in Sequence Designer.

If, at some later time, you want to:

- Change the end state form, you must update the End State Form property in the sequence for the new form and re-save the sequence.
- Use the end state form as a regular form, re-save it with the “End State” property cleared. This will remove it from the “End State Form” list in the Sequence Properties and put it in the Forms list in Sequence Designer. This will not affect any previous sequences where it is used as an End State. Doing this in reverse is also true.

  You can also add a new State in the workflow or rename an existing State to use an End State form without consequences.

  When editing a workflow definition, keep in mind that a state in the state machine workflow equates to a form within the sequence. The name of a **State** must match the name of a **Form** to be rendered properly. If Renderer encounters a State in workflow definition that does not match name of any Form created in Form Designer, an error similar to the following will be generated.



Create a Custom Welcome Form

In Form Designer, select the Welcome form and select **Save As**. Then modify the new form.

— OR —

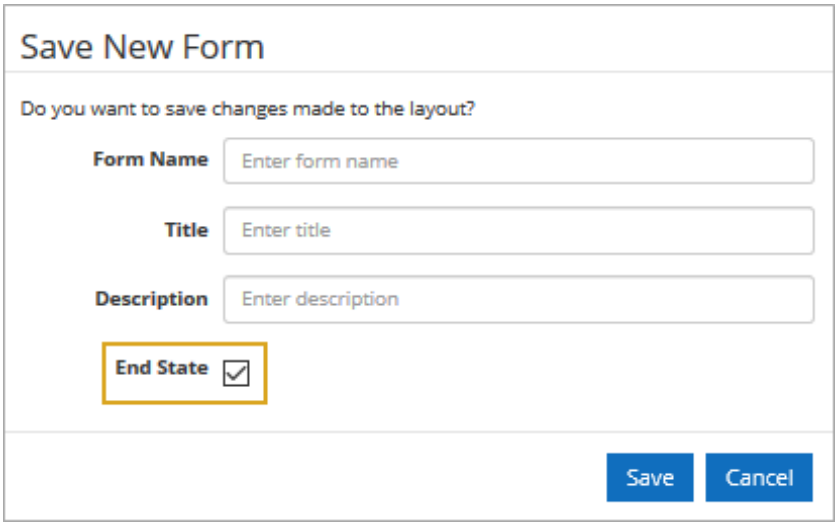
1. In Form Designer, select **New**, and drag the controls you want to use to the Layout pane. Typically, the HTML component will be used to display a 'Welcome' statement.
2. In the **Control Property Settings** pane, specify the desired properties.
3. **Save** the form with a new name.

Create a Custom Confirmation Form

In Form Designer, select the Default-Confirmation form and select **Save As**. Then modify the new form.

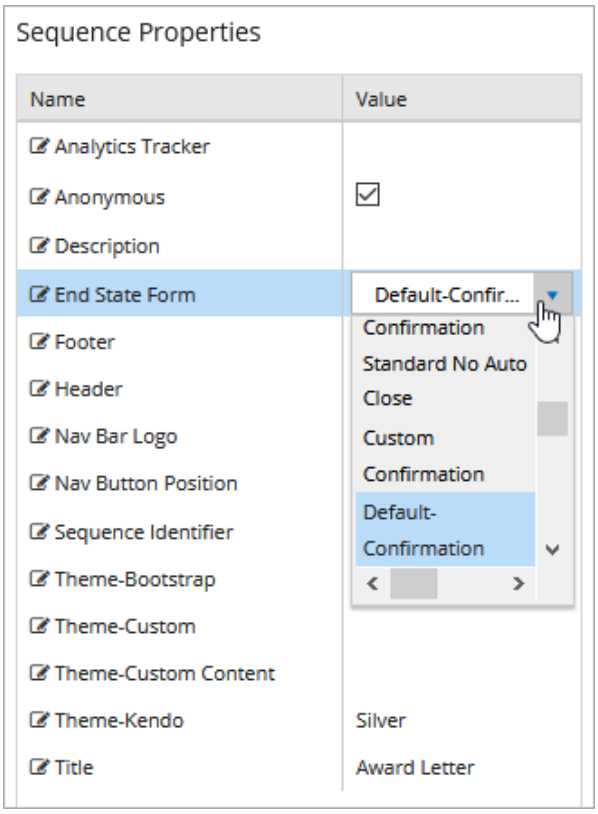
— OR —

1. In Form Designer, select **New** and drag the controls you want to use to the Layout pane. Typically, the HTML component will be used to display a "Thank You" statement.
2. In the **Control Property Settings** pane, specify the desired properties.
3. Select **Save** and complete the dialog. Be sure to select the **End State** option.



   The form is saved and added to the Forms list in Form Designer. You can modify the form as needed. If you save the form again without the End State attribute (after previously saving the form with the attribute), the form appears in the Forms column in Sequence Designer but remains available as an End State Form in existing workflows that already contain the form.
4. In Sequence Designer, select the sequence that will use your new form.
5. In the Sequence Properties pane, in the Value field of the End State Form property, select an **End State Form**.

   Do not select the Default-Confirmation form in sequences that will be moved to production because the Default-Confirmation will be overwritten during the next upgrade.

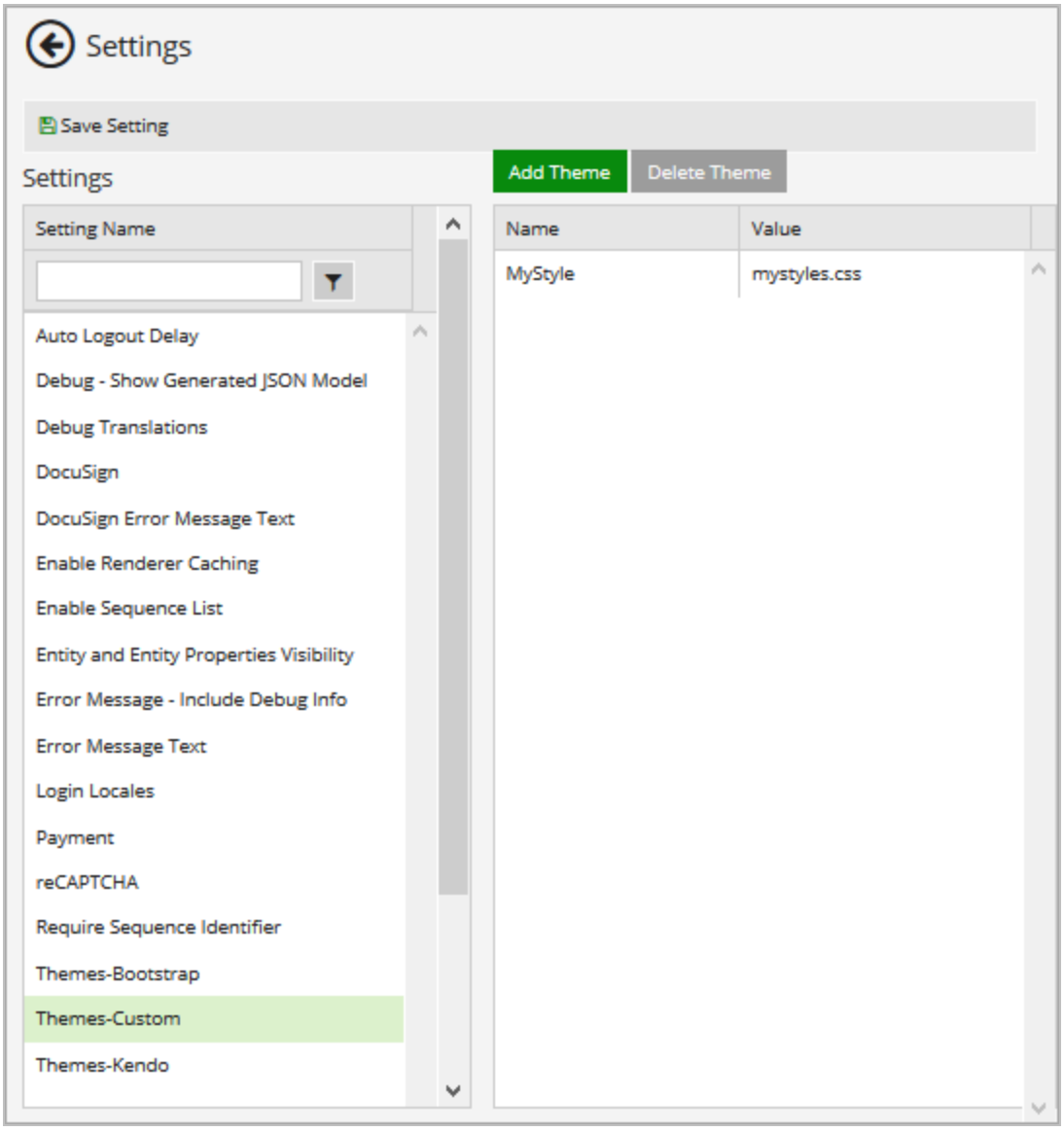


6. **Save** the sequence.
7. Navigate to the Sequence List and test the sequence.

> ⚠ Do not leave the *Default-Confirmation* as the End State Form for sequences that are moved to production. This form will be overwritten during the next upgrade. In all sequences that are exported to production, replace the Default-Confirmation form with a custom confirmation form.

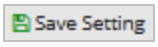
Settings

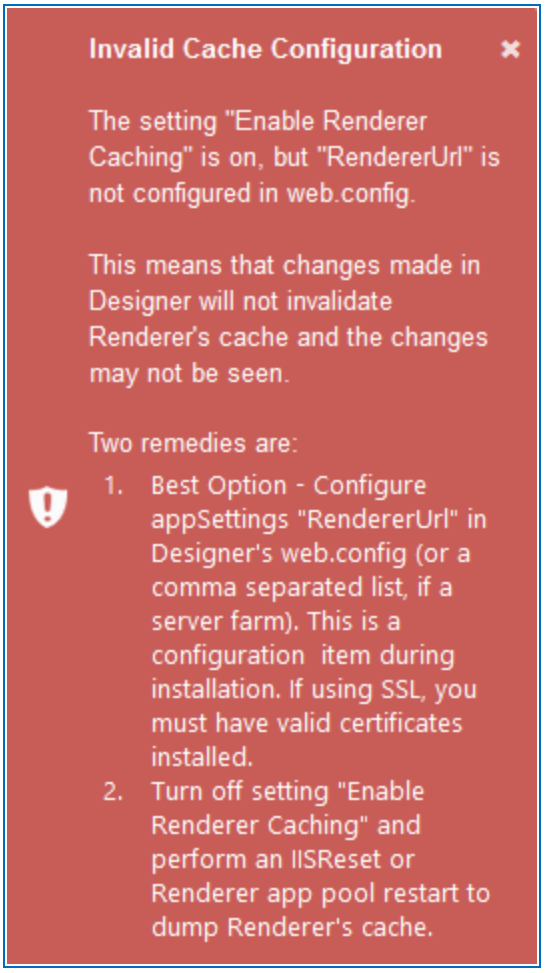
The Settings workspace is displayed when you select the Settings tile in the home page. This workspace enables you to define configuration settings for Forms Builder.

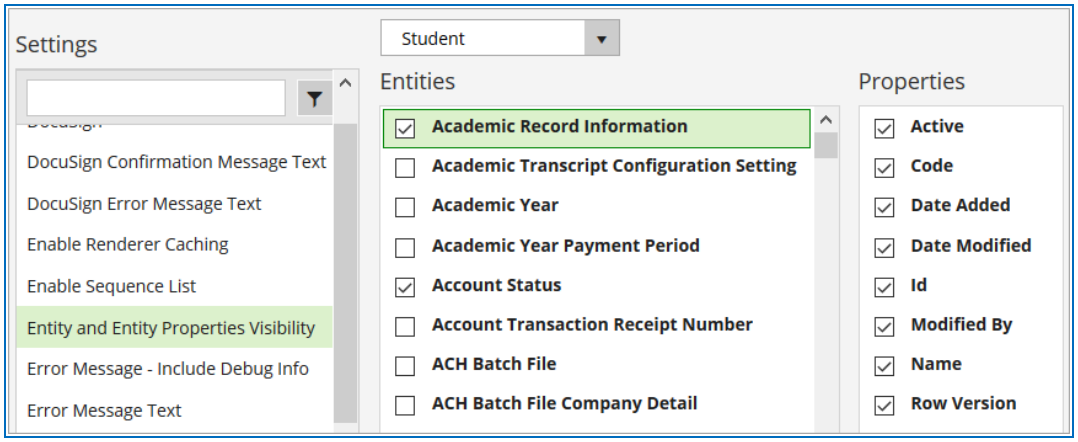
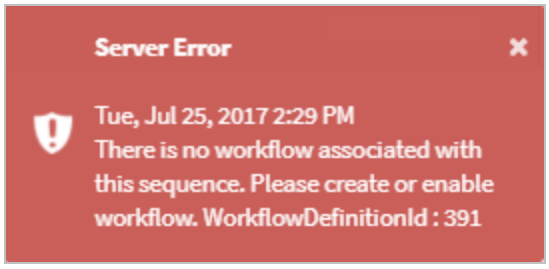


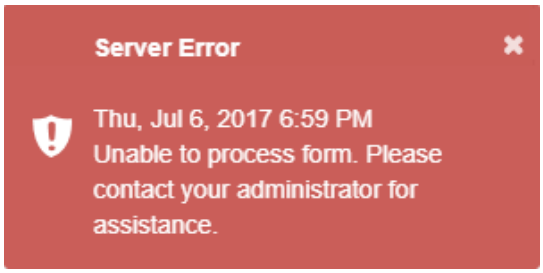
After a fresh installation of Forms Builder, before you can publish the URLs of rendered form sequences for end users, you must log in to Form Designer, select the **Settings** tile, provide your e-signature provider credentials, your Google Map credentials, your reCAPTCHA keys, and your Payment test keys as applicable. Updating the Error Message Text in the Settings is recommended but not mandatory.

Settings Workspace

Element	Description
	Select the left arrow to return to the home page.
	Save changes made to the Forms Builder settings.
	Use the search/filter tool to find a setting or scroll through the list. When you select a setting, the right pane displays the values associated with the setting.
Auto Logout Delay	Specify the delay in seconds before a user is automatically logged out from a sequence. The default value is 10 seconds. The value must be 0 or greater and there is no max. The delay applies if the Auto Logout when Complete property is selected in Sequence Designer.
Debug - Show Generated JSON Model	When this option is set to true (default=false), additional data that shows the values for objects on the form will be shown at the bottom of each rendered form. For more information, refer to Troubleshoot Rendered Sequences.
Debug Trans-lations	When this option is selected, text processed by the translation engine on rendered forms will be wrapped with '[ ]' markers and untranslated text will be prefixed with `[MISSING]:`.
DocuSign	Refer to DocuSign Settings.
DocuSign Error Message Text	Specify the message that is displayed when the DocuSign process was not completed due to an error condition, for example: "The DocuSign site reported that the signing authorization timed out, or some other network error occurred." For more information, refer to GetDocuSignRecipientStatus.

Element	Description
Enable Renderer Caching	In Forms Builder 3.4 and later, Renderer Caching is enabled (default=true) to increase the overall efficiency of sequence execution and performance of Forms Builder. If you change the Enable Renderer Caching setting, you must execute an **IISreset** on the server. Designer caching is on all the time and produces a noticeable performance improvement when moving between panels and workspaces within Designer. If Renderer Caching is enabled and no value is set for <add key="RendererUrl" value=""/> under <appSettings> in the Designer web.config file, the following message is displayed upon logging into Designer. Follow the instructions in the message to correct the configuration. 
Enable Sequence List	This setting determines whether users can access the Sequence List at the Forms Renderer URL (http://<server>.<domain>:<port>/#/Sequencelist). To enable the Sequence List, set this option to true (default=false). In previous versions of Forms Builder, access to the Sequence List was controlled by a setting in the web.config file for Forms Renderer.

Element	Description
Entity and Entity Properties Visibility	This setting enables you to filter entities and entity properties that are exposed in Form Designer when the "Show All Fields" option is cleared. 1. Select the database provider (**CRM** or **Student**). The Entities panel is populated. The first time this setting is accessed, it takes a few minutes to update and cache the list of entities. 2. In the Entities panel, select an entity. The Properties panel is populated with properties for the selected entity. Clear the check box to hide an entity. For each database provider several entities are pre-selected. You can change the settings for the pre-selected entities. 3. In the Properties panel, select the properties you want to expose. Clear the check boxes for properties you want to hide.  4. Select **Save**. 5. Return to Form Designer, select the provider, and verify that the list of entities matches your settings.
Error Message - Include Debug Info	When this setting option is set to true (default=false), log messages for server-side errors will be displayed in Forms Renderer.  This option is useful in a testing environment; however, it should be set to false in a production environment.

Element	Description
Error Message Text	When a server-side error occurs during the processing of a rendered sequence that cannot be corrected via form resubmission, an error message is displayed. The default message text is *"Unable to process form. Please contact your administrator for assistance."*  You can edit the message text on the Settings tile in Form Designer and save your custom message. You can use HTML markup to encode a URL or email address if desired. ⚠ By design, the error details will only be captured in the log files. You need to check the log files to troubleshoot the error. For more information, refer to Log Files.
Login Locales	Use this setting to select locales that will be displayed in a drop-down list on an Azure AD login page. Designate one of the selected cultures as the default. If only one locale is selected, the Azure AD login page will apply the selected locale without displaying a drop-down list. The Login Locales should match the Locale component used on the forms. If the Locale component on the form: • does not contain the Login Locales setting, the form will use the default locale set in the Locale component. • contains the Login Locales setting, the Login Locale will be passed to the Locale component. If the "Create Account" option is selected on the Azure AD login page, the user is directed to the "New Account Creation" page in Portal. The header bar in Portal has a "Choose language" drop-down that is not linked to the Login Locales setting.
Payment	Refer to Credit Card Payment component.
reCAPTCHA	Refer to CAPTCHA component.
Require Sequence Identifier	Refer to Sequence Identifier.
Themes-Bootstrap	When a Themes setting is selected, the **Add Theme** and **Delete Theme** buttons become available. You can add or delete individual .css files associated with the themes. For more information, refer to Themes.
Themes-Custom	
Themes-Kendo	

⚠ **In Forms Builder 3.5.1 and later**, the ability to set NLog levels in the Settings workspace of Form Designer is removed to prevent conflicts with Azure log configurations. Azure logs are stored in customer-specific tables. **If your Forms Builder deployment is in an Azure environment, contact Anthology Inc. obtain access to the Azure log tables or to request changes in the NLog settings.**

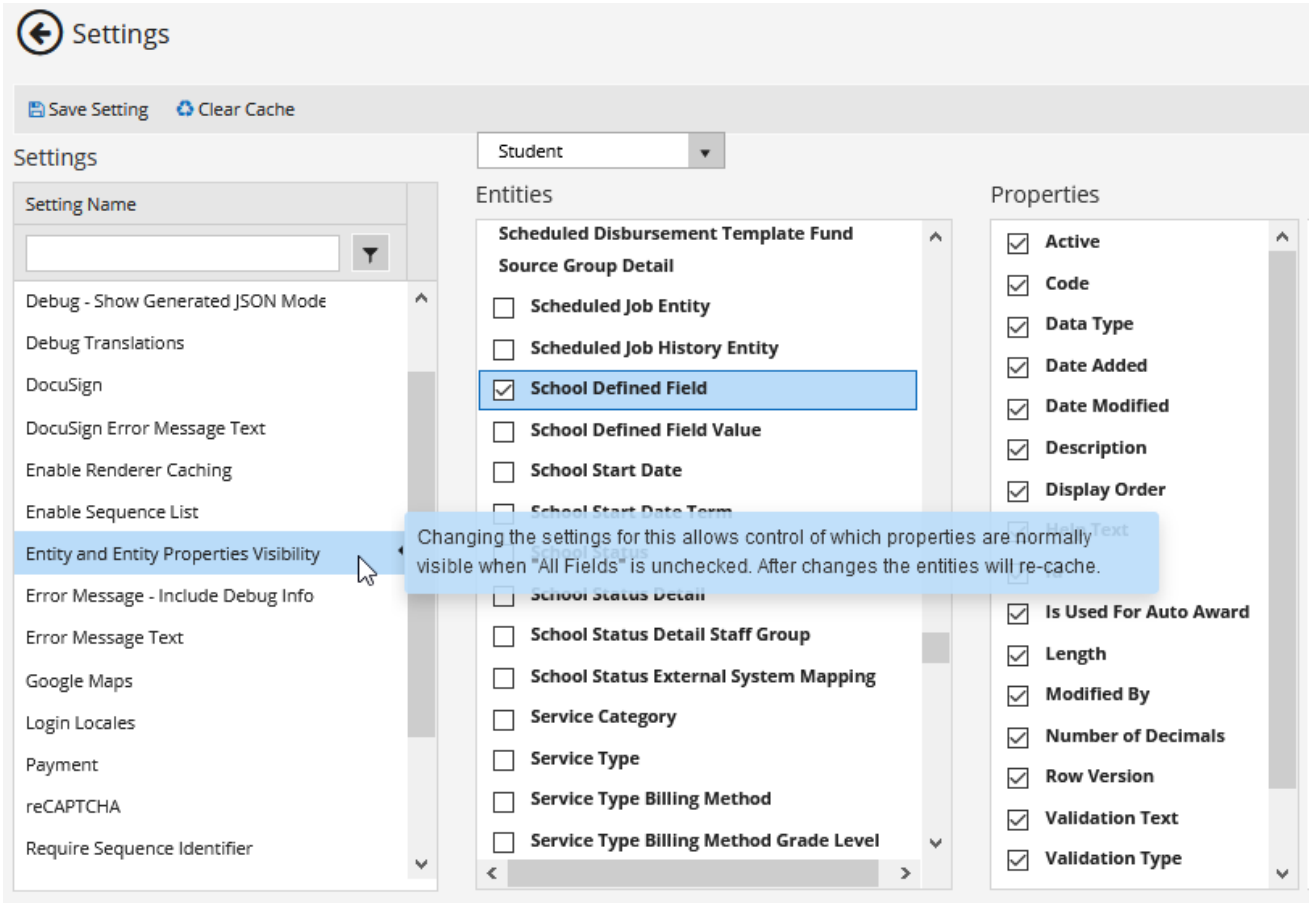
Entity and Property Visibility Settings

The Entity and Entity Properties Visibility setting enables you to filter entities and entity properties that are exposed in Form Designer when the "Show All Fields" option is cleared.

1. Select the database provider (**CRM** or **Student**). The Entities panel is populated. The first time this setting is accessed, it takes a few minutes to update and cache the list of entities.
2. If any new fields have been created in CampusNexus CRM, select the **Clear Cache** button at the top of the Settings pane in Forms Builder 3.7 and later, and confirm. This will reset the Application Pool for Form Designer and display the newly created fields in the Settings pane.

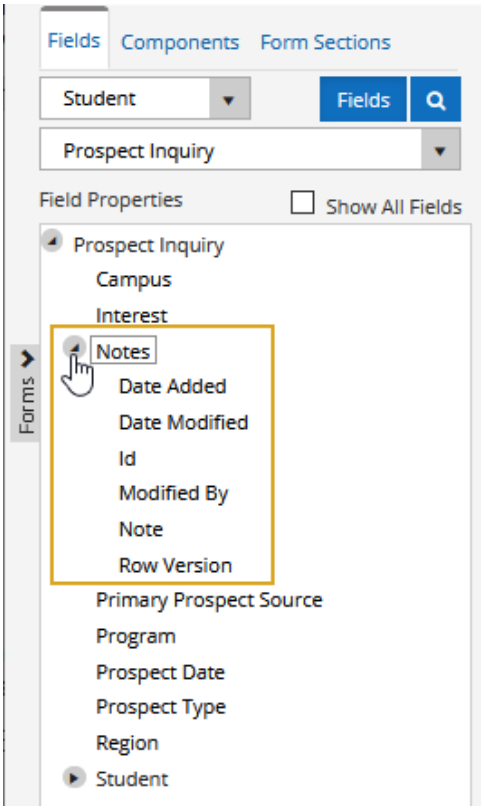
   Notes:

   - If any new School Defined Fields (SDF) have been created in Anthology Student, you need to perform an IIS reset to view the new fields.
   - If you were previously working in Form Designer, you may also have to clear your browser cache to access the new entities/entity properties.
3. In the Entities panel, select an entity. The Properties panel is populated with properties for the selected entity. Clear the check box to hide an entity. For each database provider several entities are pre-selected. You can change the settings for the pre-selected entities.
4. In the Properties panel, select the properties you want to expose. Clear the check boxes for properties you want to hide.



Navigation properties are marked with an icon (▶). To display the navigation properties in Form Designer, the name of the related entity must be selected along with the navigation property.

When a navigation property is expanded in Form Designer, the fields within a navigation property are displayed.



In some cases, the navigation property display name in Form Designer and the property name in the Settings workspace are slightly different (e.g., Program <> Prospect Inquiry Program of Interest).

5. Select **Save Setting**.
6. Return to Form Designer, select the provider, and verify that the list of exposed entities and properties satisfies your form design needs.

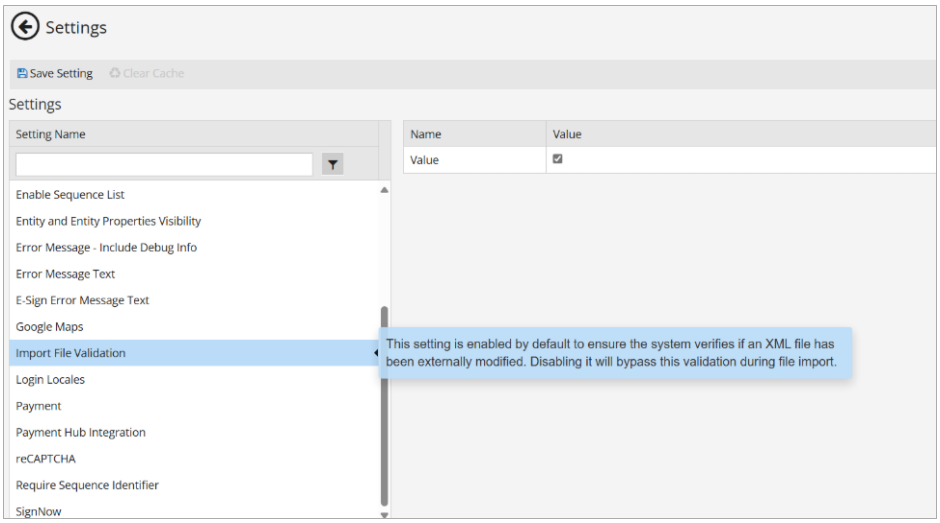
Import File Validation

The Import File Validation setting ensures the system verifies whether an XML file has been externally modified during imports. By default, this setting is enabled to validate file integrity.

If the Import File Validation setting is disabled:

- Only the user who disables the setting can import modified files.
- Other users will need to disable the setting themselves if they wish to import modified files.
- Once a modified file is imported, the system automatically re-enables the setting for security.
- If no file is imported, the setting remains disabled until manually re-enabled or an import is completed by that user.

All actions related to enabling/disabling the setting and importing files are tracked in the audit table.



How to Use

- Navigate to Settings in Forms Builder.
- Find Import File Validation and check or uncheck the box to enable or disable validation.
- Import your XML files based on the configured setting.

Workflows

Form sequences created in Forms Builder automatically create state machine workflows. These workflows can be modified using Workflow Composer to perform specific activities when the form sequences are executed. The following topics provide a general overview of state machine workflows, highlight common tasks associated with workflows, and provide reference information on workflows activities designed for use with Forms Builder.

The user who creates workflows is responsible for catching exceptions. Any unexpected and uncaught exceptions will abort workflows. For the guidelines on exception handling within workflows, refer to Recommended Coding for Activity Errors in Workflow Help.

Activities for Forms Builder

Workflows created by Forms Builder use workflow activities from the **Cmc.Nexus.FormsBuilder.Workflow** namespace. These activities are made available by installing the **Forms Builder Contracts x.x.x** package using the Package Manager within Workflow Composer.

In addition to the workflow activities from the Cmc.Nexus.FormsBuilder.Workflow namespace, workflows for form sequences use other activities, such as generic activities (Assign, If, StateMachine, etc.) and activities designed for Anthology products (CreateEntity, SaveStudentPortalUserAssociation, LookupStudent, etc.). For detailed information about these activities, refer to Workflow Help.

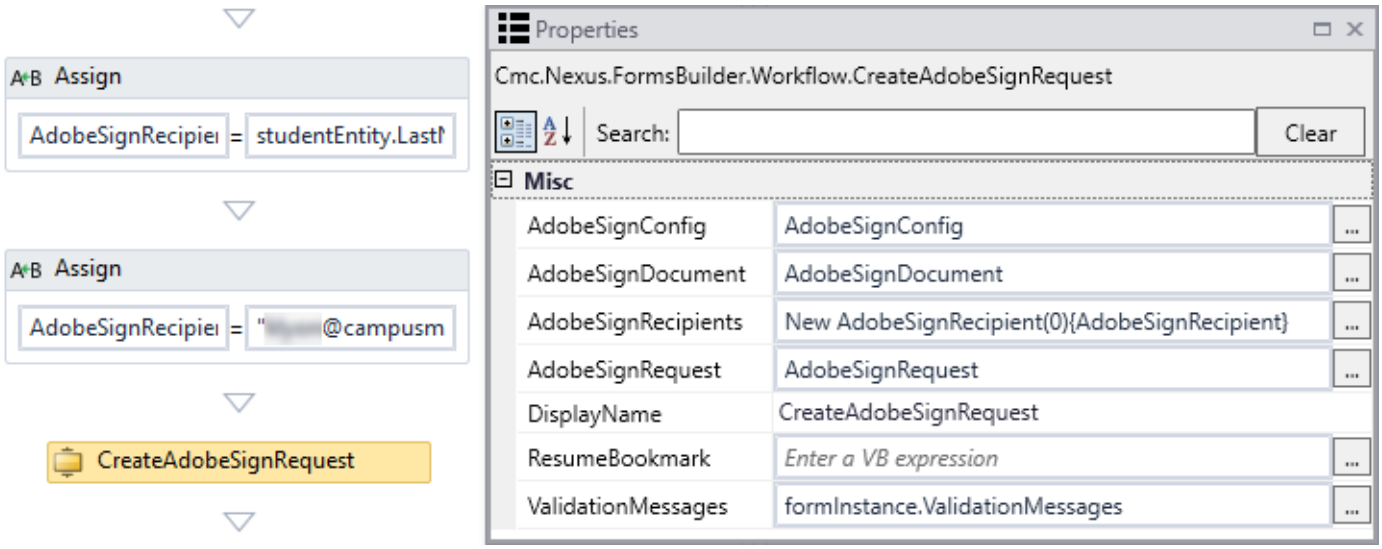
CreateAdobeSignRequest

The CreateAdobeSignRequest activity creates an e-signature request in the Adobe Sign service. The activity sends all applicable email addresses of signers and PDF files to be signed to Adobe Sign and receives a URL and Envelope Id (if applicable) in the response.

Prerequisites: refer to Adobe Sign Settings

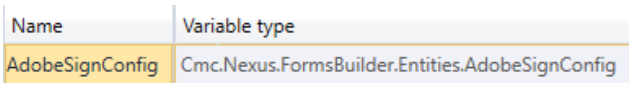
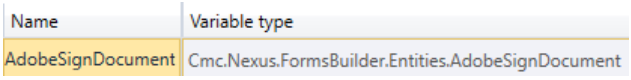
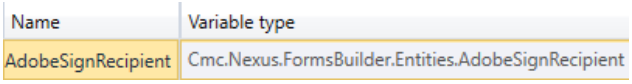
The CreateAdobeSignRequest activity uses the end point of the API login and account information configured in the Forms Builder Settings to obtain the Adobe Sign base URL for use in subsequent API calls.

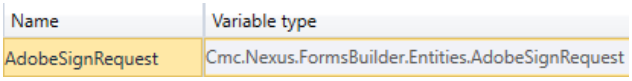
The CreateAdobeSignRequest activity should be placed in the form (state) that requires e-signatures. Refer to Adobe Sign Workflow Sample - Single Signer.



Properties

CreateAdobeSignRequest Properties

Property	Value	Required	Notes
AdobeSignConfig	`InArgument<AdobeSignConfig>`	Yes	Specify the AdobeSignConfig property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **AdobeSignConfig**, and select **OK**. 
AdobeSignDocument	`InArgument<AdobeSignDocument>`	Yes	Specify the AdobeSignDocument property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **AdobeSignDocument**, and select **OK**. 
AdobeSignRecipients	`InArgument<AdobeSignRecipient[]>`	Yes	Specify the AdobeSignRecipients property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, select **Array of [T]**, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **AdobeSignRecipient**, and select **OK**. 

Property	Value	Required	Notes
AdobeSignRequest	`OutArgument<AdobeSignRequest>`	Yes	Specify the AdobeSignRequest property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **AdobeSignRequest**, and select **OK**. 
DisplayName	`String`	No	Specify a name for the activity or accept the default.
ResumeBookmark	`InArgument<String>`	No	The ResumeBookmark property is used to resume a workflow that has been persisted while waiting for a signature. Specify a string to be displayed when the bookmark is resumed, e.g., "Continue when all signatures are collected".
ValidationMessages	`InOutArgument<ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

CreateDocuSignRequest

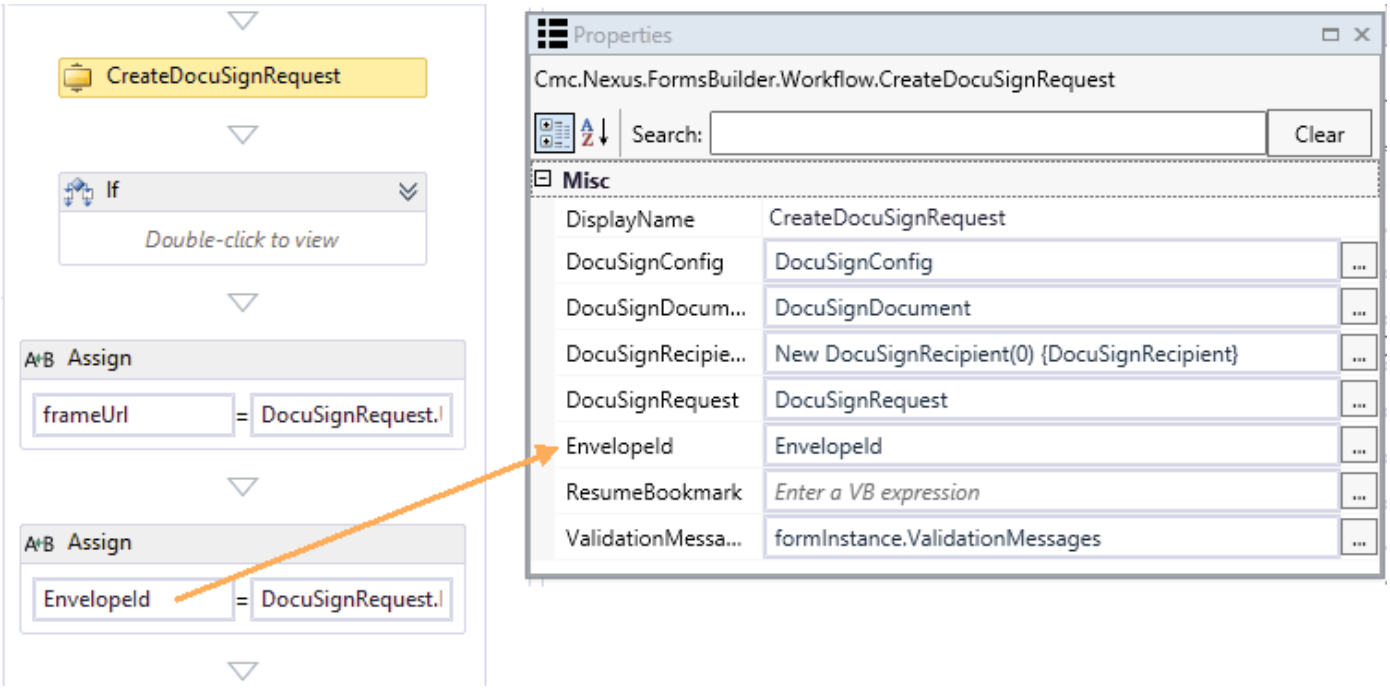
The CreateDocuSignRequest activity creates a new signature request in the DocuSign service. The activity sends all applicable email addresses of signers and PDF files to be signed to DocuSign and receives an Envelope Id and a URL in the response.

This workflow activity requires the following DocuSign Settings:

- DocuSign API account email and password
- Integrators Key

The CreateDocuSignRequest activity uses the end point of the API login information and the user's account ID to obtain the DocuSign base URL for use in subsequent API calls. DocuSign returns an Envelope ID and the base URL identifier to the CreateDocuSignRequest activity.

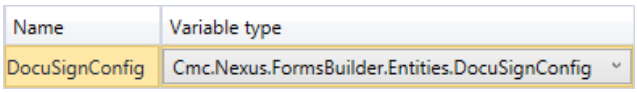
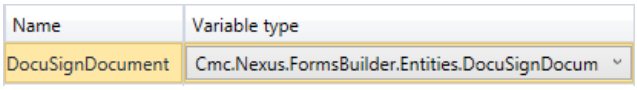
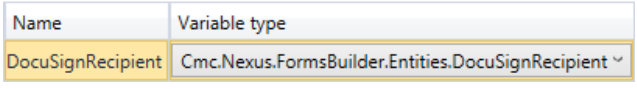
An uninitialized string variable (EnvelopeId) is passed to CreateDocuSignRequest with the EnvelopeId parameter. An unconditional assignment after the CreateDocuSignRequest assigns the value: EnvelopeId = DocuSignRequest.EnvelopeId.

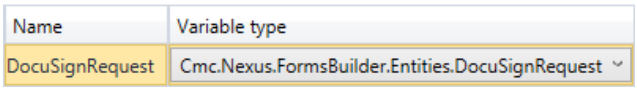


Properties

CreateDocuSignRequest Properties

Property	Value	Required	Notes
DisplayName	`String`	No	Specify a name for the activity or accept the default.

Property	Value	Required	Notes
DocuSignConfig	`InArgument<DocuSignConfig>`	Yes	Specify the DocuSignConfig property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **DocuSignConfig**, and select **OK**. 
DocuSignDocument	`InArgument<DocuSignDocument>`	Yes	Specify the DocuSignDocument property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **DocuSignDocument**, and select **OK**. 
DocuSignRecipients	`InArgument<DocuSignRecipient[]>`	Yes	Specify the DocuSignRecipients property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, select **Array of [T]**, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **DocuSignRecipient**, and select **OK**. 

Property	Value	Required	Notes
DocuSignRequest	`OutArgument<DocuSignRequest>`	Yes	Specify the DocuSignRequest property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **DocuSignRequest**, and select **OK**. 
EnvelopeId	`InArgument<String>`	No	Specify the existing EnvelopeId property if the workflow for the DocuSign sequence contains a GetDocuSignRecipientStatus activity which controls the workflow execution dependent on the signed status of the document. For example, if a user has not yet signed document, the workflow logic may route the user back into the DocuSign page. In this case, the CreateDocuSignRequest activity should reuse the existing EnvelopeId instead of creating a new envelope (DocuSign customers are charged per envelope).
ResumeBookmark	`InArgument<String>`	No	The ResumeBookmark property is used to resume a workflow that has been persisted while waiting for a signature. Specify a string to be displayed when the bookmark is resumed, e.g., "Continue when all signatures are collected"
ValidationMessages	`InOutArgument<ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

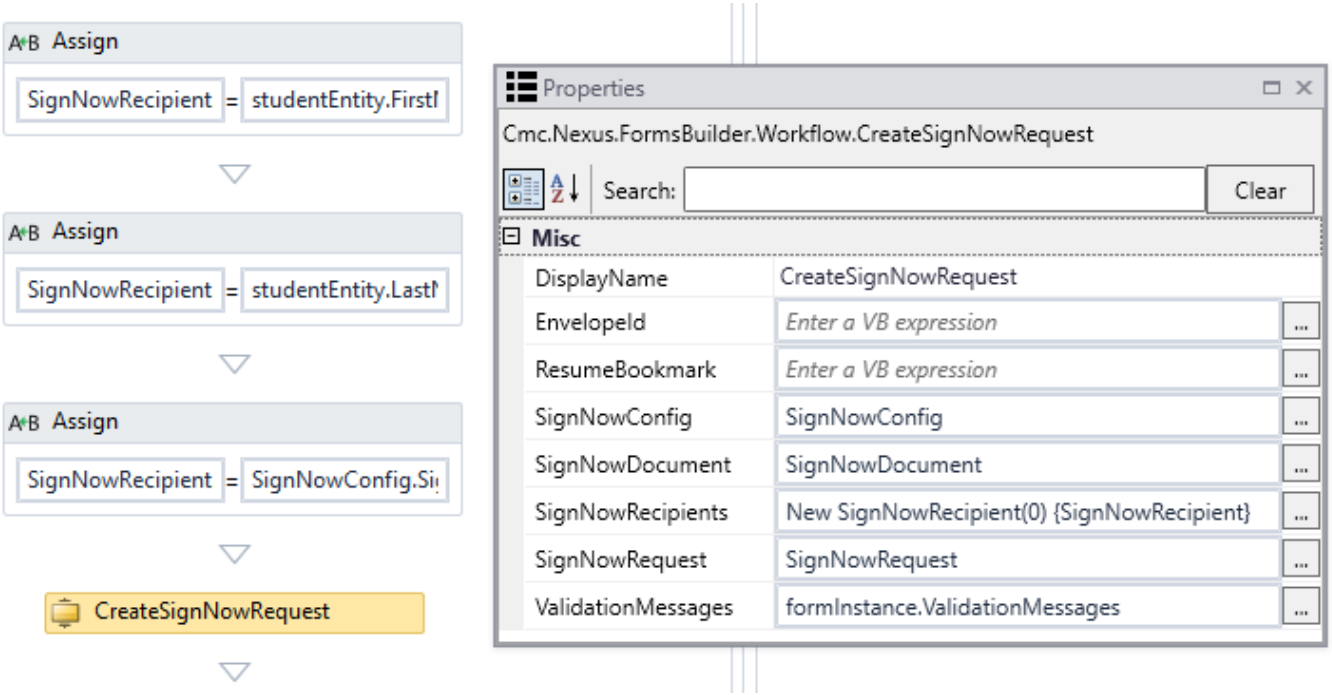
CreateSignNowRequest

The CreateSignNowRequest activity creates a new signature request in the SignNow service. The activity sends all applicable email addresses of signers and PDF files to be signed to SignNow and receives a URL and Envelope Id (if applicable) in the response.

Prerequisites: refer to SignNow Settings

The CreateSignNowRequest activity uses the end point of the API login and account information configured in the Settings to obtain the SignNow base URL for use in subsequent API calls.

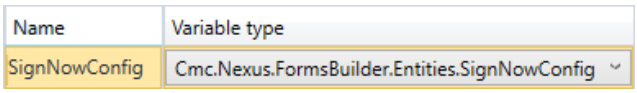
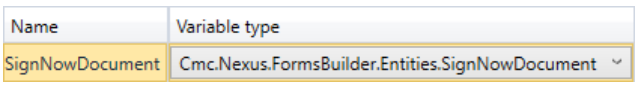
The CreateSignNowRequest activity should be placed in the form (state) that requires e-signatures. Refer to SignNow Workflow Sample - Single Signer.

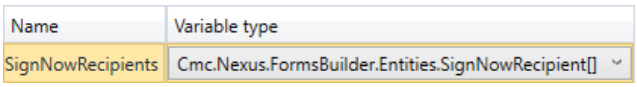
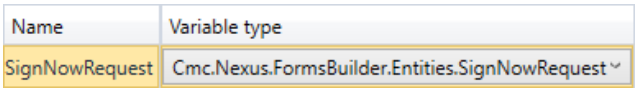


Properties

CreateSignNowRequest Properties

Property	Value	Requir-ed	Notes
DisplayName	`String`	No	Specify a name for the activity or accept the default.

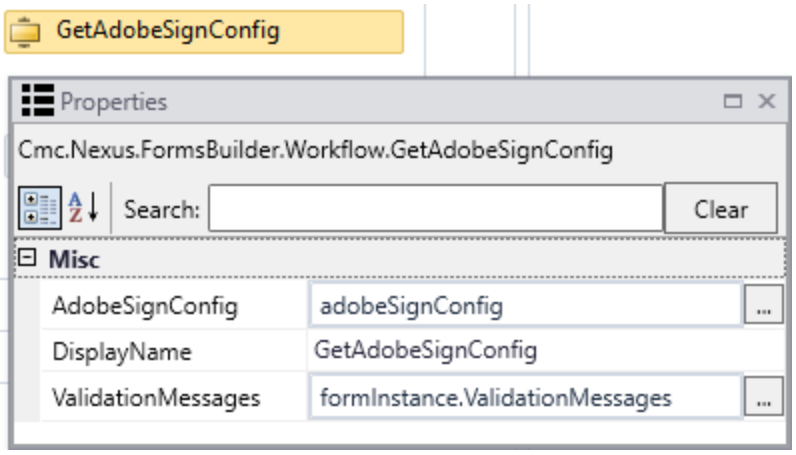
Property	Value	Required	Notes
EnvelopeId	`InArgument<String>`	No	Specify the existing EnvelopeId property if the workflow for the SignNow sequence contains a GetSignNowRecipientStatus activity which controls the workflow execution dependent on the signed status of the document. For example, if a user has not yet signed the document, the workflow logic may route the user back into the SignNow page. In this case, the CreateSignNowRequest activity should reuse the existing EnvelopeId instead of creating a new envelope (SignNow customers are charged per envelope).
ResumeBookmark	`InArgument<String>`	No	The ResumeBookmark property is used to resume a workflow that has been persisted while waiting for a signature. Specify a string to be displayed when the bookmark is resumed, e.g., "Continue when all signatures are collected".
SignNowConfig	`InArgument<SignNowConfig>`	Yes	Specify the SignNowConfig property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Entities.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **SignNowConfig**, and select **OK**. 
SignNowDocument	`InArgument<SignNowDocument>`	Yes	Specify the SignNowDocument property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Entities.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **SignNowDocument**, and select **OK**. 

Property	Value	Required	Notes
SignNowRecipients	`InArgument<SignNowRecipient[]>`	Yes	Specify the SignNowRecipients property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, select **Array of [T]**, navigate to **Cmc.Nexus.FormsBuilder.Entities**, select **SignNowRecipient**, and select **OK**. 
SignNowRequest	`OutArgument<SignNowRequest>`	Yes	Specify the SignNowRequest property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Entities**, select **SignNowRequest**, and select **OK**. 
ValidationMessages	`InOutArgument<ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

GetAdobeSignConfig

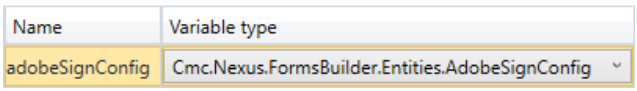
The GetAdobeSignConfig activity retrieves the Client ID, Client Secret, REST API Url, and Redirect URI from the Adobe Sign settings in Forms Builder. These values enable the workflow to log in to Adobe Sign.

The GetAdobeSignConfig activity should be placed in the Action area of the form (State) that requires the Adobe Sign signatures.



Properties

GetAdobeSignConfig Properties

Property	Value	Required	Notes
AdobeSignConfig	`OutArgument<AdobeSignConfig>`	Yes	Specify the AdobeSignConfig property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **AdobeSignConfig**, and select **OK**. 
DisplayName	`String`	No	Specify a name for the activity or accept the default.
ValidationMessages	`InOutArgument <ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

GetAdobeSignRecipientStatus

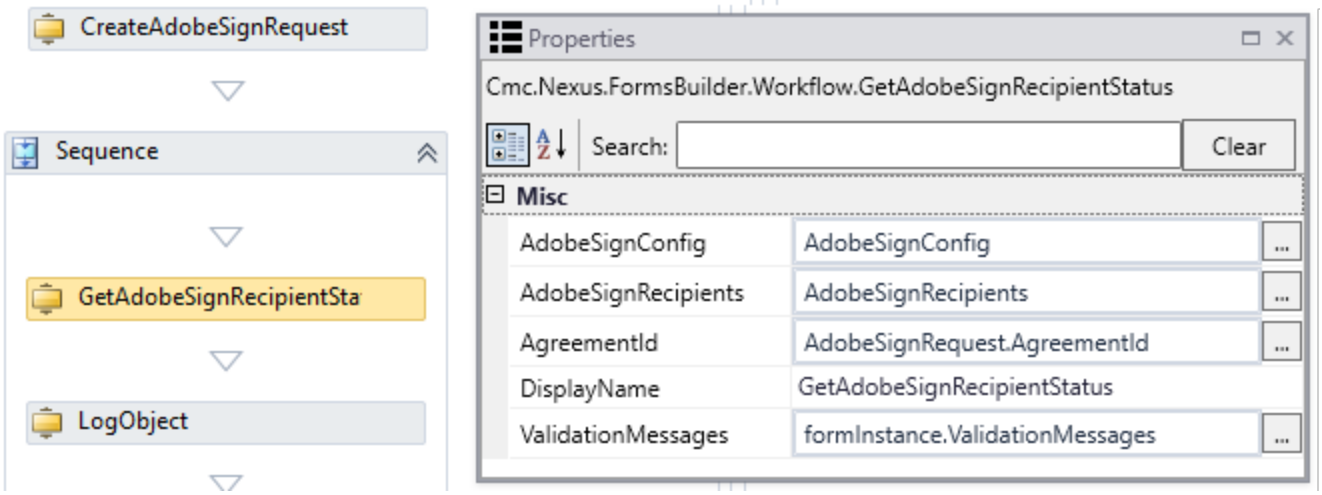
The GetAdobeSignRecipientStatus activity retrieves the signature status of Adobe Sign recipients. The activity passes an Agreement Id and receives the signature status of each recipient. This activity allows for conditional processing in the workflow so that if the document has not been successfully signed, the flow of the workflow can be routed back to the state immediately prior to the Adobe Sign page ensuring that a document has been electronically signed before proceeding with any additional processing in the workflow.

Information about the Adobe Sign signature status is useful to handle all multi route workflows (e.g., retry, completed, denied) and to recover from error conditions including connection loss. For example, when the connection to the Adobe Sign server is interrupted and the signature status indicates that the document was not signed successfully, the user may be prompted to retry the sequence.

The Adobe Sign API used by the GetAdobeSignRecipientStatus activity supports the following statuses:

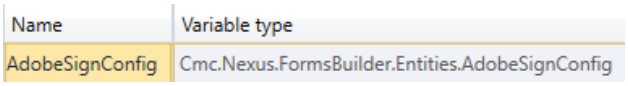
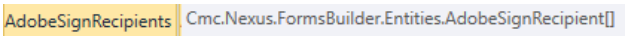
- CANCELLED
- COMPLETED
- EXPIRED
- NOT_YET_VISIBLE
- WAITING_FOR_OTHERS
- WAITING_FOR_MY_APPROVAL
- WAITING_FOR_AUTHORING
- WAITING_FOR_MY_ACKNOWLEDGEMENT
- WAITING_FOR_MY_ACCEPTANCE
- WAITING_FOR_MY_FORM_FILLING
- WAITING_FOR_MY_DELEGATION
- WAITING_FOR_MY_SIGNATURE
- WAITING_FOR_MY_VERIFICATION
- WAITING_FOR_PREFILL

Some status values do not apply to the Forms Builder use case.



Properties

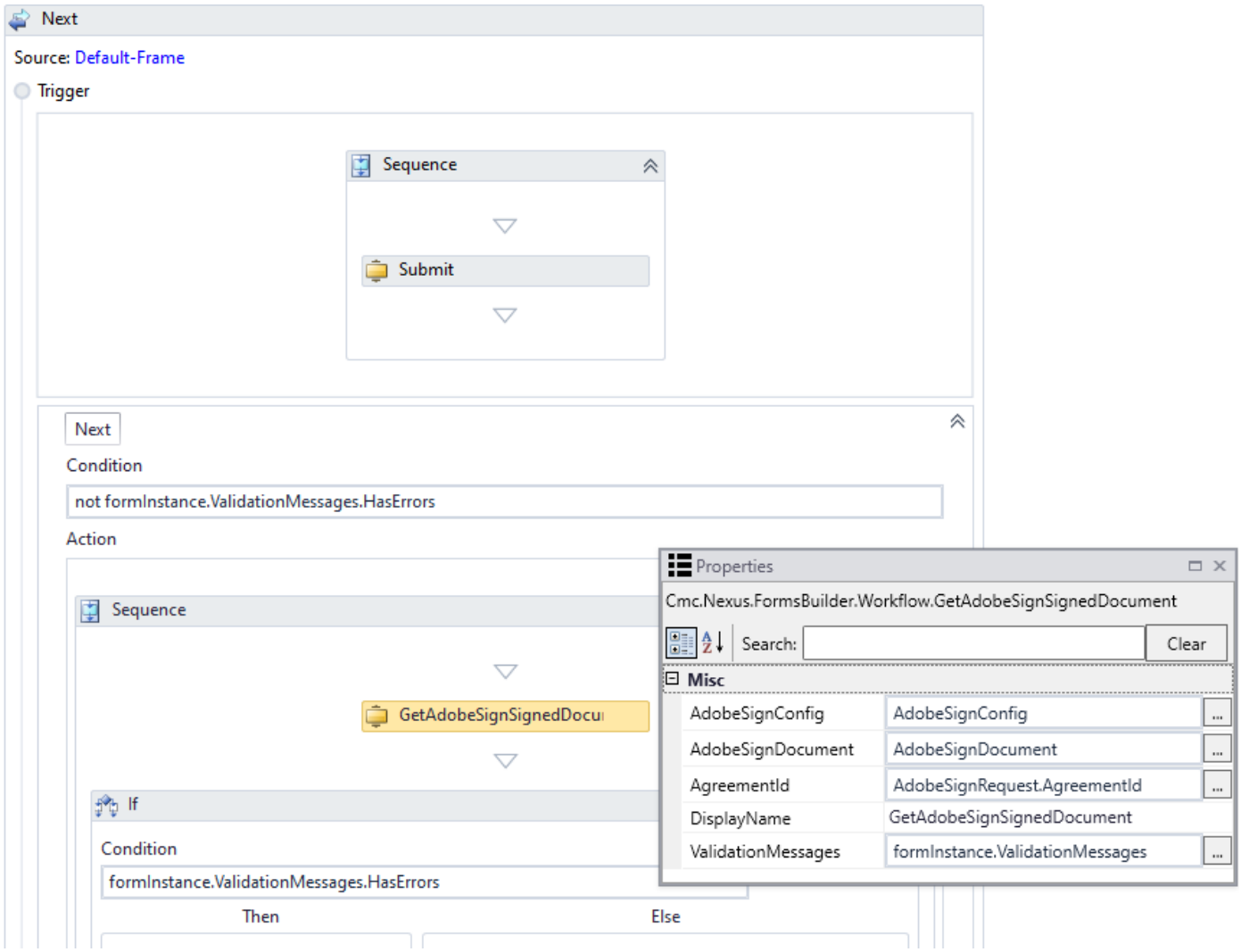
GetAdobeSignSignedDocument Properties

Property	Value	Required	Notes
AdobeSignConfig	`InArgument<AdobeSignConfig>`	Yes	Specify the AdobeSignConfig property using a VB expression or variable. This is the out argument returned by the GetAdobeSignConfig activity. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Entities.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **AdobeSignConfig**, and select **OK**. 
AdobeSignRecipients	`OutArgument<AdobeSignRecipient[]>`	Yes	Specify the AdobeSignRecipients array property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Entities.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **AdobeSignRecipient**, and select **OK**. 
AgreementId	`InArgument<String>`	Yes	Specify the Agreement Id as AdobeSignRequest.AgreementId.
DisplayName	`String`	No	Specify a name for the activity or accept the default.
ValidationMessages	`InOutArgument<ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

GetAdobeSignSignedDocument

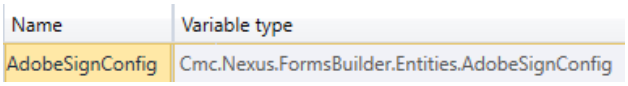
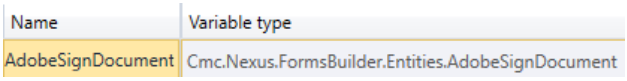
The GetAdobeSignSignedDocument activity retrieves a signed document from Adobe Sign. The activity passes an Agreement Id and receives the document for the passed Agreement Id in the response.

The GetAdobeSignSignedDocument activity should be placed in the Trigger area of the transition immediately following the IFrame (Default-Frame) that will hold the signed document.



Properties

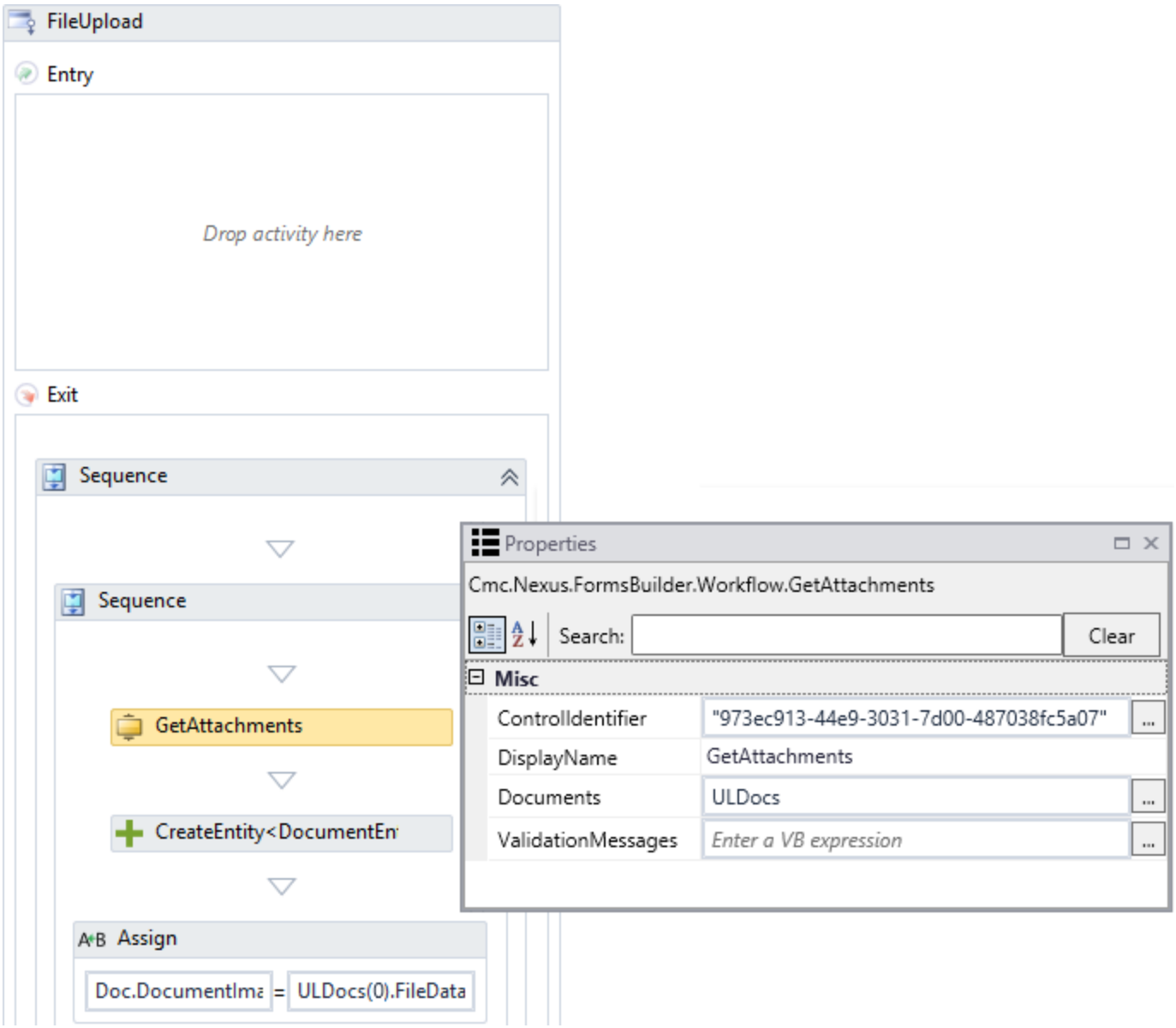
GetAdobeSignSignedDocument Properties

Property	Value	Required	Notes
AdobeSignConfig	`InArgument<AdobeSignConfig>`	Yes	Specify the AdobeSignConfig property using a VB expression or variable. This is the out argument returned by the GetAdobeSignConfig activity. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Entities**, select **AdobeSignConfig**, and select **OK**. 
AdobeSignDocument	`OutArgument<AdobeSignDocument>`	Yes	Specify the AdobeSignDocument property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Entities**, select **AdobeSignDocument**, and select **OK**. 
AgreementId	`InArgument<String>`	Yes	Specify the Agreement Id as AdobeSignRequest.AgreementId.
DisplayName	`String`	No	Specify a name for the activity or accept the default.
ValidationMessages	`InOutArgument<ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

GetAttachments

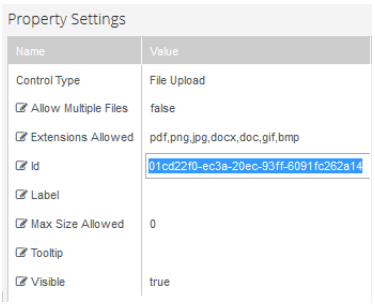
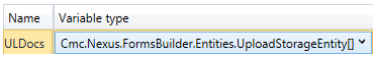
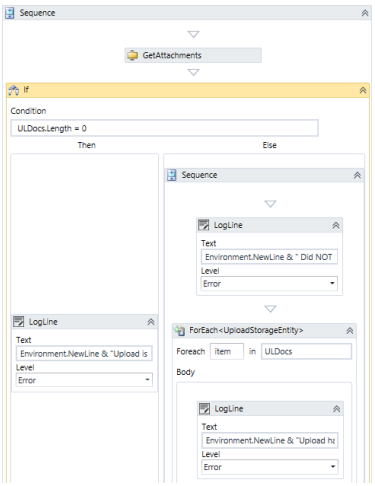
The GetAttachments activity is used with the File Upload component in Forms Builder. The activity returns an array of files uploaded through the File Upload component as `UploadStorageEntity[]`.

To access all the attached files, you can use a ForEach loop to iterate over the array; or, if a single file is attached, access the file contents as `UploadStorageEntity(0).FileData`.



Properties

GetAttachments Properties

Property	Value	Required	Notes
ControlIdentifier	`InArgument<String>`	Yes	Specify the Control Identifier property from the Id field of the File Upload control in Form Designer. 
DisplayName	`String`	No	Specify a name for the activity or accept the default.
Documents	`OutArgument <UploadStorageEntity[]>`	Yes	The activity returns an array of document images. This property is a variable that can be used as input for subsequent activities in the workflow.  To check if a file was attached, you can insert a condition similar to the following `ULDocs.Length = 0`(where ULDocs is the variable of the OutArgument). 

Property	Value	Required	Notes
ValidationMessages	`InOutArgument <ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

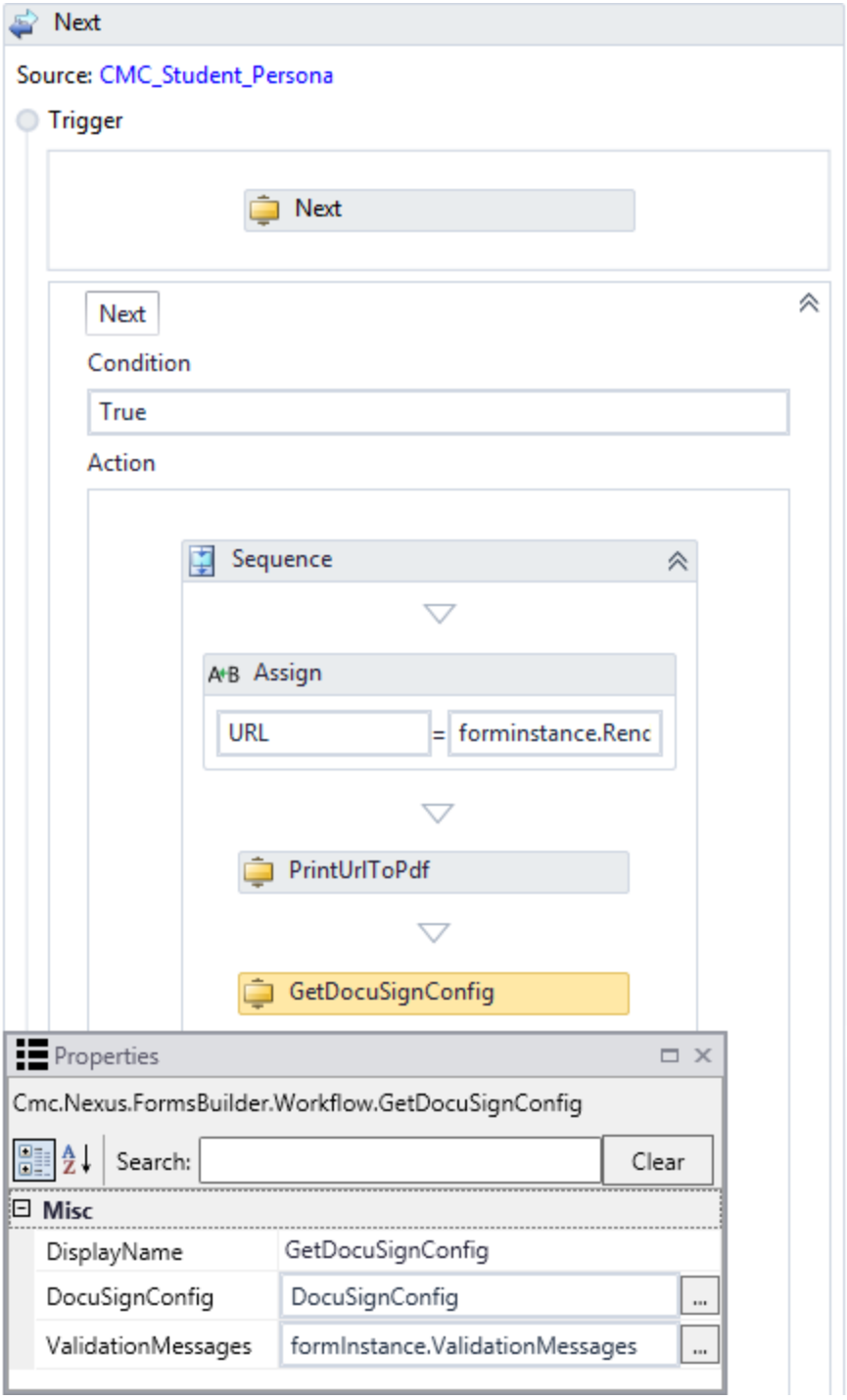
GetDocuSignConfig

The GetDocuSignConfig activity retrieves the User Name, Password, Integrators Key, and REST API Url from the DocuSign settings in Forms Builder. These values enable the workflow to log in to DocuSign.

The GetDocuSignConfig activity should be placed in the Action area of the form (state) that requires the DocuSign signatures.

This workflow activity requires the following DocuSign Settings:

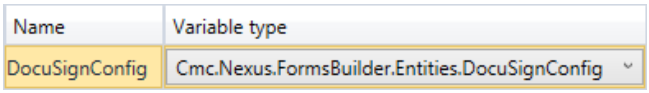
- DocuSign API account email and password
- Integrators Key



Properties

GetDocuSignConfig Properties

Property	Value	Require-d	Notes
DisplayName	`String`	No	Specify a name for the activity or accept the default.

Property	Value	Required	Notes
DocuSignConfig	`OutArgument<DocuSignConfig>`	Yes	Specify the DocuSignConfig property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **DocuSignConfig**, and select **OK**.  As of Forms Builder 3.4, the DocuSignConfig.TestMode assignment (=true or =false) is no longer supported or functional. Assign statements containing it can be deleted. ⚠ The only two properties for the DocuSignConfig object that should ever be modified in the workflow definition are **EmailSubject** and **ReturnUrl**. The values for all the other properties are retrieved from the DocuSign Settings saved in the database. Any modification done to the values for the other DocuSignConfig properties in the workflow definition will likely result in errors when the DocuSign portion of the sequence is executed.
ValidationMessages	`InOutArgument <ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

GetDocuSignRecipientStatus

The GetDocuSignRecipientStatus activity retrieves the signature status of DocuSign recipients. The activity passes an Envelope ID and receives the Envelope IDs as well as the signature status of each recipient. This activity allows for conditional processing in the workflow so that if the document has not been successfully signed, the flow of the workflow can be routed back to the state immediately prior to the DocuSign page ensuring that a document has been electronically signed before proceeding with any additional processing in the workflow.

The activity will mainly be used for the primary signer (i.e., the first recipient in an array - Recipients(0)) who signs the document within the rendered form. The activity is not applicable for additional signers who will sign via email.

Information about the DocuSign signature status is useful to handle all multi route workflows and to recover from error conditions including connection loss. For example, when the connection to the DocuSign server is interrupted and the signature status indicates that the document was not signed successfully, the user may be prompted to retry the sequence.

The GetDocuSignRecipientStatus activity supports the following statuses:

DocuSign Recipient Statuses

Status	Description
Completed	Success path
Created	This status would likely not be relevant in a workflow. The process would not proceed from DocuSign.
Declined	This status occurs if a signer declines to sign a document. This is a specific case that can be handled at the workflow designer's discretion (e.g., trigger an email or some other action)
Delivered	This status is applicable as a status for additional signers.
Other	Network/timeout case The error message displayed when a DocuSign process fails due to network or timeout error can be configured in the Forms Builder Settings workspace under the setting item "DocuSign Error Message Text".
Sent	This status occurs if the user selects “Finish Later”. This status should probably be handled at the workflow designer's discretion.
Signed	This status occurs briefly before the Completed status. The Signed status would likely not be relevant in a workflow.

Multi Route Workflow Example

You could use the GetDocuSignRecipientStatus activity in a multi route workflow to select a path based on the status returned by the activity.

For example, if the status is:

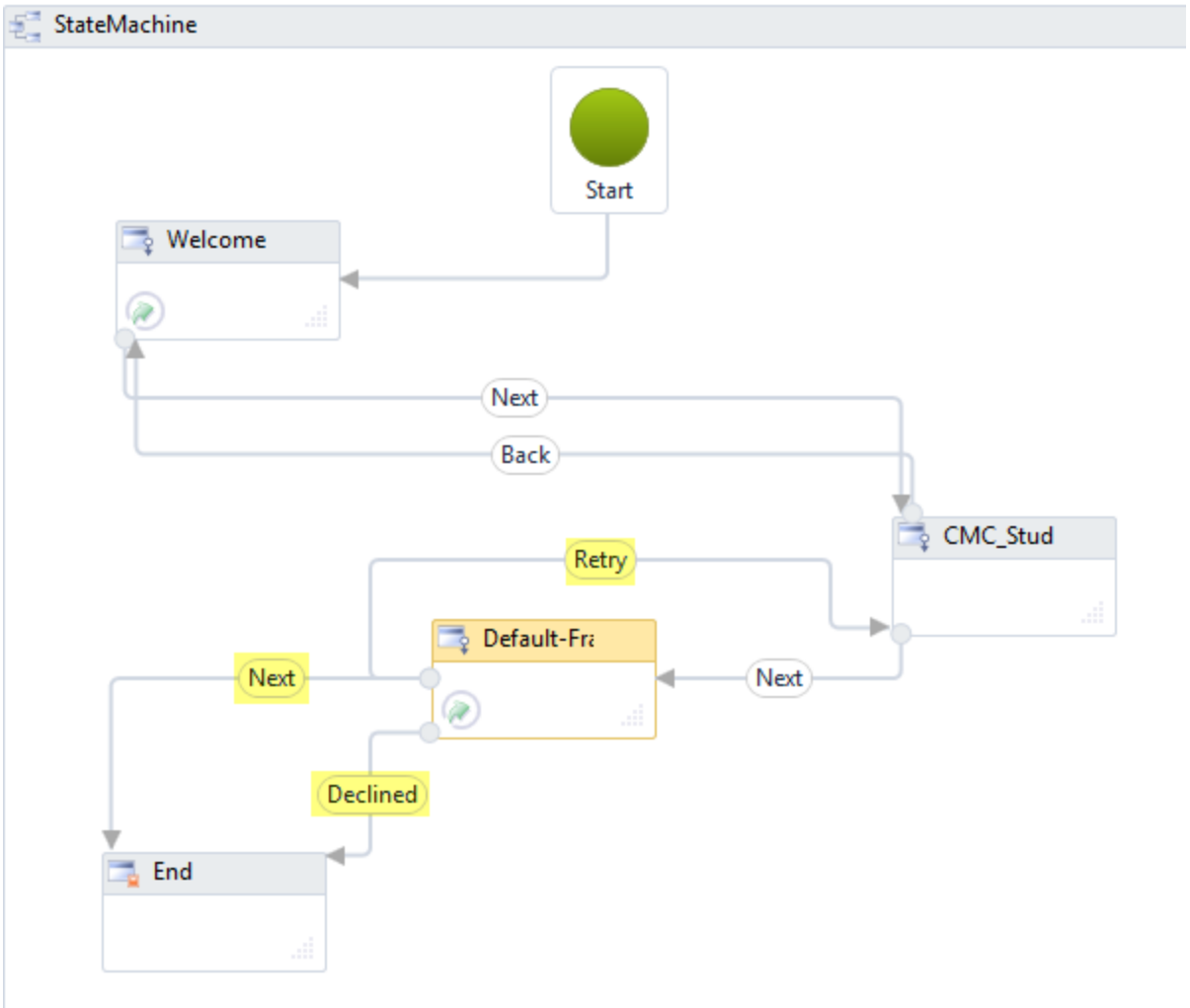
a. **Declined**, go to the end of the workflow, display a validation error, and do not update document status. The document should remain in the student's Portal page because the document status would not have changed.

b. **Completed/Signed**, go to the end of the workflow, and update the document status. (**Next** path highlighted below.)

c. Anything else, retry the workflow and go to the form prior to the DocuSign sequence. (**Retry** path highlighted below).

Note that the order of the multi route conditions is important:

- The first check is for a specific Declined status.
- The 2nd check is for a specific Completed status.
- The 3rd check ("Retry") is a catch-all for any status condition.

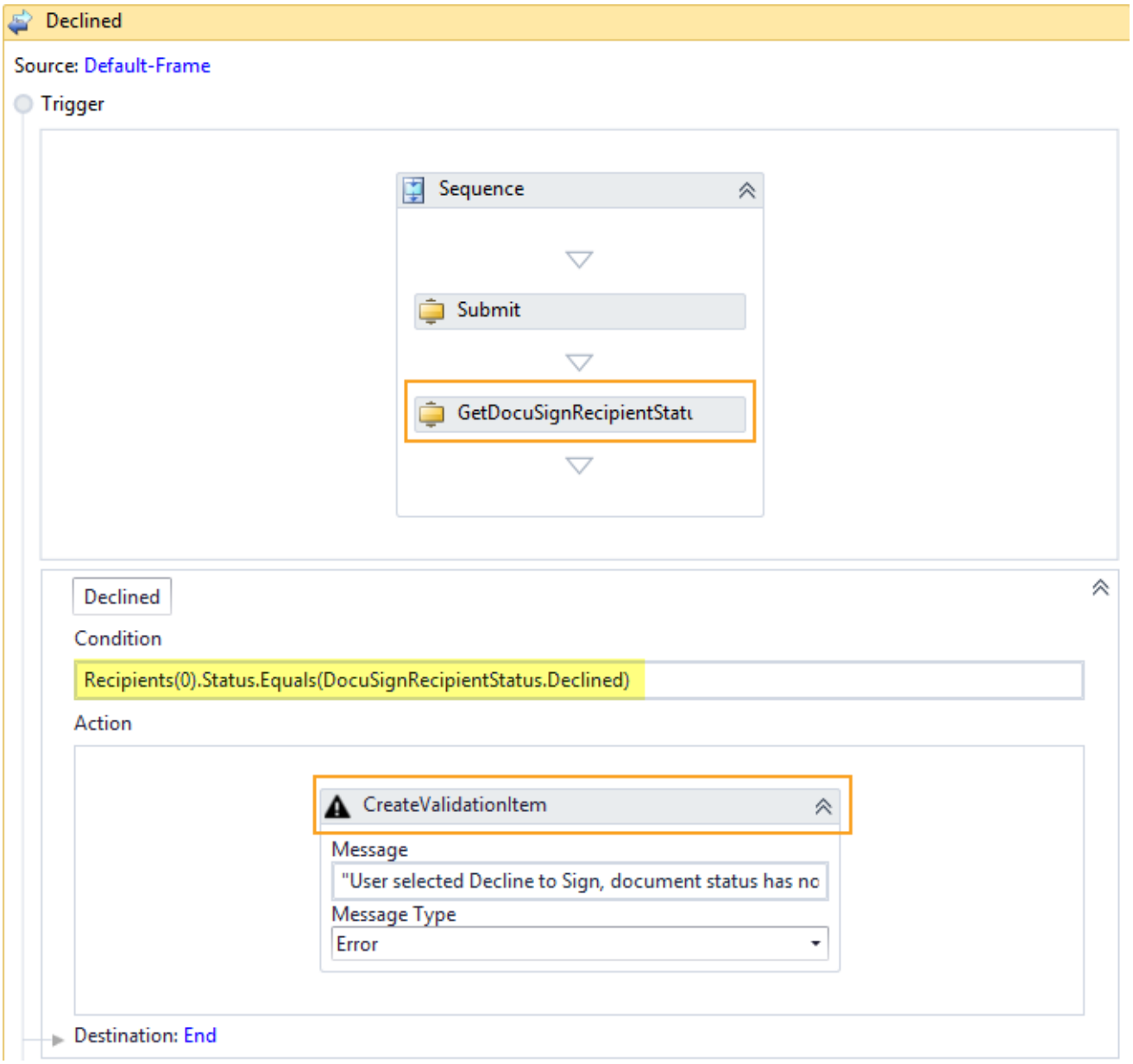


a. **Declined path**

The transition from the Default-Frame state to the End state contains a GetDocuSignRecipientStatus activity and the following condition:

Recipients(0).Status.Equals(DocuSignRecipientStatus.Declined)

This is followed by a CreateValidationItem activity that generates the following error message: "User selected Decline to Sign, document status has not been updated."



In a DocuSign sequence with multiple signers:

- If the additional signer declines to sign the document, the additional signer receives a second email canceling their access to sign the document.
- If the connection is interrupted and the additional signer does not sign the document upon receiv-

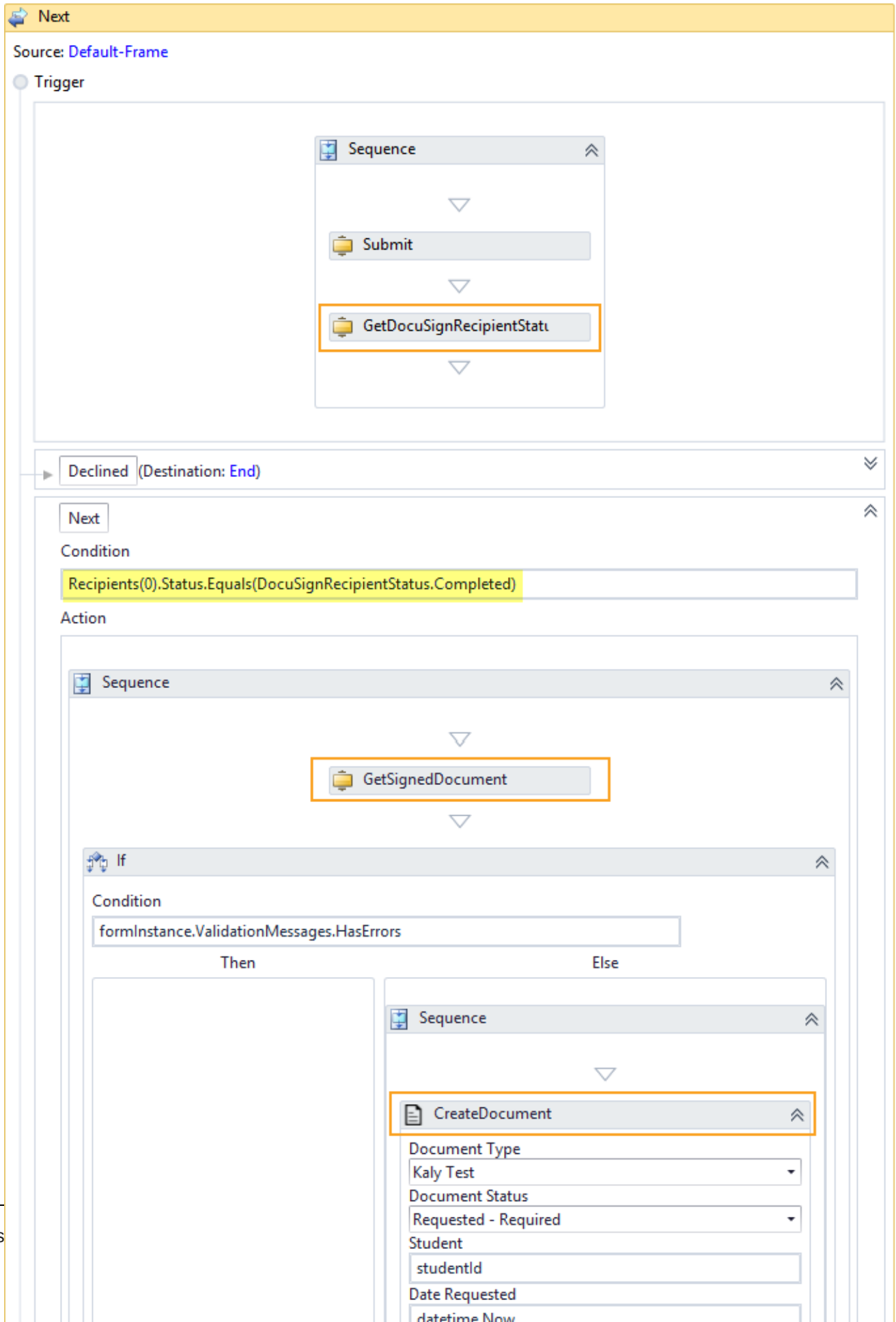
ing the first email (interrupted), the additional signer will receive a second email and will be allowed to sign (good retry).

b. **Completed/Signed path**

The transition from the Default-Frame state to the End state contains a GetDocuSignRecipientStatus activity and the following condition:

Recipients(0).Status.Equals(DocuSignRecipientStatus.Completed)

This is followed by GetSignedDocument and CreateDocument activities.


Next
Source: Default-Frame
Trigger
Sequence
Submit
GetDocuSignRecipientStatı
Declined (Destination: End)
Next
Condition
Recipients(0).Status.Equals(DocuSignRecipientStatus.Completed)
Action
Sequence
GetSignedDocument
If
Condition
formInstance.ValidationMessages.HasErrors
Then
Else
Sequence
CreateDocument
Document Type
Kaly Test
Document Status
Requested - Required
Student
studentId
Date Requested
datetime.Now

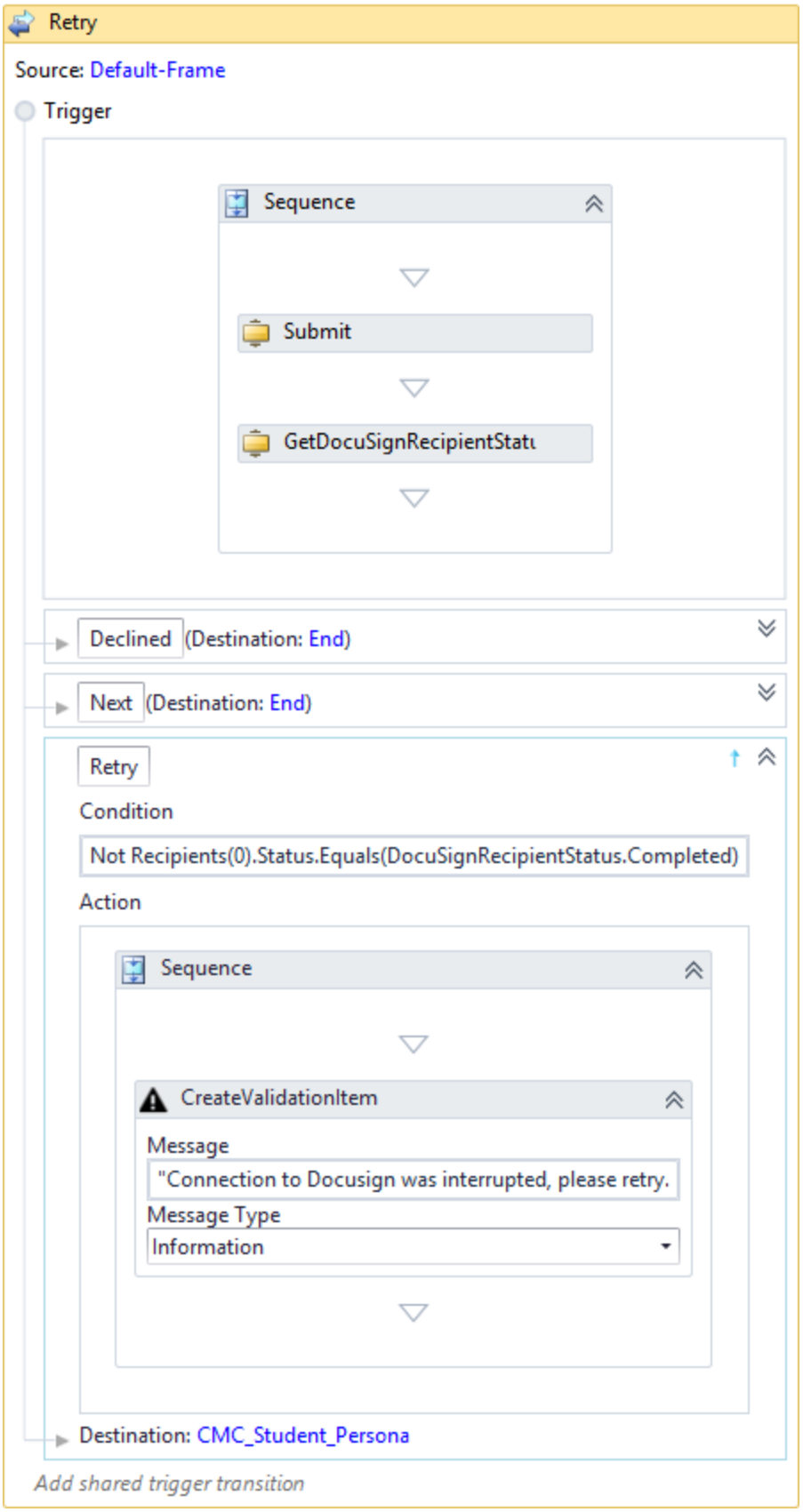
c. **Retry path**

The transition from the Default-Frame state to the CMC_Student_PersonalInfo state (this is the form prior to the DocuSign sequence) contains a GetDocuSignRecipientStatus activity and the following condition:

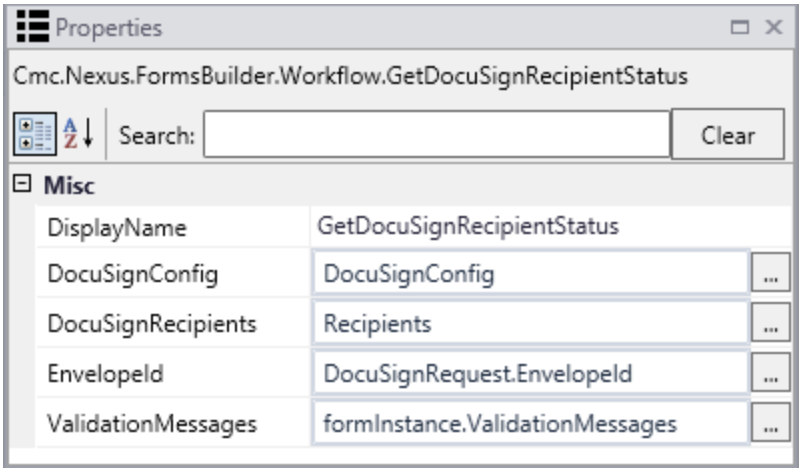
Not Recipients(0).Status.Equals(DocuSignRecipientStatus.Completed)

Reuse the originally generated Envelope Id since DocuSign charges for each unique Envelope Id.

This is followed by a CreateValidationItem activity that generates the following information message: "Connection to DocuSign was interrupted, please retry."

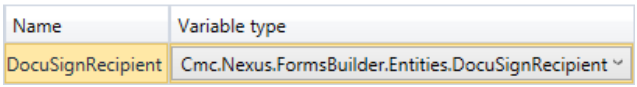

Retry
Source: Default-Frame
Trigger
Sequence
Submit
GetDocuSignRecipientStat
Declined (Destination: End)
Next (Destination: End)
Retry
Condition
Not Recipients(0).Status.Equals(DocuSignRecipientStatus.Completed)
Action
Sequence
CreateValidationItem
Message
"Connection to Docusign was interrupted, please retry.
Message Type
Information
Destination: CMC_Student_Persona
Add shared trigger transition

Properties



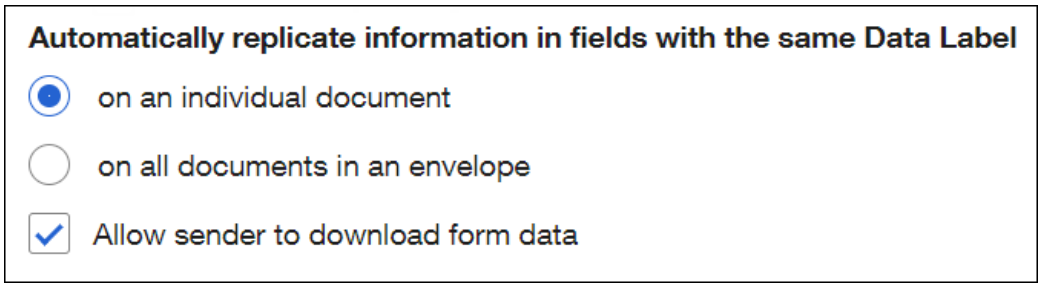
GetDocuSignRecipientStatus Properties

Property	Value	Required	Notes
DisplayName	`String`	No	Specify a name for the activity or accept the default.
DocuSignConfig	`InArgument<DocuSignConfig>`	Yes	Specify the DocuSignConfig property using a VB expression or variable. This is the out argument returned by the GetDocuSignConfig activity. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **DocuSignConfig**, and select **OK**. Name: DocuSignConfig Variable type: Cmc.Nexus.FormsBuilder.Entities.DocuSignConfig

Property	Value	Required	Notes
DocuSignRecipients	`OutArgument<DocuSignRecipient[]>`	No	Specify the DocuSignRecipients array property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, select **Array of [T]**, navigate to **Cmc.Nexus.FormsBuilder.Entities**, select **DocuSignRecipient**, and select **OK**. 
EnvelopeId	`InArgument<String>`	Yes	Specify the EnvelopeId as DocuSignRequest.EnvelopeId.
ValidationMessages	`InOutArgument <ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

GetDocuSignRecipientFormData

Prerequisite: Before using this activity, log on to your DocuSign Account and select **Sending Settings** in the left pane. Select the permission **Allow sender to download form data**.



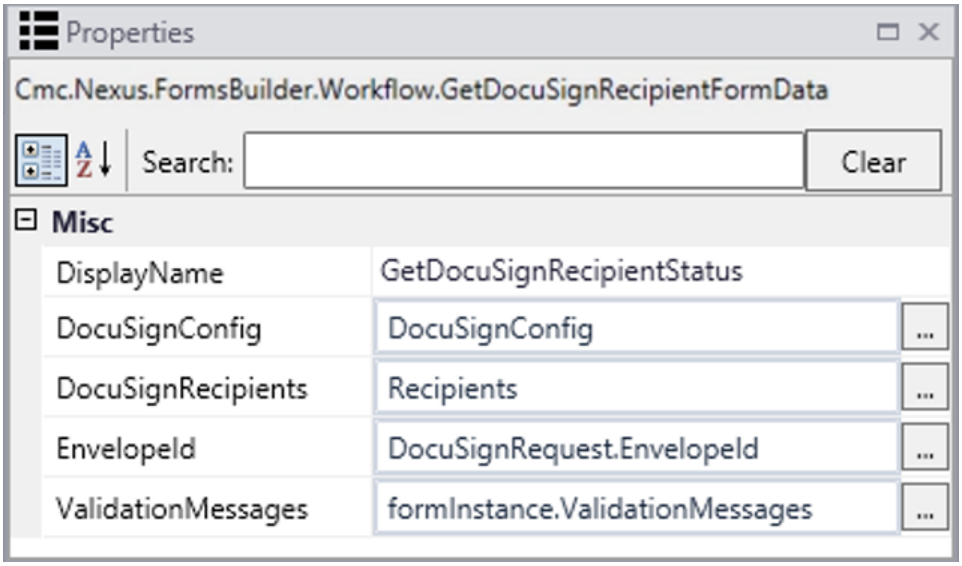
Based on the envelope ID, this activity returns form data, i.e., the list of values typed by each signer in the signed document along with the tag name of the DocuSign component.

In the rendered form, this activity will be used for the primary signer and all additional signers who will sign the document.

The following envelope information will be returned:

Property	Description
Field	The value of the **Tag Name** field typed in Forms Builder designer.

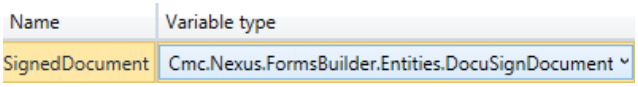
Property	Description
Value	The value typed in the field by the recipient.



Properties

GetSignedDocument Properties

Property	Value	Required	Notes
DisplayName	`String`	No	Specify a name for the activity or accept the default.
DocuSignConfig	`InArgument<DocuSignConfig>`	Yes	Specify the DocuSignConfig property using a VB expression or variable. This is the out argument returned by the GetDocuSignConfig activity. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Entities.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **DocuSignConfig**, and select **OK**. Name: DocuSignConfig Variable type: Cmc.Nexus.FormsBuilder.Entities.DocuSignConfig

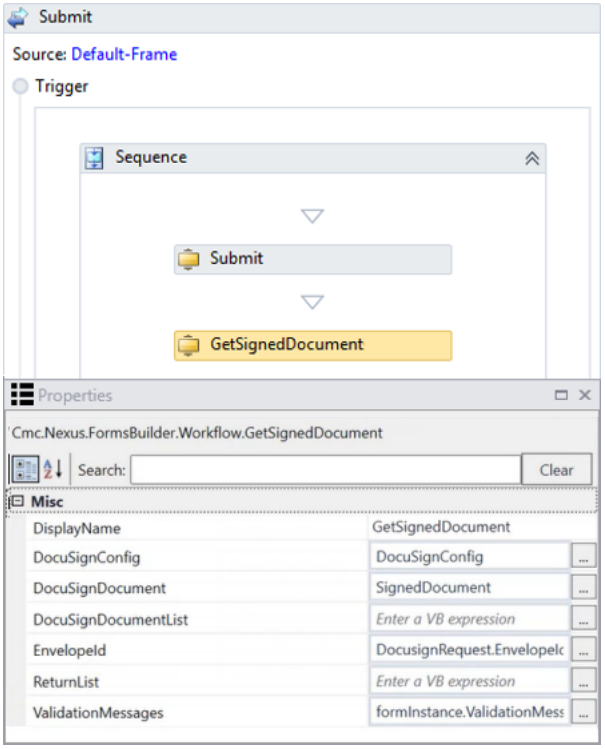
Property	Value	Required	Notes
DocuSignRecipients	`OutArgument<DocuSignDocument[]>`	No	Specify the DocuSignDocument property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Array of [T]**. In the Select Type dialog, select Browse for Types.... In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Entities**, select **DocuSignDocument** and select **OK**. 
EnvelopeId	`InArgument<String>`	Yes	Specify the EnvelopeId as DocuSignRequest.EnvelopeId.
ValidationMessages	`InOutArgument<ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

Next steps can be triggered after this activity to iterate through the retrieved list to save or print retrieved data, or trigger other steps in the workflow based on an institution's business case.

GetSignedDocument

The GetSignedDocument activity retrieves a signed document and its attachments from DocuSign. The activity passes an Envelope Id and receives the document for the passed Envelope Id in the response.

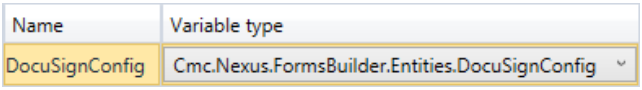
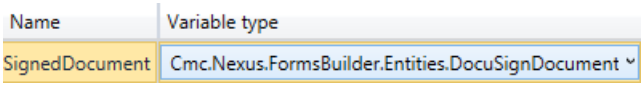
The GetSignedDocument activity should be placed in the Trigger area of the transition immediately following the IFrame component (State) that will hold the signed document.

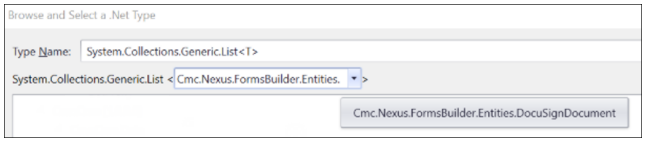


Properties

GetSignedDocument Properties

Property	Value	Requir-ed	Notes
DisplayName	`String`	No	Specify a name for the activity or accept the default.

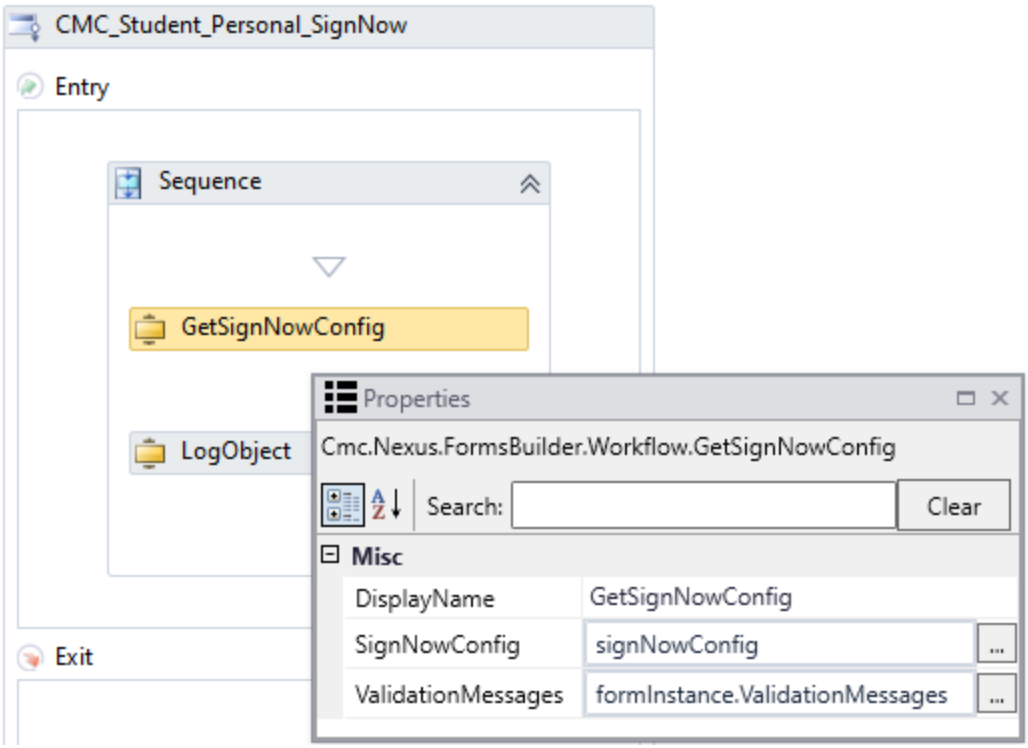
Property	Value	Required	Notes
DocuSignConfig	`InArgument<DocuSignConfig>`	Yes*	Specify the DocuSignConfig property using a VB expression or variable. This is the out argument returned by the GetDocuSignConfig activity. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **DocuSignConfig**, and select **OK**.  * This property is required in Forms Builder 3.3 and later. It is optional for workflows created in prior versions of Forms Builder (backward compatibility).
DocuSignDocument	`OutArgument<DocuSignDocument>`	Yes	Specify the DocuSignDocument property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Entities**, select **DocuSignDocument**, and select **OK**.  The signed document will be returned, excluding any attachments if the ReturnList property is set to **False**.

Property	Value	Required	Notes
DocuSignDocumentList	`OutArgument List<DocuSignDocument>`	No	Specify the DocuSignDocumentList property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and in Select a .NET Type" window, navigate to System.Collections.Generic.List. In the Browse and Select a .Net Type dialog, in the System.Collections.Generic.List dropdown, select **Browse for Types...**. In the **Type Name** field, navigate to CMC.Nexus.FormsBuilder.Entities.DocuSignDocument and select **OK**.  When ReturnList is set to **True**, along with the signed document this property will also return the list of attached files uploaded by all signers.
EnvelopeId	`InArgument<String>`	Yes	Specify the EnvelopeId as DocuSignRequest.EnvelopeId.
ReturnList	`InArgument<Boolean>`	Yes	Specify the boolean values True or False. Setting this value to: • **True** will trigger the DocuSignDocumentList property to return a collection of signed documents, documents attached by all signers and a summary report. OR **False** (default value) will trigger the DocuSignDocument property to return only the signed document. On upgrading to version 4.1, the signed document will continue to be returned because the default value of this property in existing instances of this activity will be False.
ValidationMessages	`InOutArgument <ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

GetSignNowConfig

The GetSignNowConfig activity retrieves the credentials and Urls from the SignNow settings in Forms Builder. These values enable the workflow to log in to the SignNow service.

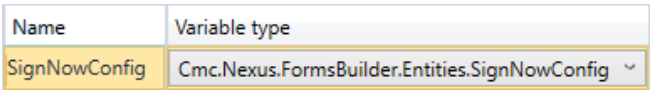
The GetSignNowConfig activity should be placed in the form (state) that requires e-signatures.



Properties

GetSignNowConfig Properties

Property	Value	Require-d	Notes
DisplayName	`String`	No	Specify a name for the activity or accept the default.

Property	Value	Required	Notes
SignNowConfig	`OutArgument<SignNowConfig>`	Yes	Specify the SignNowConfig property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **SignNowConfig**, and select **OK**. 
ValidationMessages	`InOutArgument <ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

GetSignNowRecipientStatus

The GetSignNowRecipientStatus activity retrieves the signature status of SignNow recipients. The activity passes a Document ID and receives the signature status of each recipient. This activity allows for conditional processing in the workflow so that if the document has not been successfully signed, the flow of the workflow can be routed back to the state immediately prior to the SignNow page ensuring that the document has been electronically signed before proceeding with any additional processing in the workflow.

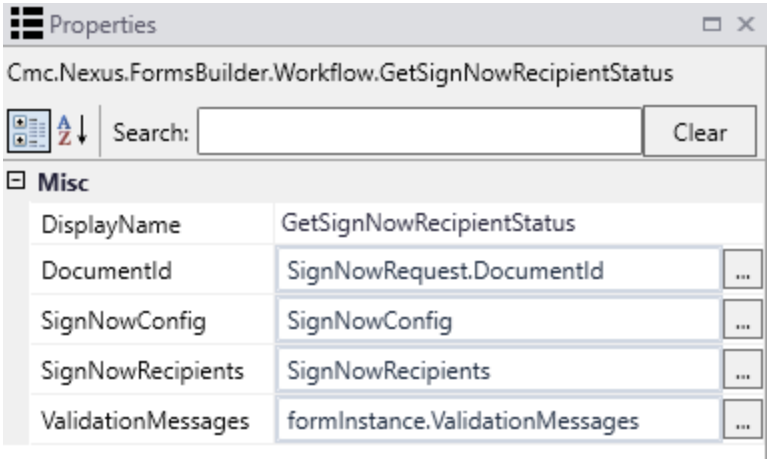
Information about the SignNow signature status is useful to handle all multi route workflows (e.g., retry, completed, denied) and to recover from error conditions including connection loss. For example, when the connection to the SignNow server is interrupted and the signature status indicates that the document was not signed successfully, the user may be prompted to retry the sequence.

The GetSignNowRecipientStatus activity supports the following statuses:

SignNow Recipient Statuses

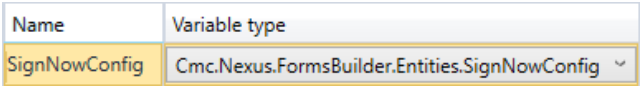
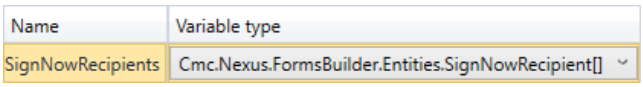
Status	Description
fulfilled	Invite has been completed by the signer.
pending	Invite has been sent to the signer but has not yet been completed.
created	Invite has not yet been sent to the signer, i.e., invite is in order 2, and order 1 invites have not been completed yet (are in pending status).

Properties



GetSignNowRecipientStatus Properties

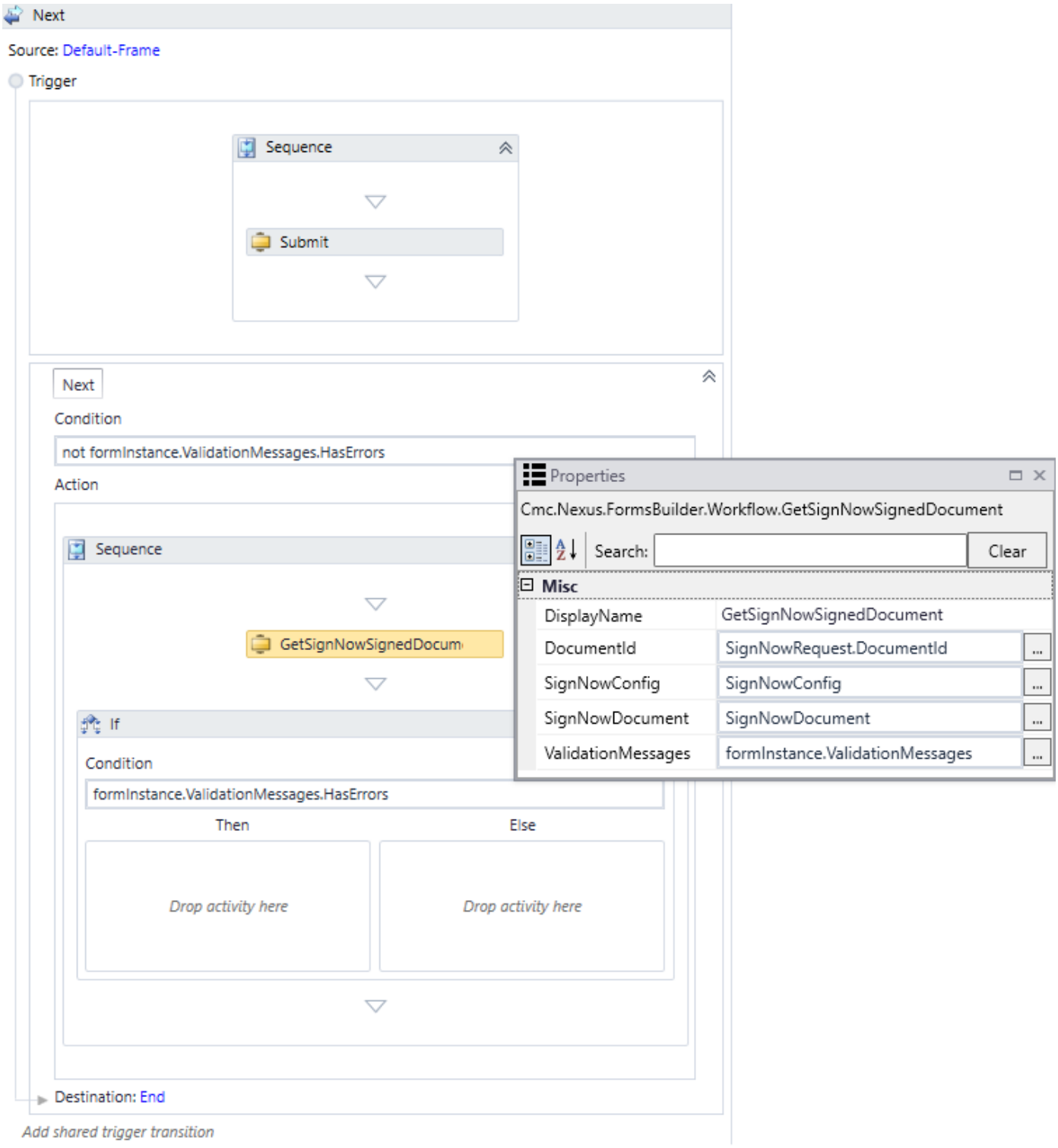
Property	Value	Required	Notes
DisplayName	`String`	No	Specify a name for the activity or accept the default.

Property	Value	Required	Notes
DocumentId	`InArgument<String>`	Yes	Specify the Document Id using a VB expression or variable.
SignNowConfig	`InArgument<SignNowConfig>`	Yes	Specify the SignNowConfig property using a VB expression or variable. This is the out argument returned by the GetSignNowConfig activity. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **SignNowConfig**, and select **OK**. 
SignNowRecipients	`OutArgument<SignNowRecipient[]>`	Yes	Specify the SignNowRecipients array property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, select **Array of [T]**, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **SignNowRecipient**, and select **OK**. 
ValidationMessages	`InOutArgument <ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

GetSignNowSignedDocument

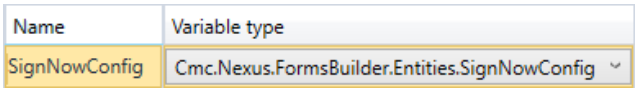
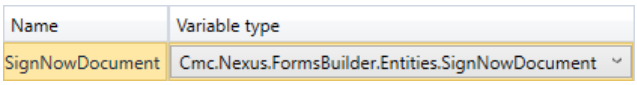
The GetSignNowSignedDocument activity retrieves a signed document from SignNow. The activity passes a Document Id and receives the document for the passed Document Id in the response.

The GetSignNowSignedDocument activity should be placed in the Trigger area of the transition immediately following the IFrame (Default-Frame) that will hold the signed document.


Next
Source: Default-Frame
Trigger
Sequence
Submit
Next
Condition
not formInstance.ValidationMessages.HasErrors
Action
Sequence
GetSignNowSignedDocum
If
Condition
formInstance.ValidationMessages.HasErrors
Then
Else
Drop activity here
Drop activity here
Destination: End
Add shared trigger transition
Properties
Cmc.Nexus.FormsBuilder.Workflow.GetSignNowSignedDocument
Search:
Clear
Misc
DisplayName
GetSignNowSignedDocument
DocumentId
SignNowRequest.DocumentId
SignNowConfig
SignNowConfig
SignNowDocument
SignNowDocument
ValidationMessages
formInstance.ValidationMessages

Properties

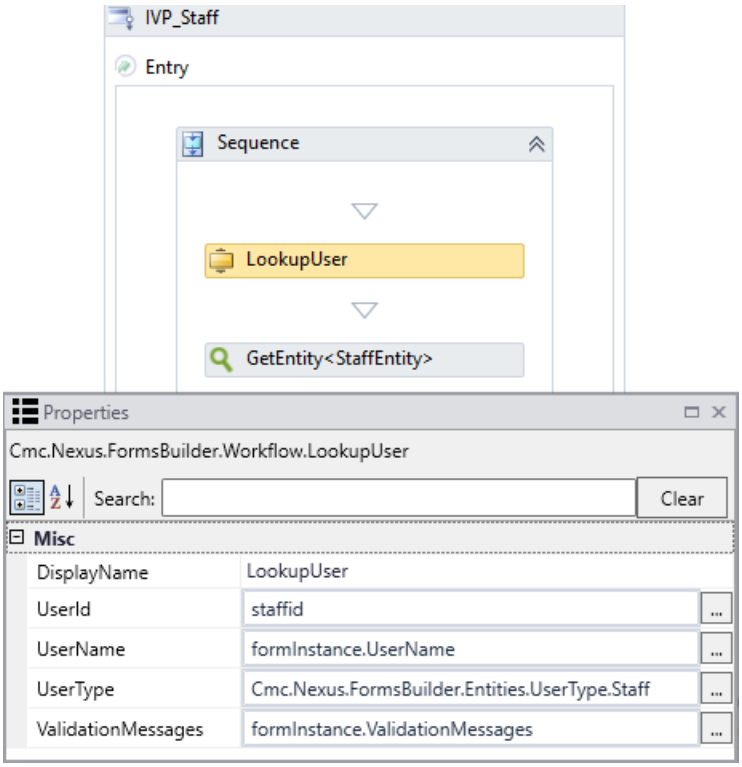
GetSignNowSignedDocument Properties

Property	Value	Required	Notes
DisplayName	`String`	No	Specify a name for the activity or accept the default.
DocumentId	`InArgument<String>`	Yes	Specify the DocumentId as SignNowRequest.DocumentId.
SignNowConfig	`InArgument<SignNowConfig>`	Yes	Specify the SignNowConfig property using a VB expression or variable. This is the out argument returned by the GetSignNowConfig activity. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **SignNowConfig**, and select **OK**. 
SignNowDocument	`OutArgument<SignNowDocument>`	Yes	Specify the SignNowDocument property using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, navigate to **Cmc.Nexus.FormsBuilder.Contracts.Cmc.Nexus.FormsBuilder.Entities**, select **SignNowDocument**, and select **OK**. 
ValidationMessages	`InOutArgument<ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

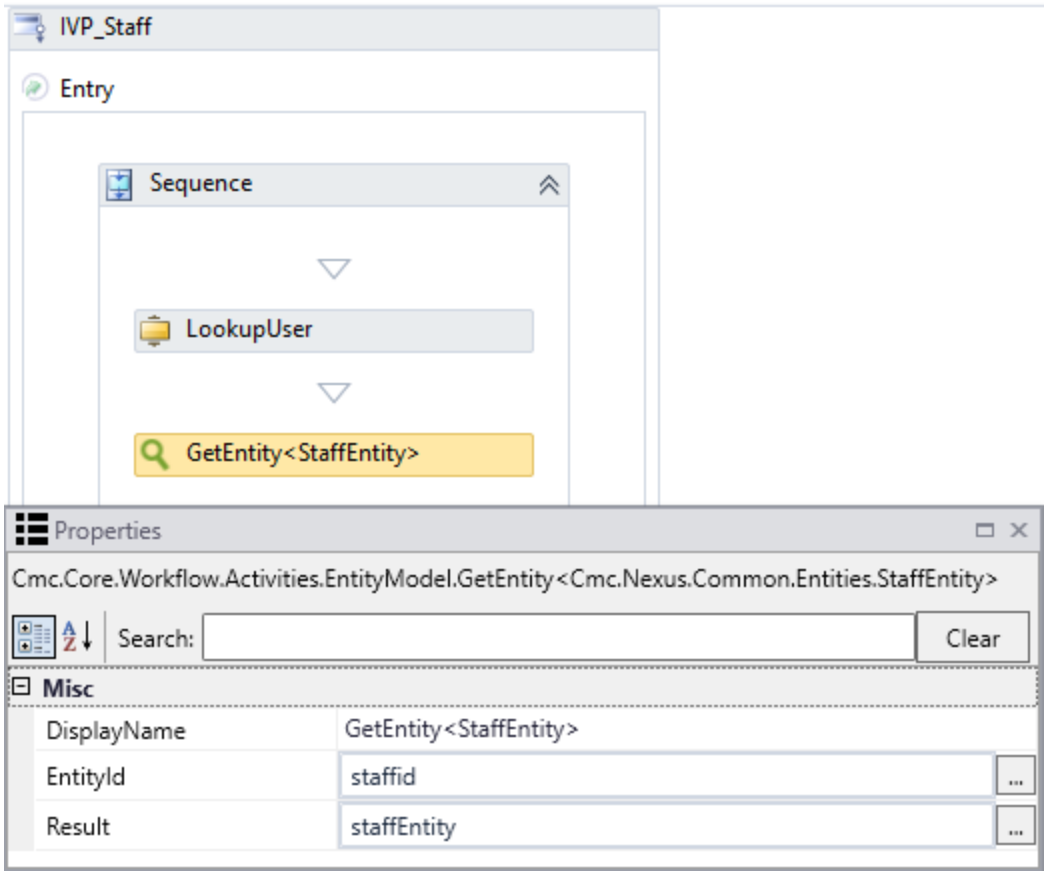
LookupUser

The LookupUser activity takes the UserName from formInstance.UserName and returns the Student Id or Staff Id in the UserId property.

- If a UserType value of 'Staff' is passed in the activity, a Staff Id is returned.
- If a blank UserType (default) is passed, a Student Id is returned.



The UserId is used by the GetEntity<> activity for the StudentEntity or StaffEntity to prepopulate forms where StudentEntity or StaffEntity fields are used.



Optionally, you can add an If activity to the workflow to check for a valid Staff Id. For example, specify the condition "staffid=0" and add a CreateValidationItem activity to the Then branch with a message like "This staff user was not found."

Properties

LookupUser Properties

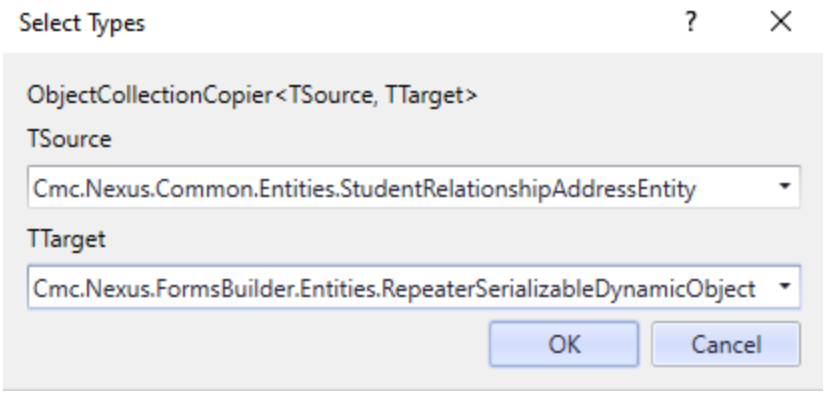
Property	Value	Required	Notes
DisplayName	`String`	No	Specify a name for the activity or accept the default.
UserId	`OutArgument<Int32>`	Yes	Specify the UserId using a VB expression or variable.
UserName	`InArgument<String>`	Yes	Specify the UserName using a VB expression or variable. The property `formInstance.UserName` contains the value for the UserName.

Property	Value	Required	Notes
UserType	`InArgument<UserType>`	No	Specify the UserType using a VB expression or variable. The namespace Cmc.Nexus.FormsBuilder.Entities.UserType provides the option to select 'Staff' or 'Student' as the UserType. When the UserType property is left blank, the default UserType is Student to ensure backward compatibility. If a staff sequence is accessed via cloud services (Azure), you must include a LookupUser activity with UserType=Staff in the workflow to ensure proper authentication and authorization for the Staff role. For more information, refer to Azure AD Authentication.
ValidationMessages	`InOutArgument<ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages.

ObjectCollectionCopier<>

The ObjectCollectionCopier<> workflow activity copies one object array (source) to another object array (target) without having to iterate through the properties of the objects within the array. The activity copies all properties that match in Name and Type to another object array. The most common reason for using this activity is to copy items out of a dynamic Repeater or Grid to be saved to the database.

When you drag the ObjectCollectionCopier<> into the Designer pane, you are prompted to select the TSource and TTarget.



The object Types for selection include:

- Boolean
- Int32 (default)
- String
- Object
- Array of (T)
- **Browse for Types...**

The only valid choice here is **Browse for Types**.

You should choose:

- a class with properties,
- an entity,
- a SerializableDynamicObject, or
- a RepeaterSerializableDynamicObject.

The activity already assumes an array and you just need to select the type of object in the source and target arrays.

Type of Source and Target Objects	Description
Same type	This is typically an entity in the Anthology object model, but it can be any class with read/write properties, a SerializableDynamicObject, or a RepeaterSerializableDynamicObject.

Type of Source and Target Objects	Description
Different types	All properties that have the same Type and Name will be copied by the activity.
SerializableDynamicObject	As these are dynamic objects. When writing them, the Dictionary defined properties will be created by the activity. When reading them, all properties defined in their Dictionary will be copied.
RepeaterSerializableDynamicObject	

Once the TSource and TTarget are selected, the header of properties pane for the ObjectCollectionCopier<> activity displays the selected objects. You can now specify the source and target arguments to be used by the activity. For an ObjectCollectionCopier<>, the type of argument must be an **array** of the TSource or TTarget type specified.

The ObjectCollectionCopier<> activity also provides the option to exclude or include some properties in comma separated lists. By default, all properties that can be copied (they exist in both source and target and are the same name and type) will be copied.

- If IncludedProperties are specified, only those properties will be copied (if they exist in both the source and the target).
- If ExcludedProperies are specified, all properties not excluded will be copied. Often an Id will be excluded.
- If both are specified, ExcludedProperties is ignored.

The ObjectCollectionCopier<> will only “new up” all target objects. It does not do a CreateEntity on each object.

⚠ This can cause issues when the database has field restrictions, for example, with NULL vs. empty string. Therefore, entity type objects may need additional changes/assignments after the ObjectCollectionCopier<> activity is executed.

Properties

ObjectCollectionCopier<> Properties

Property	Value	Required	Notes
Display Name	`String`	No	Specify a name for the activity or accept the default. By default, the Display Name is "ObjectCollectionCopier<Source Object Name,Target Object Name>".
Excluded Properties	`String`	No	Specify a comma separated list of properties that won't be copied. By default, no properties will be excluded. ExcludedProperties will be ignored if IncludedProperties is specified.
Included Properties	`String`	No	Specify a comma separated list of properties to include when copying. By default, all properties that can be copied (exist, and match name and type) will be copied. ExcludedProperties will be ignored if IncludedProperties is specified.

Property	Value	Required	Notes
Source Object Array	`InOutArgument<>`	Yes	Specify an argument name of an object array to copy. The argument type must be an array of the type selected for TSource.
Target Object Array	`InOutArgument<>`	Yes	Specify the target argument array to copy the source to. The argument type must be an array of the type selected for TTarget.

Using SaveEntity vs. SaveEntityCollection with ObjectCollectionCopier<>

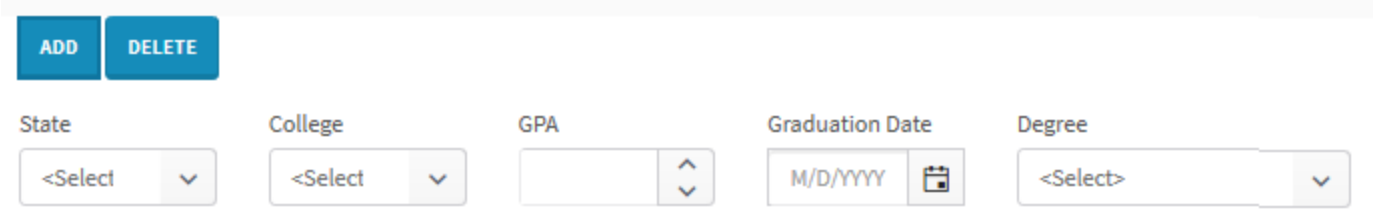
The examples below show the ObjectCollectionCopier<> activity in two different scenarios: one case uses SaveEntity to save the copied object collection, the other uses SaveEntityCollection. The reason is that the SaveEntityCollection is not usable all the time because certain defaults in the Entity collection created by the ObjectCollectionCopier<> will not allow it to be saved. The ObjectCollectionCopier<> will "new up" all entities, and they will have default values. For instance, a phone number that is null might not be savable because it is required to be an empty string in the database if not specified.

For example, if additional assignments are needed for fields not usually filled in by a user on the form (e.g., studentid) and the workflow has to loop through the array anyway, you can use a SaveEntity activity in each loop.

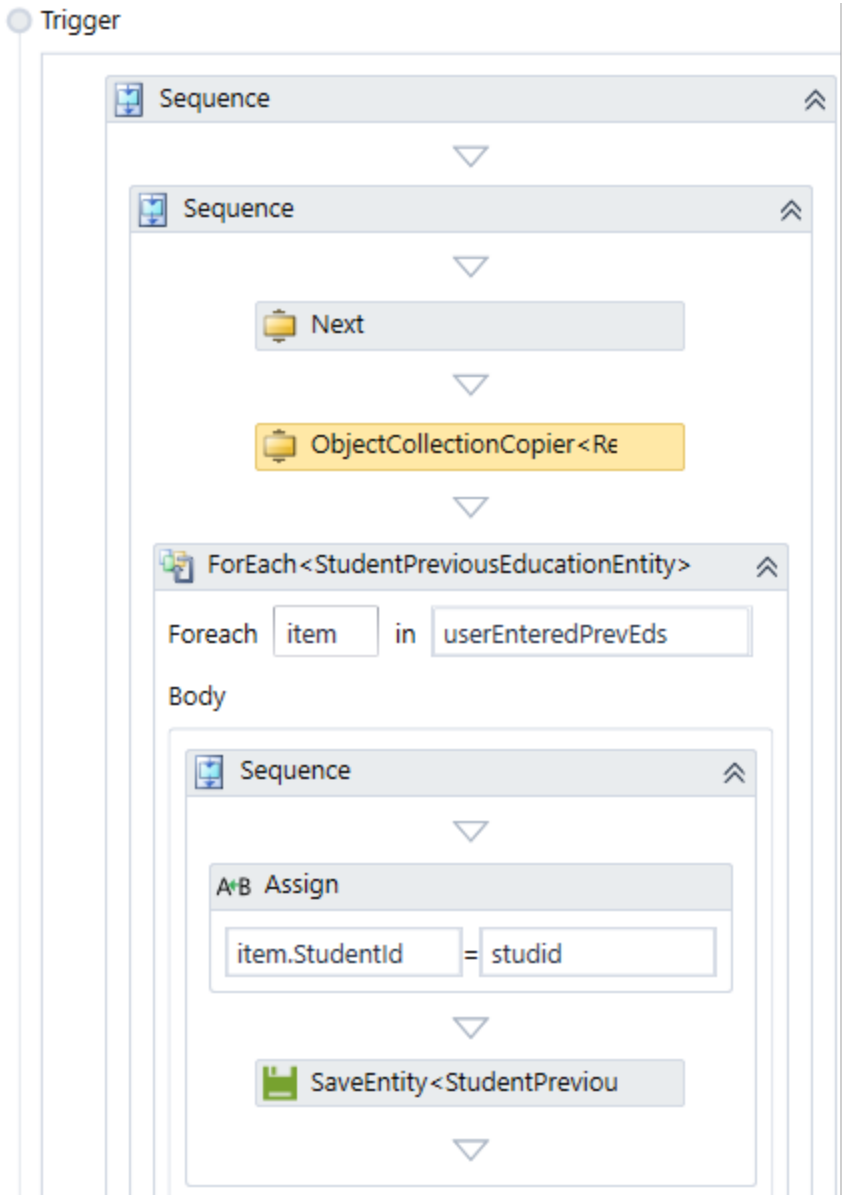
If all necessary values required to save an entity are filled in on the form, then the workflow just needs a single SaveEntityCollection activity after the ObjectCollectionCopier<>.

Example: RepeaterSerializableDynamicObject with SaveEntity

A form sequence collects information about a student's previous education. The sequence contains a Repeater control with the following layout.



The ObjectCollectionCopier<> activity is added to the initial workflow to copy the RepeaterSerializableDynamicObject[] to a specific StudentPreviousEducationEntity and save it. With the ObjectCollectionCopier<> there is no need to go through each property and use data dictionary keys to copy property values.



The Next transition before the End state has a Sequence containing the following activities:

- WaitForFormBookmark (Next)
- ObjectCollectionCopier<RepeaterSerializableDynamicObject,StudentPreviousEducationEntity>
    - TSource = Cmc.Nexus.FormsBuilder.Entities.RepeaterSerializableDynamicObject
    - TTarget = Cmc.Nexus.Admissions.Entities.StudentPreviousEducationEntity
    - Source Object Array = myPrevEd
    - Target Object Array = userEnteredPrevEds

    The source object array is an instance of the RepeaterSerializableDynamicObject array with the "myPrevEd" value.

The target object array "userEnteredPrevEds" is created and initialized by the ObjectCollectionCopier<>.

Note that the ObjectCollectionCopier<> is placed before the ForEach activity.

- ForEach<StudentPreviousEducationEntity>
    - TypeArgument = Cmc.Nexus.Admissions.Entities.StudentPreviousEducationEntity
    - Values = myPrevEds

  The Body of the ForEach activity contains a Sequence with Assign and SaveEntity<> activities.

- Assign
    - To = item.StudentId
    - Value = studid (variable created by the user)

- SaveEntity<StudentPreviousEducationEntity>
    - Entity = item
    - ValidationMessages = formInstance.ValidationMessages

The variables in this workflow are:

Name	Variable type	Scope	Default
renderedFormImage	String	StateMachine	*Enter a VB expression*
userEnteredPrevEds	StudentPreviousEducationEntity[]	StateMachine	*Enter a VB expression*
studid	Int32	StateMachine	*Enter a VB expression*

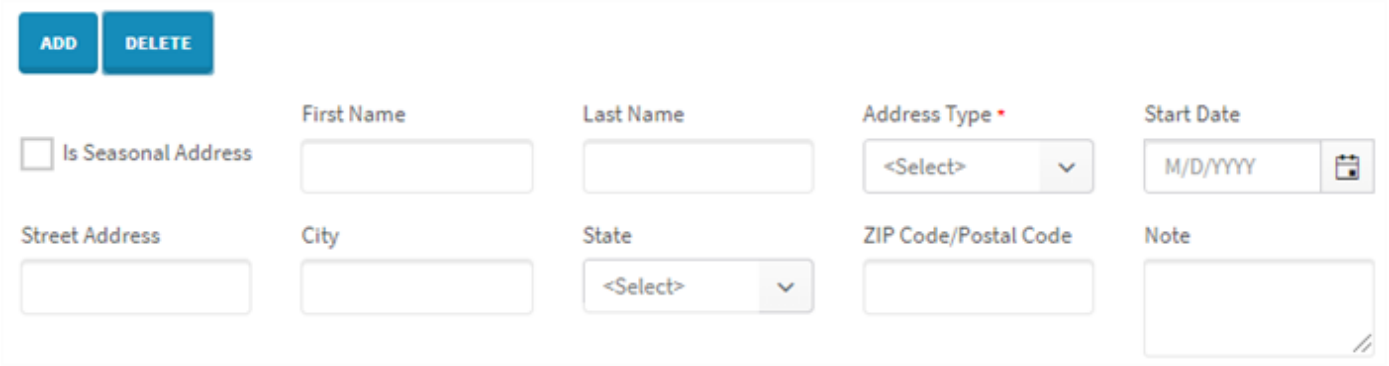
The arguments created by Forms Builder are:

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*
myPrevEd	In/Out	RepeaterSerializableDynamicObject[]	*Default value not supported*
_myPrevEd_State	In/Out	String	*Default value not supported*
_myPrevEd_Gpa	In/Out	Nullable<Decimal>	*Default value not supported*
_myPrevEd_DegreeId	In/Out	Nullable<Int32>	*Default value not supported*
_myPrevEd_CollegeId	In/Out	Nullable<Int32>	*Default value not supported*
_myPrevEd_GraduationDate	In/Out	Nullable<DateTime>	*Default value not supported*

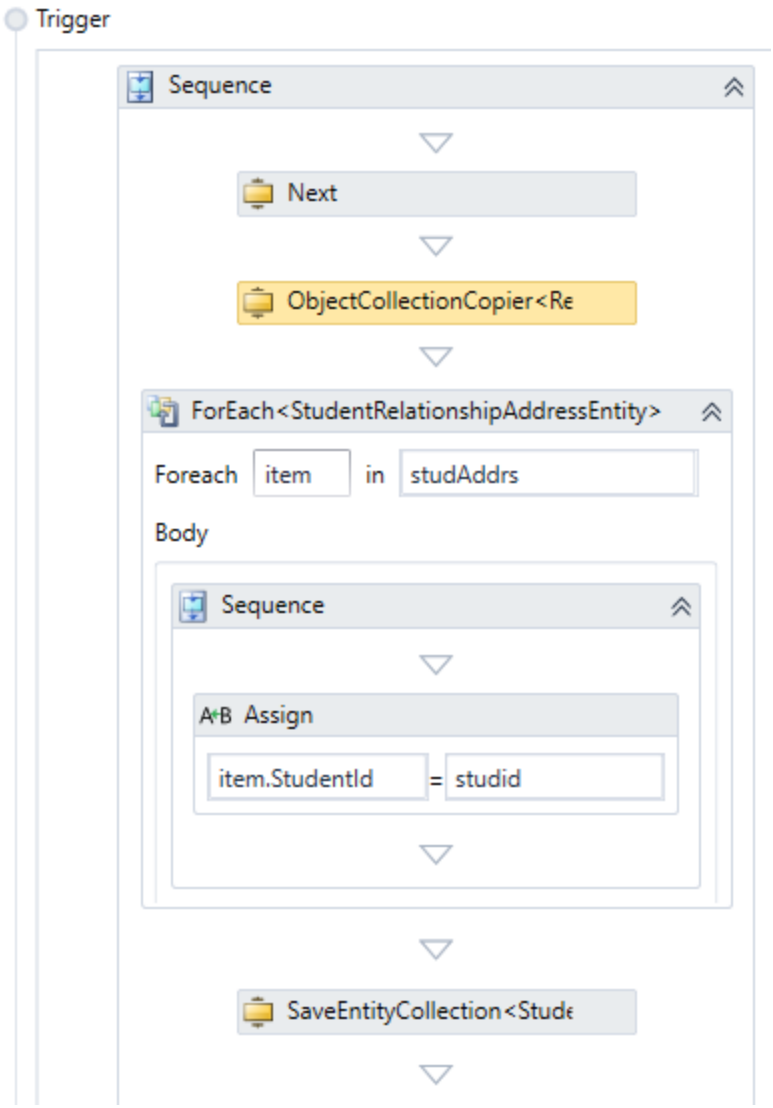
The dummy underscore arguments are automatically assigned when a workflow with Repeater control is initially created. For more information, refer to Repeater Workflow Arguments.

Example: RepeaterSerializableDynamicObject with SaveEntityCollection

A form sequence collects information about a student's addresses. The sequence contains a Repeater control with the following layout.



The ObjectCollectionCopier<> activity is added to the initial workflow to copy the RepeaterSerializableDynamicObject array to a StudentRelationshipAddressEntity and save it. With the ObjectCollectionCopier<> there is no need to go through each property to copy property values.



The Next transition before the End state has a Sequence containing the following activities:

- WaitForFormBookmark (Next)
- ObjectCollectionCopier<RepeaterSerializableDynamicObject,StudentRelationshipAddressEntity>>
    - TSource = Cmc.Nexus.FormsBuilder.Entities.RepeaterSerializableDynamicObject
    - TTarget = Cmc.Nexus.Common.Entities.StudentRelationshipAddressEntity
    - Source Object Array = myAddrs
    - TargetObject Array = studAddrs

  The source object array is an instance of the RepeaterSerializableDynamicObject with the "myAddrs" value.

  The target object array "studAddrs" is created and initialized by the ObjectCollectionCopier<>.

  Note that the ObjectCollectionCopier<> is placed before the ForEach activity.

- ForEach<StudentRelationshipAddressEntity>
    - TypeArgument = Cmc.Nexus.Common.Entities.StudentRelationshipAddressEntity
    - Value = studAddrs

  The Body of the ForEach activity contains an Assign activity for the student Id variable.

- SaveEntityCollection<StudentPreviousEducationEntity>
    - Entities = studAddrs

  The SaveEntityCollection activity saves the target object array "studAddrs" including the assignment in the ForEach loop to the database.

The variables in this workflow are:

Name	Variable type	Scope	Default
renderedFormImage	String	StateMachine	*Enter a VB expression*
studAddrs	StudentRelationshipAddressEntity[]	StateMachine	*Enter a VB expression*
studid	Int32	StateMachine	*Enter a VB expression*

The arguments created by Forms Builder are:

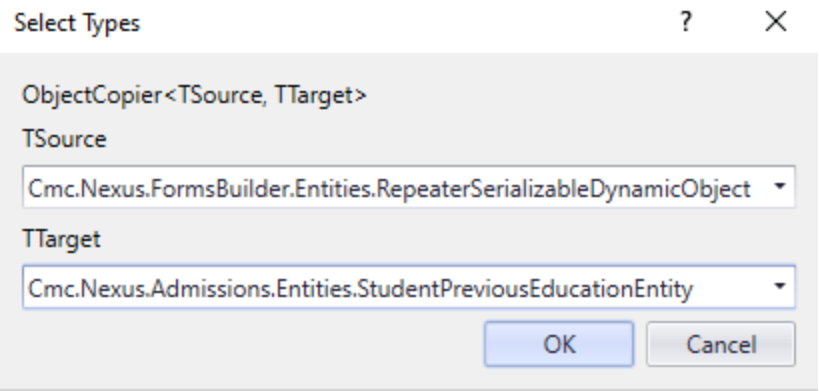
Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*
bottomRequired	In/Out	Boolean	*Default value not supported*
myAddrs	In/Out	RepeaterSerializableDynamicObject[]	*Default value not supported*
_myAddrs_FirstName	In/Out	String	*Default value not supported*
_myAddrs_LastName	In/Out	String	*Default value not supported*
_myAddrs_Note	In/Out	String	*Default value not supported*
_myAddrs_City	In/Out	String	*Default value not supported*
_myAddrs_StreetAddress	In/Out	String	*Default value not supported*
_myAddrs_State	In/Out	String	*Default value not supported*
_myAddrs_PostalCode	In/Out	String	*Default value not supported*
_myAddrs_IsSeasonalAddress	In/Out	Boolean	*Default value not supported*
_myAddrs_AddressTypeId	In/Out	Nullable<Int32>	*Default value not supported*
_myAddrs_AddressBeginDate	In/Out	Nullable<DateTime>	*Default value not supported*

The dummy underscore arguments are automatically assigned when a workflow with Repeater control is initially created. For more information, refer to Repeater Workflow Arguments.

ObjectCopier<>

The ObjectCopier<> workflow activity copies one object (source) to another object (target) without having to iterate through the object properties. The activity copies all properties that match in Name and Type to another object.

When you drag the ObjectCopier<> into the Designer pane, you are prompted to select the TSource and TTarget.



The object types for selection include:

- Boolean
- Int32 (default)
- String
- Object
- Array of (T)
- Browse for Types...

The only valid choice here is **Browse for Types**.

You should choose:

- a class with properties,
- an entity,
- a SerializableDynamicObject, or
- a RepeaterSerializableDynamicObject.

Type of Source and Target Objects	Description
Same type	This is typically an entity in the Anthology object model, but it can be any class with read/write properties, a SerializableDynamicObject, or a RepeaterSerializableDynamicObject.
Different types	All properties that have the same Type and Name will be copied by the activity.

Type of Source and Target Objects	Description
SerializableDynamicObject	As these are dynamic objects. When writing them, the Dictionary defined properties will be created by the activity. When reading them, all properties defined in their Dictionary will be copied.
RepeaterSerializableDynamicObject	

Once the TSource and TTarget are selected, the header of properties pane for the ObjectCopier<> activity displays the selected objects. You can now specify the source and target arguments to be used by the activity. For an ObjectCopier<>, the argument type must match the TSource or TTarget type.

The ObjectCopier<> activity also provides the option to exclude or include some properties in comma separated lists. By default, all properties that can be copied (they exist in both source and target and are the same name and type) will be copied.

- If IncludedProperties are specified, only those properties will be copied (if they exist in both the source and the target)
- If ExcludedProperies are specified, all properties not excluded will be copied. Often an Id will be excluded.
- If both are specified, ExcludedProperties is ignored.

If an existing object is populated with data and a user wants to update that object by means of another object using ObjectCopier<>, **all** fields in the object (source) will be copied to the target, even if the fields are empty.

For example, a form loads an editable grid with address data fields and some fields are updated, while others are left blank. If the ObjectCopier<> is used to copy the updated grid to another object, the target resulting from object copy will have the updated fields and the blank fields. The ObjectCopier<> does not check for changed values. It does not selectively copy fields that were updated. The result is that if you save this data to the database, you will be changing information you did not necessarily want to change.

To avoid this situation, make sure you are not creating a new target object. If it is first retrieved from the database and exists with the current data and you only want to update certain fields, use the Included Properties, and list them. That way the current data you do not want changed will not be changed when you save the entity.

Properties

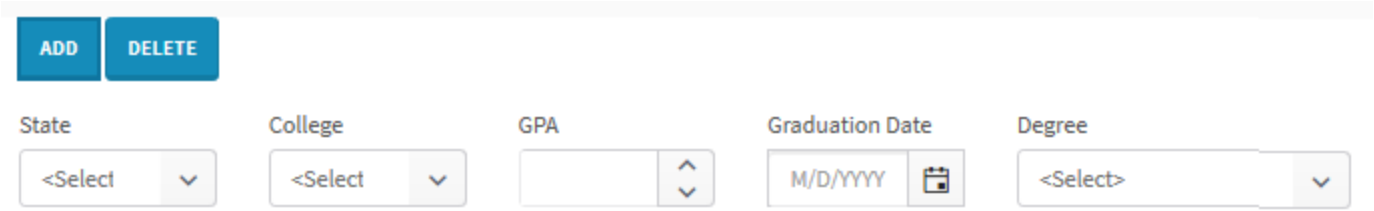
ObjectCopier<> Properties

Property	Value	Required	Notes
Display Name	`String`	No	Specify a name for the activity or accept the default. By default, the Display Name is "ObjectCopier<Source Object Name,Target Object Name>".
Excluded Properties	`String`	No	Specify a comma separated list of properties that won't be copied. By default, no properties will be excluded. ExcludedProperties will be ignored if IncludedProperties is specified.

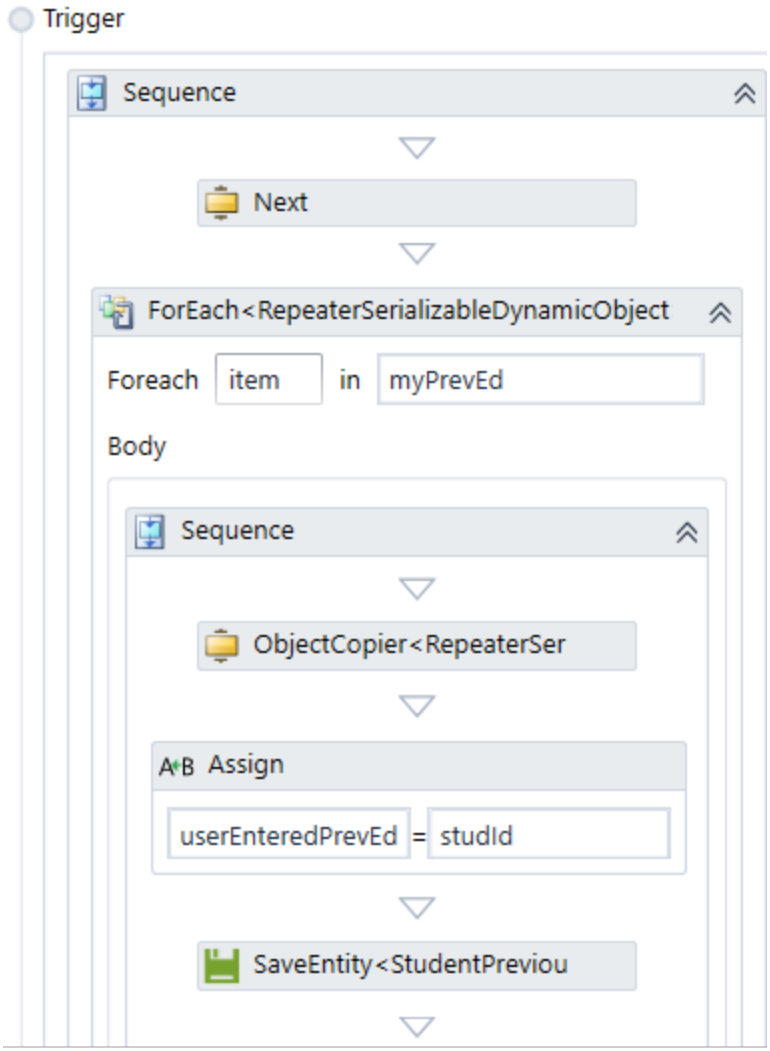
Property	Value	Required	Notes
Included Properties	`String`	No	Specify a comma separated list of properties to include when copying. By default, all properties that can be copied (exist, and match name and type) will be copied. ExcludedProperties will be ignored if IncludedProperties is specified.
Source Object	`InOutArgument<>`	Yes	Specify an argument name of an object to copy.
Target Object	`InOutArgument<>`	Yes	Specify the target argument to copy the source to.

Example: RepeaterSerializableDynamicObject

A form sequence collects information about a student's previous education. The sequence contains a Repeater control with the following layout.



The ObjectCopier<> activity is added to the initial workflow to copy the RepeaterSerializableDynamicObject[] to a specific StudentPreviousEducationEntity and save it. With the ObjectCopier<> there is no need to go through each property and use data dictionary keys to copy property values.



The Next transition before the End state has a Sequence containing the following activities:

- WaitForFormBookmark (Next)
- ForEach<RepeaterSerializableDynamicObject>
    - TypeArgument = Cmc.Nexus.FormsBuilder.Entities.RepeaterSerializableDynamicObject
    - Value = myPrevEd

  The Body of the ForEach activity contains a Sequence with ObjectCopier<>, Assign, and SaveEntity<> activities.
- ObjectCopier<RepeaterSerializableDynamicObject,StudentPreviousEducationEntity>
    - TSource = Cmc.Nexus.FormsBuilder.Entities.RepeaterSerializableDynamicObject
    - TTarget = Cmc.Nexus.Admissions.Entities.StudentPreviousEducationEntity
    - Source Object = item
    - Target Object = userEnteredPrevEd

The source object "item" is an instance of the RepeaterSerializableDynamicObject with the "myPrevEd" value.

The target object "userEnteredPrevEd" is created and initialized by the ObjectCopier<>.

- Assign
    - To = userEnteredPrevEd.StudentId
    - Value = studId (variable created by the user)
- SaveEntity<StudentPreviousEducationEntity>
    - Entity = userEnteredPrevEd
    - ValidationMessages = formInstance.ValidationMessages

The variables in this workflow are:

Name	Variable type	Scope	Default
studId	Int32	Sequence	*Enter a VB expression*
renderedFormImage	String	StateMachine	*Enter a VB expression*
userEnteredPrevEd	StudentPreviousEducationEntity	StateMachine	*Enter a VB expression*

The arguments created by Forms Builder are:

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*
myPrevEd	In/Out	RepeaterSerializableDynamicObject[]	*Default value not supported*
_myPrevEd_State	In/Out	String	*Default value not supported*
_myPrevEd_Gpa	In/Out	Nullable<Decimal>	*Default value not supported*
_myPrevEd_DegreeId	In/Out	Nullable<Int32>	*Default value not supported*
_myPrevEd_CollegeId	In/Out	Nullable<Int32>	*Default value not supported*
_myPrevEd_GraduationDate	In/Out	Nullable<DateTime>	*Default value not supported*

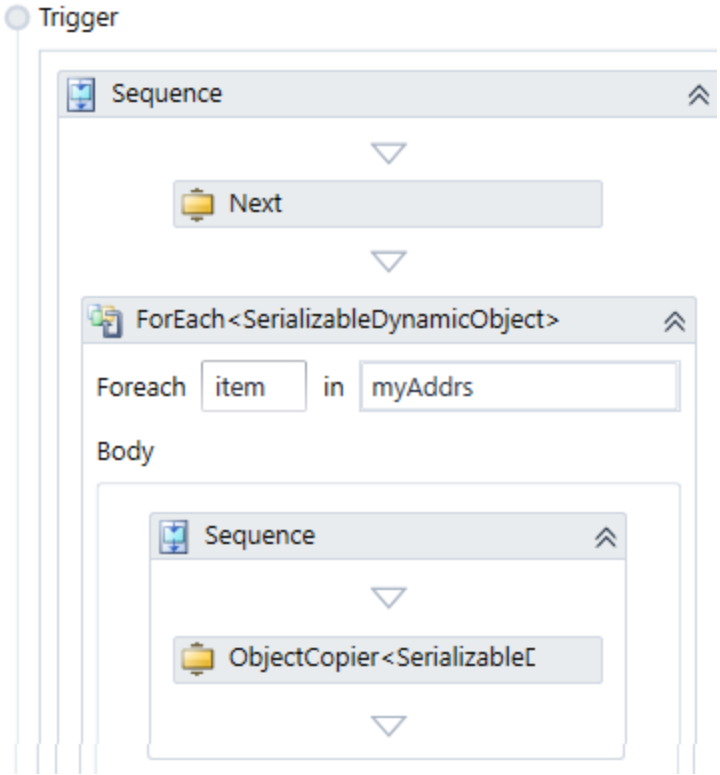
The dummy underscore arguments are automatically assigned when a workflow with Repeater control is initially created. For more information, refer to Repeater Workflow Arguments.

Example: SerializableDynamicObject

A form sequence collects information about a student's addresses. The sequence contains a Grid control with the following layout.

FirstName	LastName	Note	City
Joes	Minoj	My note	Miami
Candy	Jones	My note 2	Boca Raton
Missy	Grissom	My note 3	Tamarac

The ObjectCopier<> activity is added to the initial workflow to copy the SerializableDynamicObject[] to a StudentRelationshipAddressEntity and save it. With the ObjectCopier<> there is no need to go through each property to copy property values.



The Next transition before the End state has a Sequence containing the following activities:

- WaitForFormBookmark (Next)
- ForEach<SerializableDynamicObject>
    - TypeArgument = Cmc.Nexus.FormsBuilder.Entities.SerializableDynamicObject
    - Value = myAddrs

The Body of the ForEach activity contains a Sequence with ObjectCopier<> activity.

- ObjectCopier<SerializableDynamicObject,StudentRelationshipAddressEntity>
    - TSource = Cmc.Nexus.FormsBuilder.Entities.SerializableDynamicObject
    - TTarget = Cmc.Nexus.Common.Entities.StudentRelationshipAddressEntity
    - Source Object = item
    - Target Object = studAddr

  The source object "item" is an instance of the SerializableDynamicObject with the "myAddrs" value.

  The target object "stuAddr" is created and initialized by the ObjectCopier<>.

The variables in this workflow are:

Name	Variable type	Scope	Default
renderedFormImage	String	StateMachine	*Enter a VB expression*
studAddr	StudentRelationshipAddressEntity	StateMachine	*Enter a VB expression*

The arguments created by Forms Builder are:

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event	In/Out	ConstructedEvent	*Default value not supported*
bottomRequired	In/Out	Boolean	*Default value not supported*
myAddrs	In/Out	SerializableDynamicObject[]	*Default value not supported*

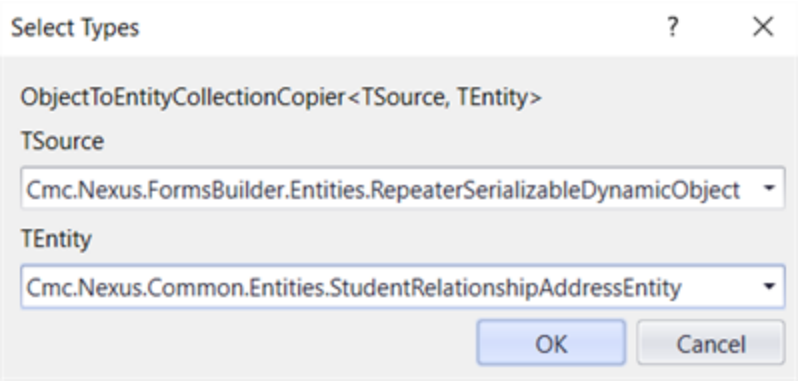
ObjectToEntityCollectionCopier<>

The ObjectToEntityCollectionCopier<> workflow activity copies any object collection whose property names and types match to an **Entity** object collection. The most common reason for using this activity is to copy items out of a dynamic Repeater or Grid to be saved to the database.

The difference between ObjectToEntityCollectionCopier<> and ObjectCollectionCopier<> is that ObjectCollectionCopier only does "New" on an object. It does not use the CreateEntity constructor method for the target entity. ObjectToEntityCollectionCopier<> uses the CreateEntity constructor method and handles specifics based on database settings that a basic "New" will not handle (such as Id needing to be -1).

Since the ObjectToEntityCollectionCopier<> activity creates all necessary values required to save an entity, the workflow just needs a single SaveEntityCollection activity after the ObjectToEntityCollectionCopier<> as shown in the example below.

When you drag the ObjectToEntityCollectionCopier<> into the Designer pane, you are prompted to select the TSource and TEntity.



The object Types for selection include:

- Boolean
- Int32 (default)
- String
- Object
- Array of (T)
- **Browse for Types...**

The only valid choice here is **Browse for Types**.

You should choose:

- a class with properties,
- an entity,
- a SerializableDynamicObject, or
- a RepeaterSerializableDynamicObject

TEntity must be a class derived from the base type Entity.

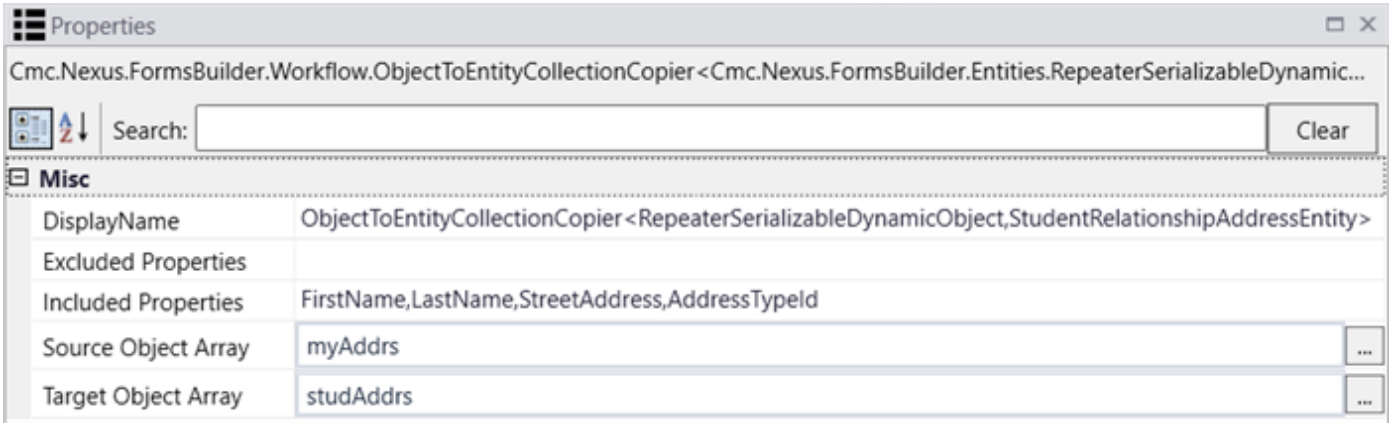
The activity already assumes an array and you just need to select the type of object in the source and target arrays.

- The type of the **source** objects arrays can be Entity, SerializableDynamicObject, or RepeaterSerializableDynamicObject.
- The **target** objects arrays can only be Entity types.

Once the TSource and TEntity are selected, the header of properties pane for the ObjectToEntityCollectionCopier<> activity displays the selected objects. You can now specify the source and target arguments to be used by the activity. For an ObjectToEntityCollectionCopier<>, the type of argument must be an **array** of the TSource or TEntity type specified.

The ObjectToEntityCollectionCopier<> activity also provides the option to exclude or include some properties in comma separated lists. By default, all properties that can be copied (they exist in both source and target and are the same name and type) will be copied.

- If IncludedProperties are specified, only those properties will be copied (if they exist in both the source and the target).
- If ExcludedProperies are specified, all properties not excluded will be copied. Often an Id will be excluded.
- If both are specified, ExcludedProperties is ignored.



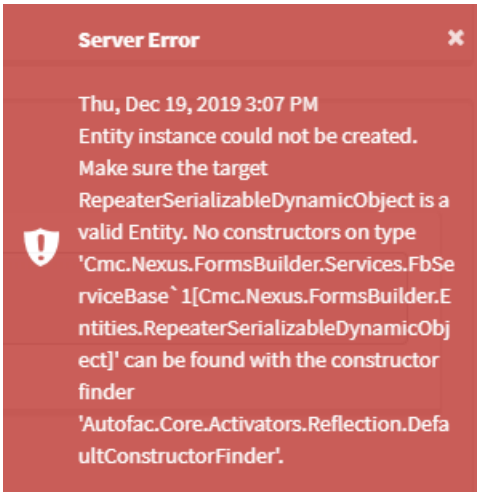
Properties

ObjectToEntityCollectionCopier<> Properties

Property	Value	Required	Notes
Display Name	`String`	No	Specify a name for the activity or accept the default. By default, the Display Name is "ObjectToEntityCollectionCopier<Source Object Name,Target Object Name>".
Excluded Properties	`String`	No	Specify a comma separated list of properties that won't be copied. By default, no properties will be excluded. ExcludedProperties will be ignored if IncludedProperties is specified.

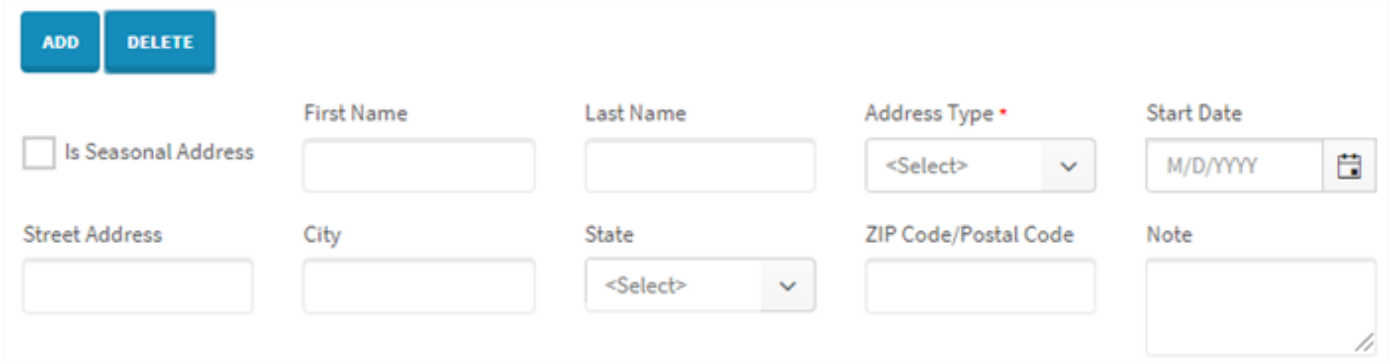
Property	Value	Required	Notes
Included Properties	`String`	No	Specify a comma separated list of properties to include when copying. By default, all properties that can be copied (exist, and match name and type) will be copied. ExcludedProperties will be ignored if IncludedProperties is specified.
Source Object Array	`InOutArgument<>`	Yes	Specify an argument name of an object array to copy. The argument type must be an array of the type selected for TSource.
Target Object Array	`InOutArgument<>`	Yes	Specify the target argument array to copy the source to. The argument type must be an array of the type selected for TEntity.

If you specified a target object that is not an entity, an error similar to the following will be displayed in Forms Renderer at runtime:

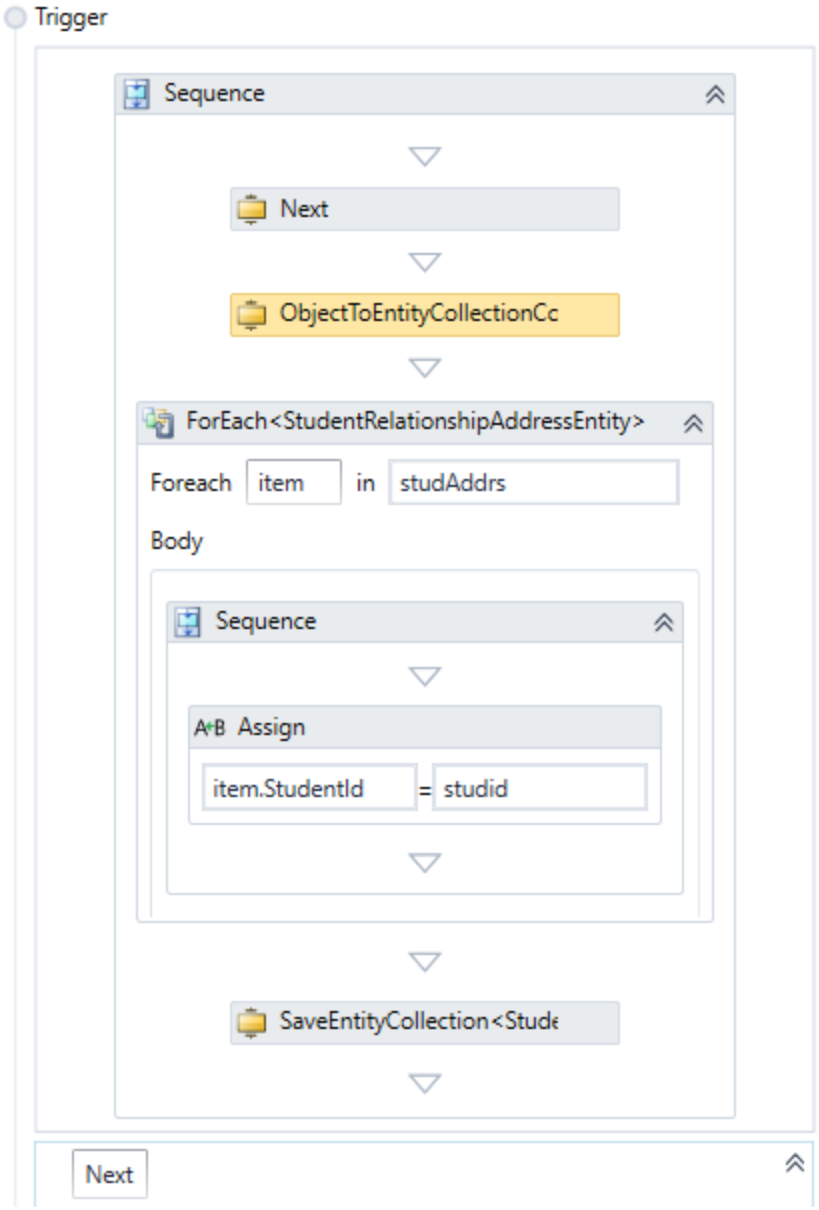


Example: RepeaterSerializableDynamicObject with SaveEntityCollection

A form sequence collects information about a student's addresses. The sequence contains a Repeater control with the following layout.



The ObjectToEntityCollectionCopier<> activity is added to the initial workflow to copy the RepeaterSerializableDynamicObject array to a StudentRelationshipAddressEntity and save it. With the ObjectToEntityCollectionCopier<> there is no need to go through each property to copy property values.



The Next transition before the End state has a Sequence containing the following activities:

- WaitForFormBookmark (Next)
- ObjectCollectionCopier<RepeaterSerializableDynamicObject,StudentRelationshipAddressEntity>>
    - TSource = Cmc.Nexus.FormsBuilder.Entities.RepeaterSerializableDynamicObject
    - TEntity = Cmc.Nexus.Common.Entities.StudentRelationshipAddressEntity
    - Source Object Array = myAddrs
    - TargetObject Array = studAddrs

The source object array is an instance of the RepeaterSerializableDynamicObject with the "myAddrs" value.

The target object array "studAddrs" is created and initialized by the ObjectToEntityCollectionCopier<>.

Note that the ObjectToEntityCollectionCopier<> is placed before the ForEach activity.

- ForEach<StudentRelationshipAddressEntity>
    - TypeArgument = Cmc.Nexus.Common.Entities.StudentRelationshipAddressEntity
    - Value = studAddrs

  The Body of the ForEach activity contains an Assign activity for the student Id variable.

- SaveEntityCollection<StudentPreviousEducationEntity>
    - Entities = studAddrs

  The SaveEntityCollection activity saves the target object array "studAddrs" including the assignment in the ForEach loop to the database.

The variables in this workflow are:

Name	Variable type	Scope	Default
renderedFormImage	String	StateMachine	*Enter a VB expression*
studAddrs	StudentRelationshipAddressEntity[]	StateMachine	*Enter a VB expression*
studid	Int32	StateMachine	*Enter a VB expression*

The arguments created by Forms Builder are:

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*
bottomRequired	In/Out	Boolean	*Default value not supported*
myAddrs	In/Out	RepeaterSerializableDynamicObject[]	*Default value not supported*
_myAddrs_FirstName	In/Out	String	*Default value not supported*
_myAddrs_LastName	In/Out	String	*Default value not supported*
_myAddrs_Note	In/Out	String	*Default value not supported*
_myAddrs_City	In/Out	String	*Default value not supported*
_myAddrs_StreetAddress	In/Out	String	*Default value not supported*
_myAddrs_State	In/Out	String	*Default value not supported*
_myAddrs_PostalCode	In/Out	String	*Default value not supported*
_myAddrs_IsSeasonalAddress	In/Out	Boolean	*Default value not supported*
_myAddrs_AddressTypeId	In/Out	Nullable<Int32>	*Default value not supported*
_myAddrs_AddressBeginDate	In/Out	Nullable<DateTime>	*Default value not supported*

The dummy underscore arguments are automatically assigned when a workflow with Repeater control is initially created. For more information, refer to Repeater Workflow Arguments.

PrintUrlToPdf

The PrintUrlToPdf activity prints a URL to a PDF file. The URL attribute in this activity is referred to as the "viewCreator" link. The link contains the forms, the Workflow Definition ID, and the RendererBaseUrl. The "viewCreator" feature is available for authenticated and anonymous sequences.

The PrintUrlToPdf activity can be placed in the Action area of a form (state) that requires e-signatures. A **Persist** activity should always precede the PrintUrlToPdf activity.

The PrintUrlToPdf activity allows an empty URL input argument. The activity constructs the URL using:

- formInstance.RendererBaseUrl
- formInstance.WorkflowDefinitionId
- formInstance.FormsTraversed

This addresses the common scenario whereby all forms need to be included in the PDF by eliminating the need to type every form name with perfect casing for the URL. Backward compatibility is maintained by allowing users to create the URL and list of forms for the PDF output.

The Culture argument can be used to create a PDF in a language that is different from the language (culture) selected when the form sequence was rendered. If the Culture argument is not specified, the PDF will be created in the culture selected when the sequence was rendered. For more information, refer to Culture.

Workflow Example

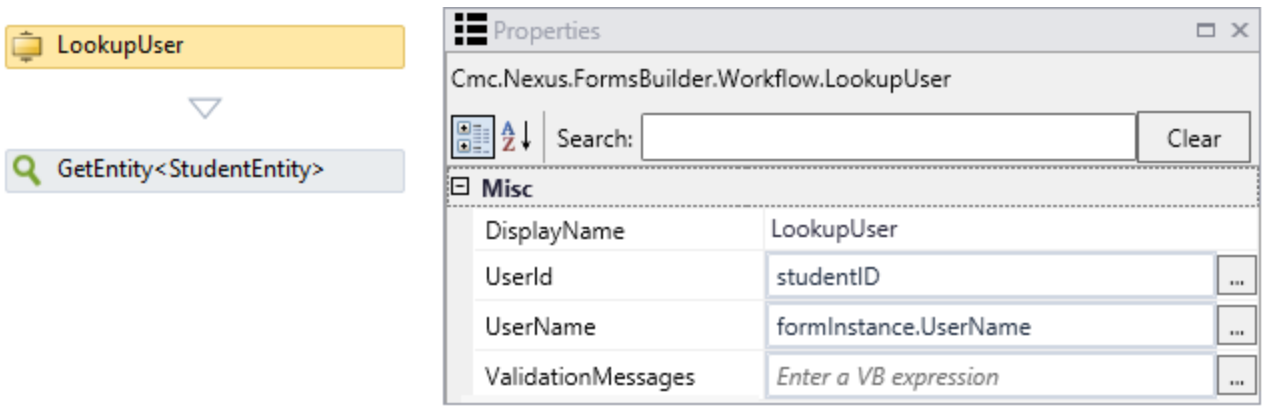
The following images show segments of a workflow in a DocuSign scenario. In this example the Assign activity preceding the PrintUrlToPdf activity assigns the following URL:

formInstance.RendererBaseUrl + "#/viewCreator/" + formInstance.WorkflowDefinitionId.ToString + "/forms=CMC_Student_Personal+Info"

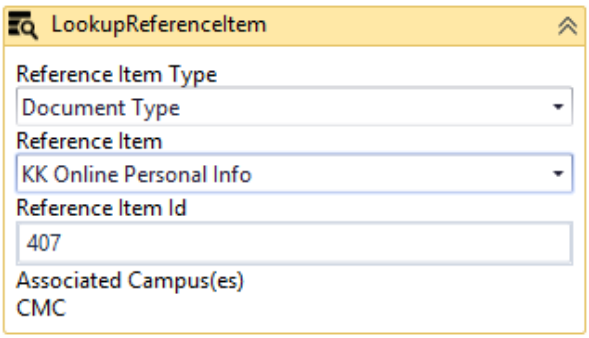
(where CMC_Student_Personal+Info is the name of the form that contains the DocuSign components).

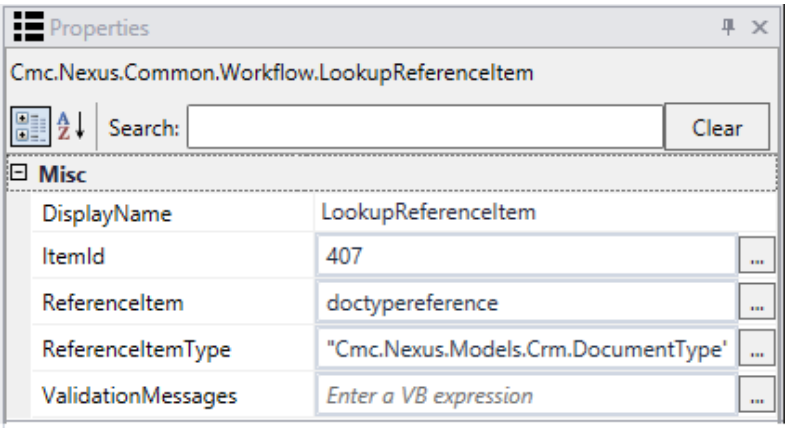
Spaces in form names must be replaced with a “+” when setting viewCreator URL.

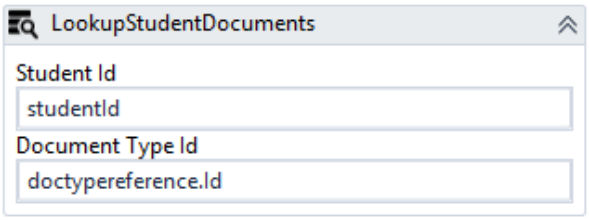
At start of the sequence, the workflow finds the student that was authenticated.



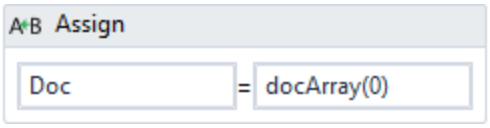
Next, the workflow finds the document tracking record associated with the student. The LookupReferenceItem activity returns the document type that will be used as input for the LookupStudent Documents activity.

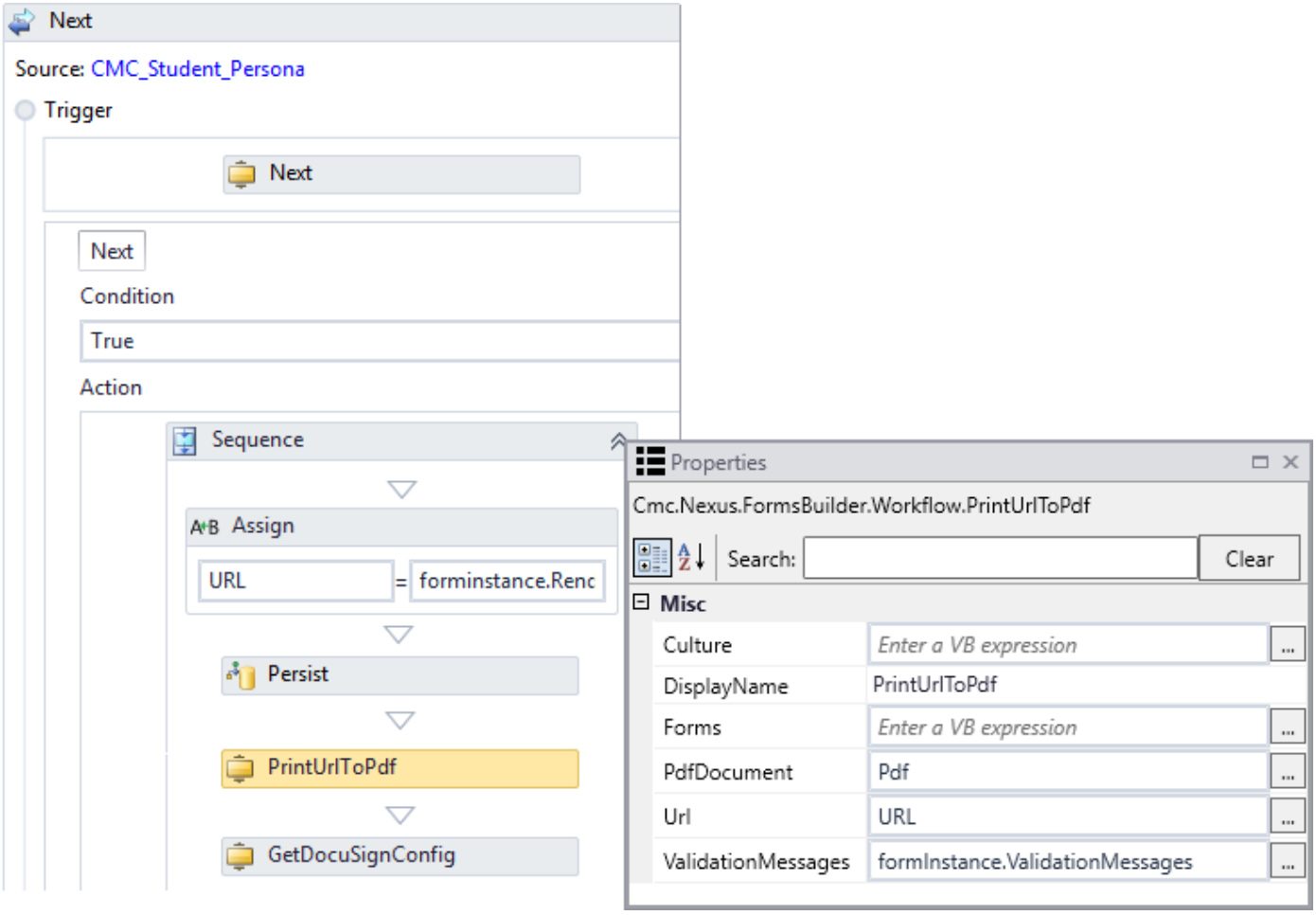




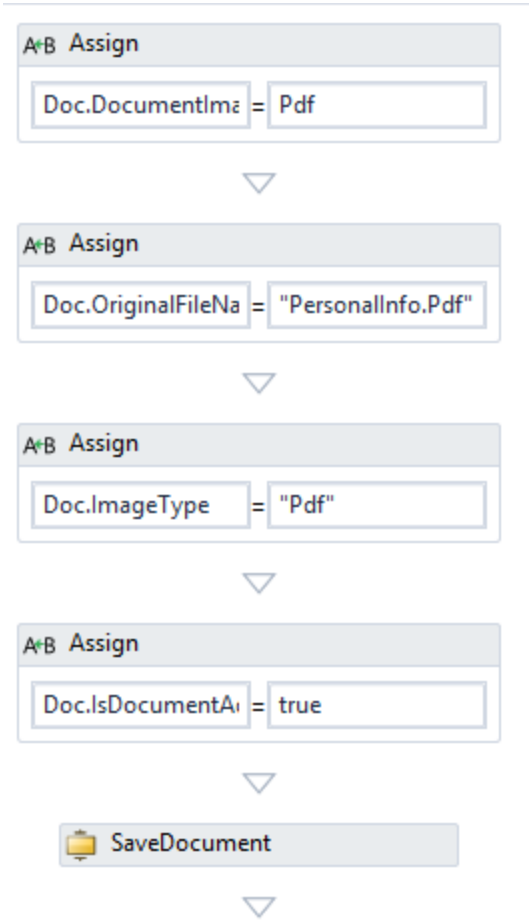


The LookupStudentDocuments activity returns an array of documents in variable docArray(0). The selected student should only have one document.





Assign activities set the document properties required for the SaveDocument activity.



Usage Notes

Sequences with File Upload Component

If the sequence you want to capture using the PrintUrlToPdf activity contains a File Upload component and you want to capture the image of the file selection in the File Upload component, be sure to place the PrintUrlToPdf activity **before** the GetAttachments activity, otherwise the file selection in the File Upload component will not be captured in the PDF file.

Multi Page Forms

On any longer forms that spread across multiple pages when converted to PDF, drop an HTML component just prior to the DocuSign control that appears close to the page break and specify the following in the Class Property: `forms_builder_page_break`. For more information, refer to Error Code "TAB_OUT_OF_BOUNDS" - DocuSign.

PrintUrlToPdf Times Out

The viewCreator wait is event-based. Forms may need to be re-saved, which will automatically update components to include a call to the event-based method "cmc-on-initialized". If PrintUrlToPdf times out, especially when many documents are merged into a single PDF file, simply re-save all forms in the sequence.

Incompatible Components

Do not use the CAPTCHA and Hyperlink components in forms that will be converted to PDF with the PrintUrlToPdf activity.

Auxiliary Service

The PrintUrlToPdf activity relies on an auxiliary service that performs the conversion to PDF for Azure deployments. This service is referenced in the `<appSettings>` section of the Renderer web.config file as highlighted below.

In Forms Builder 3.7, the names of the auxiliary service keys are changed:

From	`AuxiliaryServiceBaseUrl`	to	`PdfConversionServiceBaseUrl`
From	`UseRemotePDFConverterService`	to	`UseRemotePdfConversionService`

For on-premise deployments, the service is not required, and the `UseRemotePdfConversionService` value will be set to false (default).

```
<appSettings>
  <add key="ConfigureCampusNexusWcfProxy" value="true" />
  <add key="ConfigureCVueNexusWcfProxy" value="true" />
  <!-- Following will be populated when Crm is enabled for Forms Builder -->
  <add key="CmcNexusCrmWebUrl" value="http://<server>.campusmgmt.com:8090/" />
  <add key="PaymentProvider" value="PayPal" />
  <add key="PdfConversionServiceBaseUrl" value="" />
  <!-- Following should be set to true only in Azure environments where the Auxiliary service is installed and required. -->
  <add key="UseRemotePdfConversionService" value="false" />
</appSettings>
```

For Azure deployments, the `UseRemotePdfConversionService` value will be set to true, and value of the `PdfConversionServiceBaseUrl` will be:

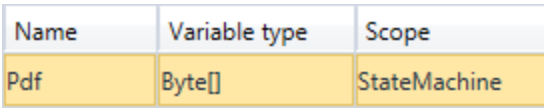
https://110001pdfrenderer.campusnexus.cloud/

This URL will be the same for all customers.

Properties

PrintUrlToPdf Properties

Property	Value	Required	Notes
Culture	`InArgument<String>`	No	If you want a PDF to be generated in a language other than the language selected by the student in the rendered form sequence, specify the Culture code, e.g., "fr" for French or "es" for Spanish. You can create a workflow with two PrintUrlToPdf activities where the first activity creates a PDF in the language selected by the student (with the student's signature if applicable) and the second PrintUrlToPdf activity creates the PDF in a different language for the staff member (without the student's signature). **Notes**: • Specify the Culture argument **only if** you want the PDF to be created in a language other than the original language used to render the sequence. Otherwise leave this field blank. • Do **not** specify the Url argument if you use the Culture argument. • If translations for the specified Culture argument are **not** available in the database, English (application default) will be used, and the student's culture selection will not be displayed.
DisplayName	`String`	No	Specify a name for the activity or accept the default.
Forms	`InArgument<String>`	No	Optionally, specify a comma separated list of form names to include in the PDF. You do not need to replace spaces with plus signs in this list. The Forms property value is ignored if the Url is **not** empty, in other words, if the Url property is specified, it takes precedence. If both the Url and Forms properties are left empty, the activity creates the Url automatically.

Property	Value	Required	Notes
PdfDocument	`OutArgument<Byte[]>`	Yes	Specify the PDF byte output array using a VB expression or variable. To identify the variable type, in the Variable type field of the Variables pane, select **Browse for Types...**. In the "Browse and Select a .NET Type" window, select **Array of [T]**, navigate to **mscorlib [4.0.0.0] > System**, select **Byte**, and select **OK**. 

Property	Value	Required	Notes

Url	`InArgument<String>`	No	Forms Builder 3.5 and later allows an empty URL input argument. The activity constructs the Url using:
			• formInstance.RendererBaseUrl
			• formInstance.WorkflowDefinitionId
			• formInstance.FormsTraversed
			
			This addresses the common scenario whereby all forms need to be included in the PDF by eliminating the need to type every form name with perfect casing for the Url. Backward compatibility is maintained by allowing users to create the Url and list of forms for the PDF output.
			
			The activity constructs the Url only if both the Forms and Url properties are left blank. If the Url property is specified, it takes precedence.
			
			Specify the Url using a VB expression or variable.
			
			Examples
			
			The following URL indicates a PDF that is created for a form named "CMC_Student_Personal Info":
			
			formInstance.RendererBaseUrl + "#/viewCreator/" + formInstance.WorkflowDefinitionId.ToString + "/forms=CMC_Student_Personal+Info"
			
			The following URL indicates a PDF that is created for multiple forms in a single document. Note the comma separated list after "forms=".
			
			formInstance.RendererBaseUrl + "#/viewCreator/" + formInstance.WorkflowDefinitionId.ToString + "/forms=CMC_Student_Personal+Info,CMC_Student_Additional+Info"
			
			If mistakes are made in the form name for the URL, the following message will be in the log, but no indication of failure will appear in the rendered sequence (no toaster popup), and the validationMessages argument will not be set. Additionally, the PDF will be created but will contain only the error message.
			
			`Form Form_Name does not exist in the database.`
			`Unable to get Form data from database`

Property	Value	Required	Notes
ValidationMessages	`InOutArgument <ValidationMessage Collection>`	No	Specify a variable that can be used to capture validation messages.

Page Breaks in PDF Files

The following styles determine the default PDF pagination scheme. They can be modified if some other pagination scheme is needed. Save your changes to a custom style sheet. For more information, refer to Custom Content and Custom Styles.

```
@media print {
#viewCreatorForm > div:not(:first-child) {
page-break-before: always;
}
}
```

/* The above @media print does not address page breaks within a long form. DocuSign may reject PDF files where DocuSign signatures cross page boundaries in the PDF files, usually with an error like "TAB_OUT_OF_ BOUNDS". Adding appropriate pages breaks is best accommodated by adding an HTML component in the form anywhere before a page break is required. Leave the HTML control empty but add the unique class name on the Class Property Setting that is used below: forms_builder_page_break. Then the following will force a page break in the PDF file before the control.
*/

```
@media print {
.forms_builder_page_break {
page-break-before: always;
}
}
```

The following css code would insert page breaks after a footer or before an h1 heading.

```
@media print {
footer {page-break-after: always;}
}
```

— OR —

```
@media print {
h1 {page-break-before: always;}
}
```

TranslateText

The TranslateText workflow activity translates text that originates in a workflow and needs to be displayed to the user. The activity is not used for translations of text that originates from the forms created in Forms Builder.

The TranslateText activity performs server-side translations for text strings in the Workflow Composer application using C# code and database queries. A precondition for this process is that the text to be translated (OriginalText) exists in the Forms Builder database and has been translated using the Gettext tools (.pot and .po files). Refer to Internationalization and Localization in Forms Builder.

Use cases:

- Translation of messages created using CreateValidationItem activities (refer to Custom Validations)

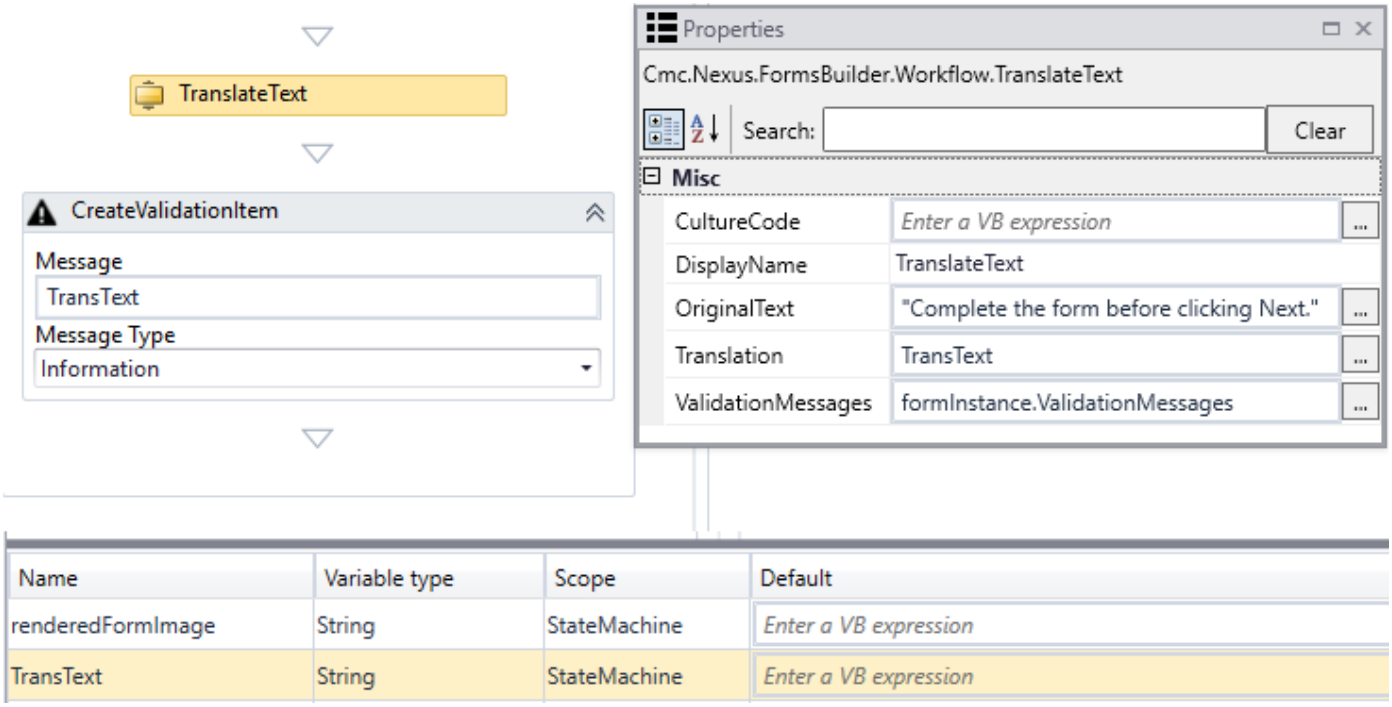
  Prerequisites: The custom validation messages are known, have been entered in a .pot file, have been translated, and the .po files have been imported into Forms Builder. The text string specified in the OriginalText field of the TranslateText activity must match the text of the custom validation message exactly.

- Translation of individual values returned from other workflow activities

  When workflow activities such as LookupReferenceItem are used to retrieve values from the database, these values can be entered in a .pot file, translated, and imported into Forms Builder via .po files. TranslateText activities can then be used to translate individual values as users are completing the form sequences.

  To provide translations for arrays such as text strings returned by OData queries in the selection lists, refer to Steps to Localize Sequences.

The image below shows the property settings for a TranslateText activity followed by a CreateValidationItem activity that uses the variable "TransText". The variable is defined as the Translation out argument in the TranslateText activity. The OriginalText argument "Complete the form before Clicking Next." will be translated based on the default Locale when an invalid condition causes the validation message to be displayed.



Properties

TranslateText Properties

Property	Value	Required	Notes
CultureCode	`InArgument<String>`	No	The CultureCode is optional. If the CultureCode is not specified, by default the activity will use the formInstance.Locale value. The Locale component, which is typically placed on first form of the sequence, sets the formInstance.Locale value. You can specify a CultureCode value in the TranslateText activity to override the formInstance.Locale value on any form or to assign the CultureCode if the Locale component is not used.
DisplayName	`String`	No	Specify a name for the activity or accept the default.
OriginalText	`InArgument<String>`	Yes	Specify the OriginalText value. The TranslateText activity takes the OriginalText value and returns a Translation value that is retrieved from the Forms Builder database. The OriginalText value must match a string in the database table exactly.

Property	Value	Required	Notes
Translation	`OutArgument<String>`	No	Specify the Translation value using a VB expression or variable. If the Translation is not found, the OriginalText value will be returned in the out argument. You can use the log file in debug mode and search for "No Translation found" to determine which items were not translated.
ValidationMessages	`InOutArgument <ValidationMessageCollection`	No	Specify a variable that can be used to capture validation messages.

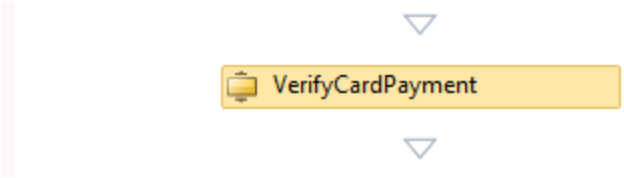
VerifyCardPayment

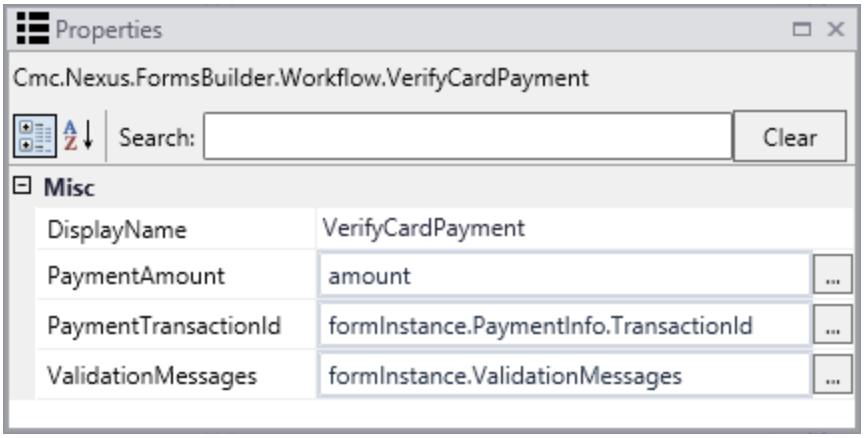
The VerifyCardPayment activity is used in workflows for sequences that contain a form with the Credit Card Payment component. The activity validates that the information received when the payment was processed matches the payment details for the provided transaction identifier that was returned when the payment was processed. A payment can be processed using the ACI, IATS, or PayPal payment gateways.

If the VerifyCardPayment activity is missing in the workflow for the sequence that contains the Credit Card Payment component, the transaction identifier will be null, and the form with the Credit Card Payment component will show a validation message or return an "event not verified" error.

The placement of the VerifyCardPayment activity must be in the **Next** transition from the form that contains the Credit Card Payment component.

> ⚠ The transition out of the form that has the Credit Card Payment component has to keep its the default name, i.e., “Next”. Do not change this default for this particular form. When receiving the post back from the payment site, Forms Builder automatically transitions forward and looks for “Next” on the form.





> ⚠ The VerifyCardPayment activity does not check for form validation errors. If errors occur, the activity will return to same form, but the state of the form sequence and payment are not guaranteed. You need to ensure that there are no input validation errors prior to the user selecting the "Make Payment" link.

Capture validation errors using an **If** activity following the VerifyCardPayment activity. Specify the Condition for the If statement as: **formInstance.ValidationMessages.HasErrors**.

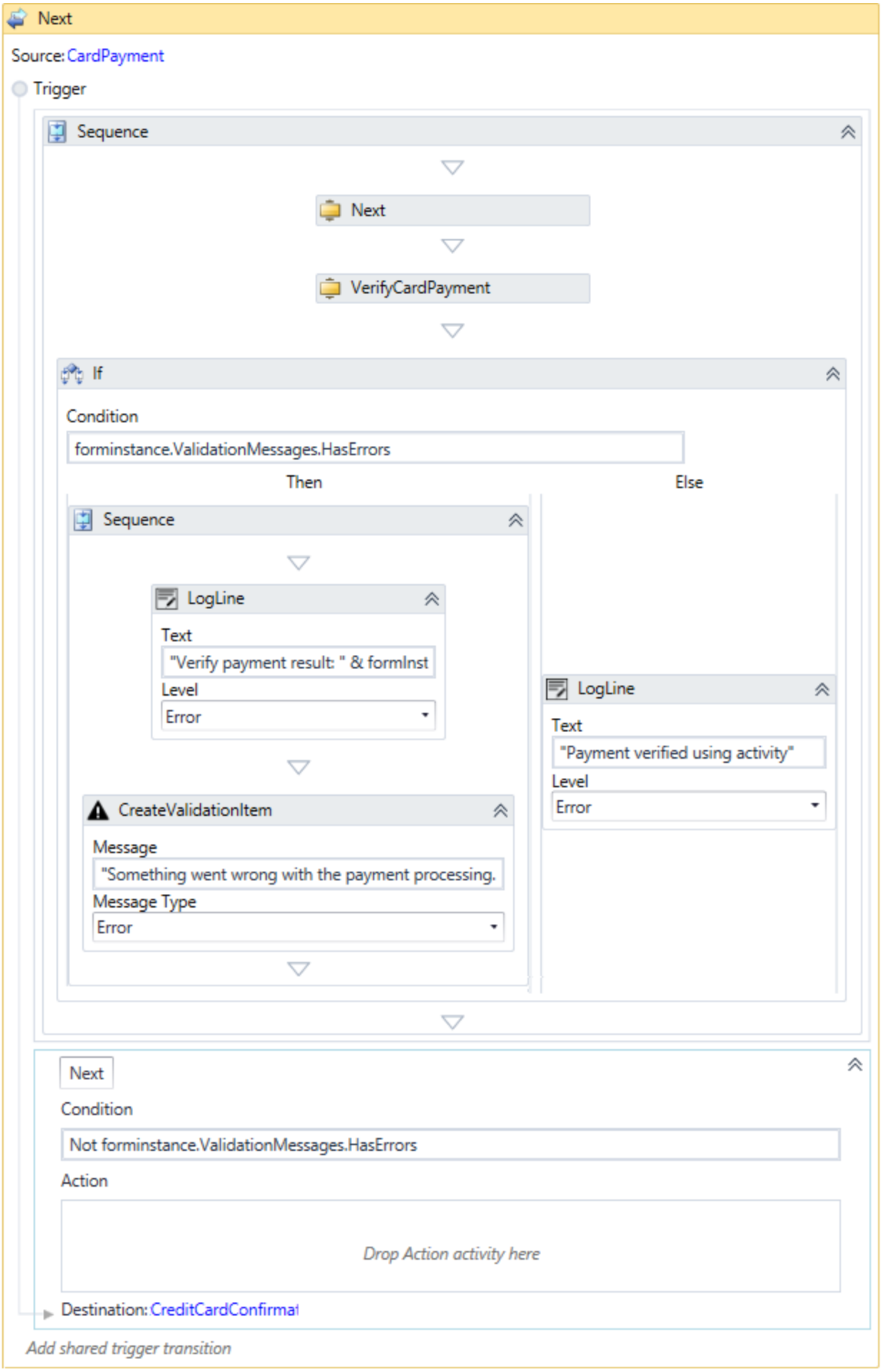
- In the Then branch, add a LogLine (e.g., **Text = "Verify payment result: " & formInstance.ValidationMessages(0).Message**) and a CreateValidationItem activity that displays a message to

the user.

- In the Else branch, place a LogLine activity that records the success of the VerifyCardPayment activity in the log, e.g., **"Payment verified using activity."**

We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

The Condition area of the Next transition contains the expression **Not formInstance.ValidationMessages.HasErrors**. The Action area leads to the next form in the sequence, in this case, a payment confirmation form.


Next
Source: CardPayment
Trigger
Sequence
Next
VerifyCardPayment
If
Condition
forminstance.ValidationMessages.HasErrors
Then
Else
Sequence
LogLine
Text
"Verify payment result: " & formInst
Level
Error
CreateValidationItem
Message
"Something went wrong with the payment processing.
Message Type
Error
LogLine
Text
"Payment verified using activity"
Level
Error
Next
Condition
Not forminstance.ValidationMessages.HasErrors
Action
Drop Action activity here
Destination: CreditCardConfirma
Add shared trigger transition

Properties

VerifyCardPayment Properties

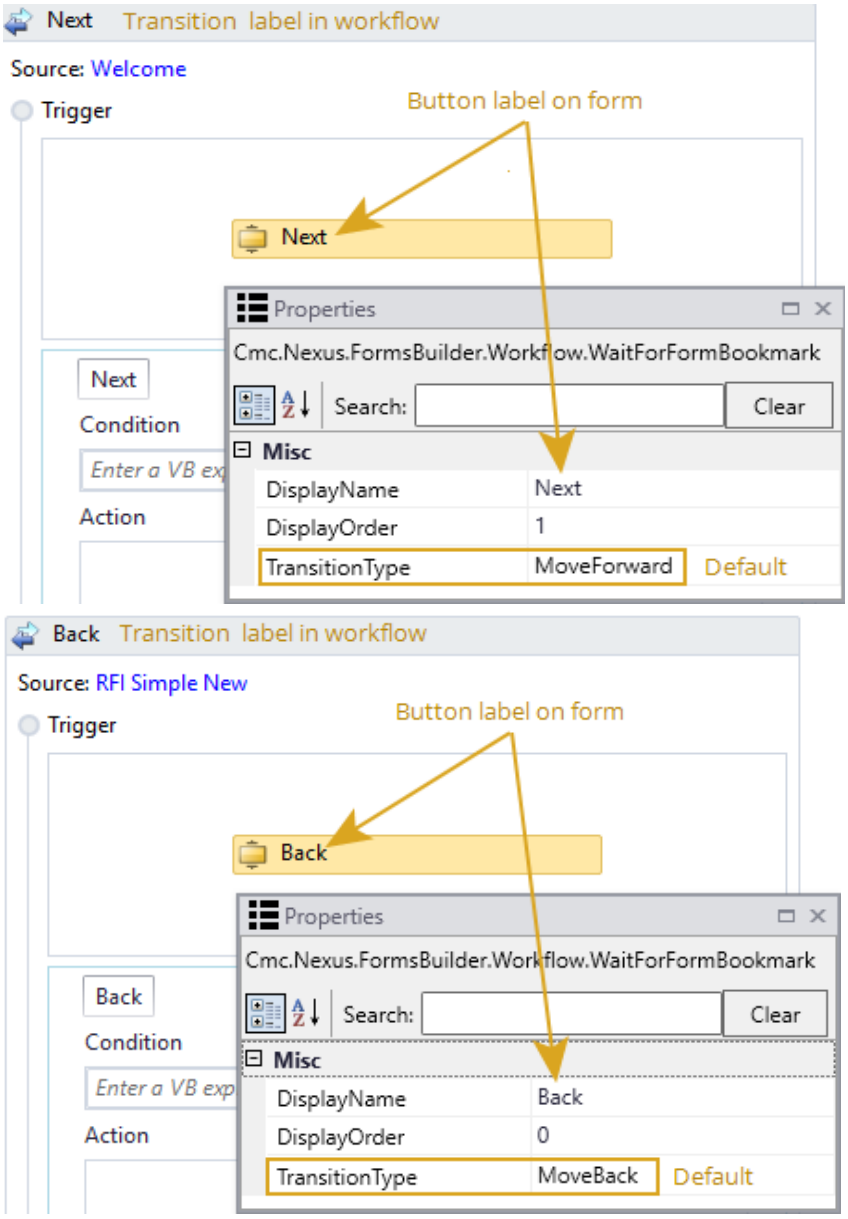
Name	Value	Required	Notes
DisplayName	`String`	No	Specify a name for the activity or accept the default.
PaymentAmount	`InArgument<Decimal>`	Yes	Specify the amount to be paid using a credit card, for example, `45.00d`. If you specify a binding to an entity in the database, for example, `depositEntity.Amount`, make sure that the workflow for the form sequence contains a matching CreateEntity<> activity, for example, `CreateEntity<DepositEntity>`.
PaymentTransactionId	`InArgument<String>`	No	Specify the transaction identifier associated with the payment form instance: `formInstance.PaymentInfo.TransactionId`
ValidationMessages	`InOutArgument<ValidationMessageCollection>`	No	Specify a variable that can be used to capture validation messages. You can either set this value to `formInstance.ValidationMessages` to show it to the end user or assign it to a local variable and customize `formInstance.ValidationMessages`.

WaitForFormBookmark

The WaitForFormBookmark activity creates a bookmark so that the workflow can resume after user input is received. This activity is also responsible for passing data (as arguments) back and forth between forms and workflows.

WaitForFormBookmark is used in all form/state transitions. By default, when you create a form sequence, the Trigger sections of all transitions will already contain WaitForFormBookmark activities with default Display Names for the Next and Back buttons, and the transitions in the workflow will be labeled "Next" and "Back" accordingly.

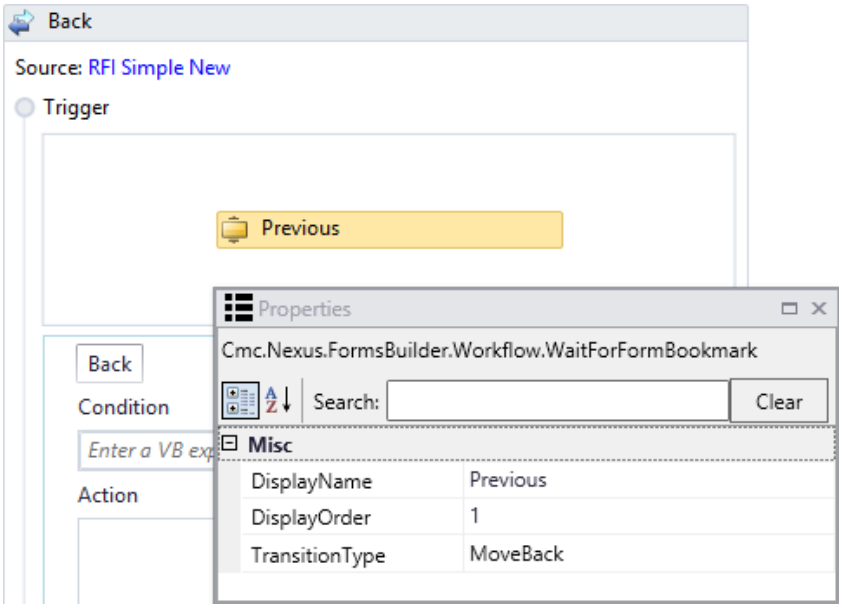
The default Transition Type for the "Next" button is "MoveForward", and the default Transition Type for the Back button is "MoveBack".

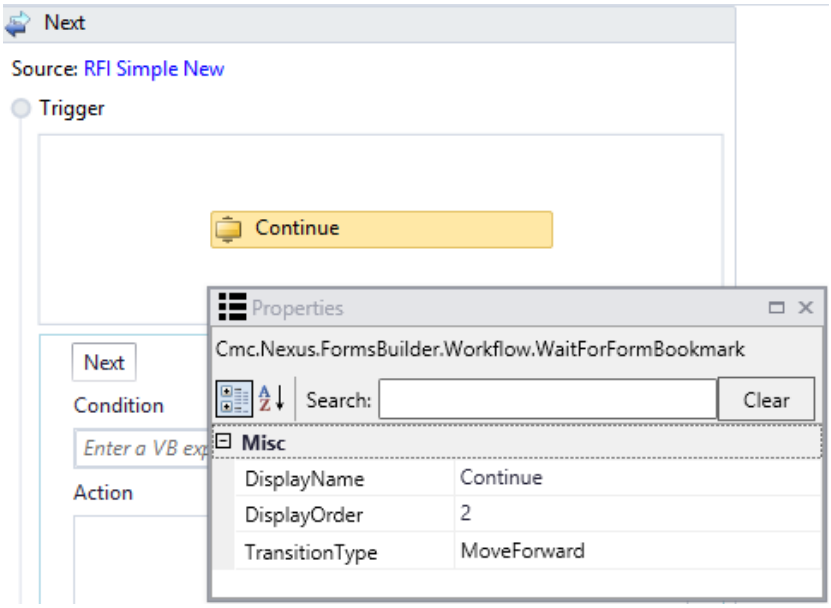


You must specify the Transition Type in the WaitForFormBookmark activities if you change the default button labels. For example, to change the label of the "Back" button to "Previous" and the "Next" button to "Continue", you would specify the WaitForFormBookmark properties as shown below.

Display Order is not required if you are changing the Transition Type. If Display Order is left at the default of 0 (or all values are the same), the sorting of buttons on the form is alphabetical. You can use this property to give the buttons a sequential (left to right) order.







WaitForFormBookmark clears validation messages. Therefore, it is necessary to add a CreateValidationItem activity after the bookmark in the transition. For more information, refer to Custom Validations.

If you add a new form (i.e., State activity) with new transitions to a workflow, ensure that a WaitForFormBookmark activity is added to the Trigger area of every transition. The default name for a new transition will be "T1". You can change the name of the transition as needed. The Display Name of the WaitForFormBookmark activity will be the label of the button on the form.

Properties

WaitForFormBookmark Properties

Property	Value	Required	Notes
Display Name	`String`	Yes	Specify a name for the activity or accept the default. The Display Name should match the button name on the form. When the default Display Names "Back" and Next" are used, the Display Order and Transition Type properties are not required. If you customize the button labels, you must specify the Display Order and Transition Type properties for each WaitForFormBookmark.
Display Order	`Int32`	No	When a form uses the default button labels "Back" and Next", the Display Order value is 0 (default). If you customize the button labels, specify the order of the buttons from left to right using increasing integer values from left to right for each WaitForFormBookmark.
Transition Type	`TransitionType`	No	When a form uses the default button labels "Back" and Next", the Transition Type value is "MoveForward" for the "Next" button and "MoveBack" for the "Back" button. If you customize the button labels, specify the direction of the transition by selecting the Transition Type values "MoveBack" or "MoveForward" as appropriate for each WaitForFormBookmark.

When the user selects the **Next** button, if the contents of the next rendered form or the activities during a transition have not been completed, the spinner that signifies that the page is currently being processed is displayed in the center of the form so that it is clearly visible on the screen and the form does not appear to be frozen.

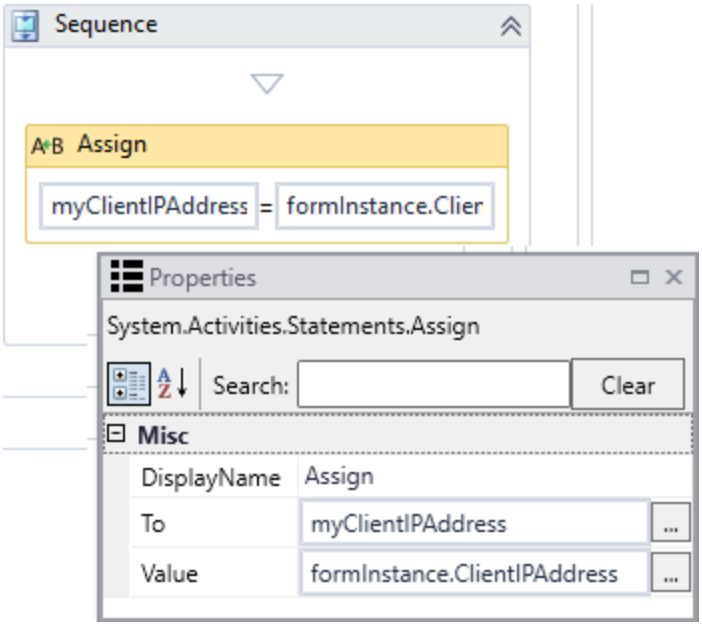
Collect IP Address

With Forms Builder 3.7 and later you can create a workflow that collects the public IP address of a user accessing a sequence. The form and form sequence do not need to be modified.

Institutions collecting IP addresses need to ensure that the privacy policy for their web site clearly indicates that the IP address is a field that may be captured.

To capture a user's IP address:

1. Add an **Assign** activity to the first form (state) in the workflow and assign the **formInstance.ClientIPAddress** value to an argument.



2. Create an **Argument** of type **String** for the ClientIPAddress value.

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event	In/Out	ConstructedEvent	*Default value not supported*
studentEntity	In/Out	StudentEntity	*Default value not supported*
myClientIPAddress	In/Out	String	*Default value not supported*

3. Depending on your use case, create other workflow operations using the ClientIPAddress argument.
4. To check an IP address on the rendered form in a test environment, add the JSON Debug Info component

to the form.



If the form is accessed from a computer...

- in a corporate network or cloud, the IP address of the gateway will be captured, i.e., the IP address will be the same for all users within that network.
- outside of a corporate network or cloud, the IP address of the individual user's computer will be captured.



Do not trust the value of the ClientIPAddress for critical business functions since IP addresses can easily be spoofed.

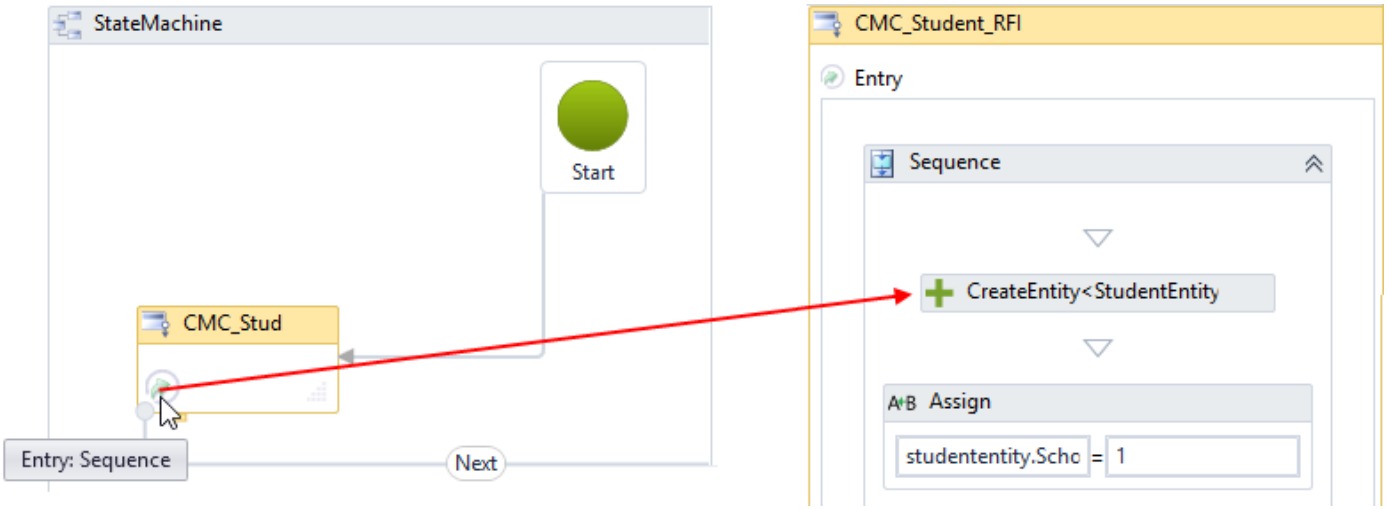
Before you use this feature in a production environment, ensure that you do not violate any privacy policies or other legal requirements regarding the collection of user data.

Create, Get, and Save Entity Activities

When a form sequence is created and saved, Sequence Designer automatically creates a workflow definition containing In/Out arguments for all entities that are referenced in the forms within the sequence. The arguments are used to pass data between Forms Builder and Workflow Composer. The argument names match the entity types. The parameters associated with the entities are not initialized by default. The entities associated with the arguments are not automatically created in the workflow. It is the responsibility of the user building the form to add the necessary CreateEntity<>, GetEntity<>, and SaveEntity<> activities in appropriate locations of the workflow definition. Appropriate locations are Transitions (e.g., Next, Submit, Back) and States (i.e., forms) in the StateMachine workflow.

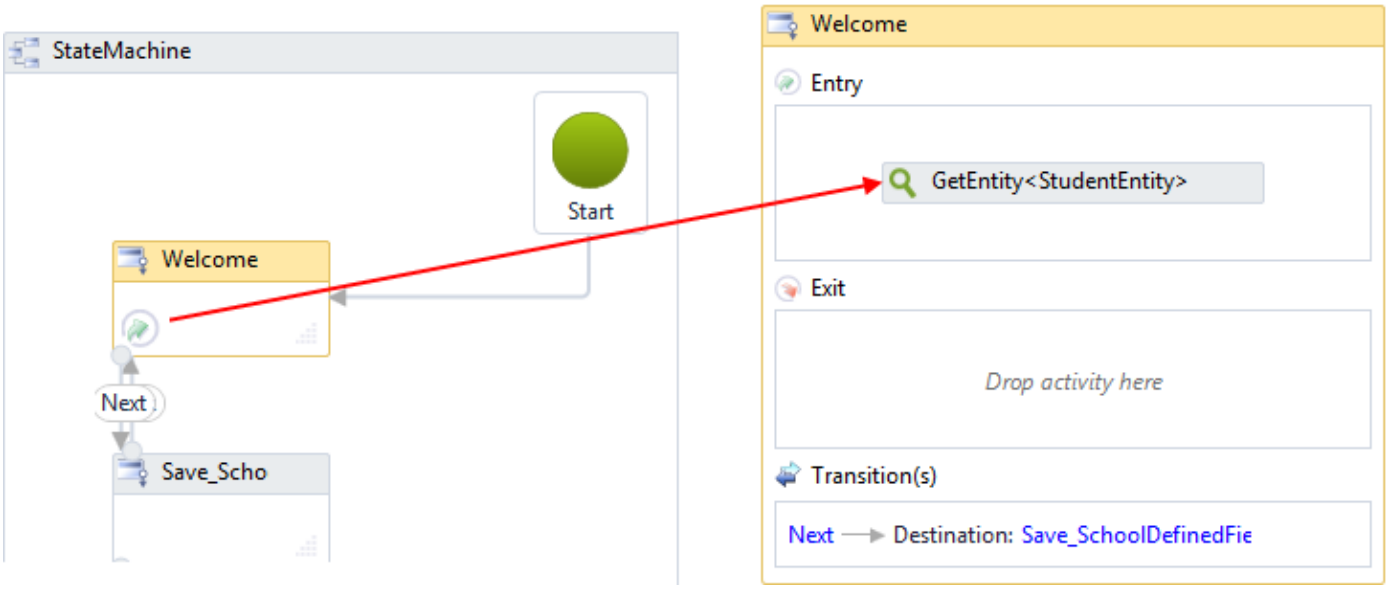
CreateEntity<>

The CreateEntity<> activity is usually placed in the Entry Sequence of the first State (form) in a StateMachine workflow. Often, one or more Assign activities follow the CreateEntity<> activity.



GetEntity<>

The GetEntity<> activity can be used to retrieve data from the database. The activity could be placed on the Welcome form so that previously entered data prefills the form fields when the form is displayed.



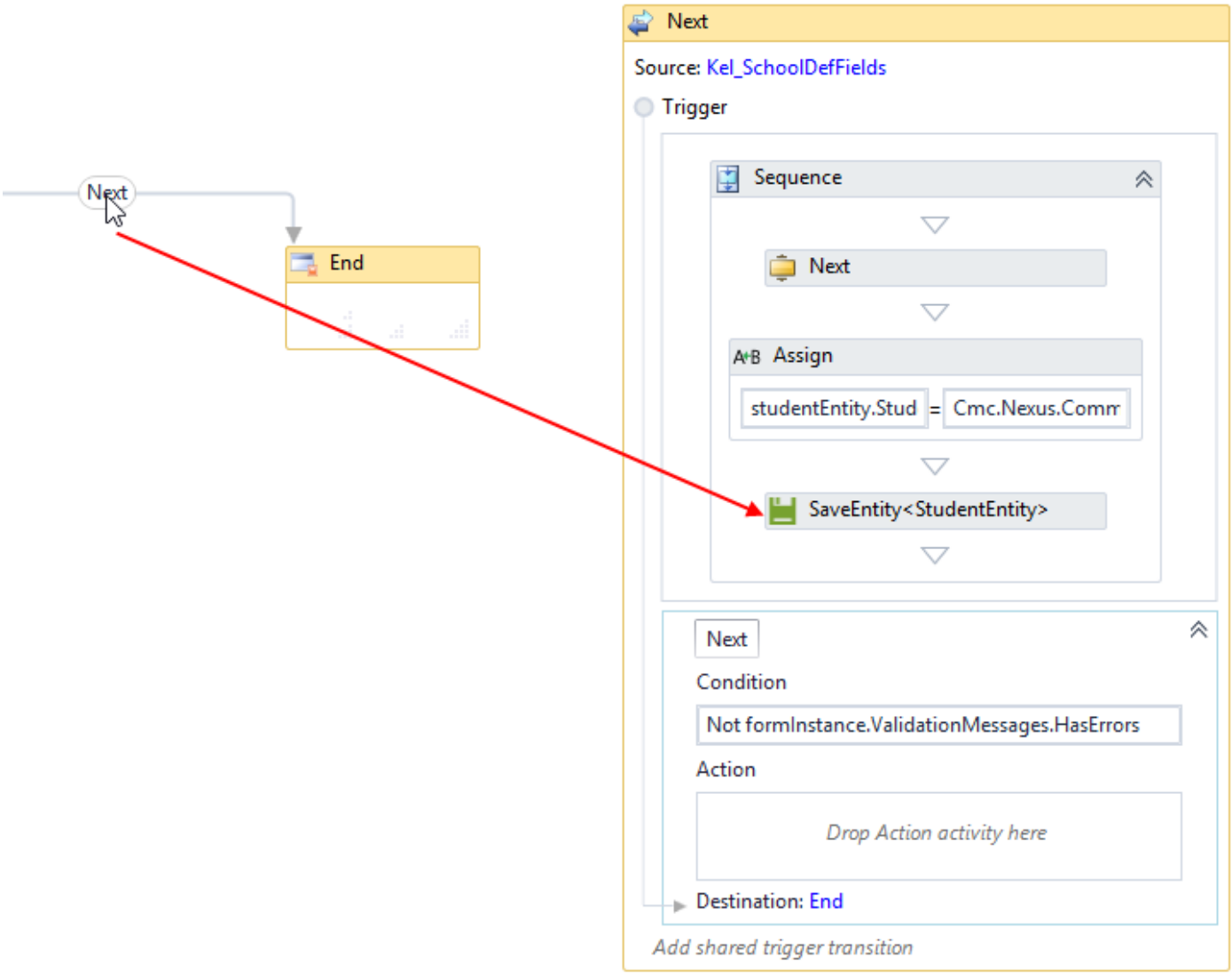
The following table illustrates the behavior of Forms Builder and Workflow Composer depending on the user scenario. The StudentEntity and ContactEntity are used as examples in the table; however, the concept applies to other entities as well.

User scenario	Which workflow activity is needed in the first form?	Is the FormInstance.UserInfo property populated in Workflow Composer?	Is the Login field populated on the form?	Notes
Anthology Student				
Anonymous user	CreateEntity – StudentEntity	– NA –	– NA –	
New user creating an account	CreateEntity – StudentEntity	Yes	Yes	Add an Assign activity to assign **studentEntity.FirstName = formInstance.UserInfo.FirstName** The FormInstance.UserInfo comes from the login account information. The same assignment can be done for all other account fields in FormInstance.UserInfo to prepopulate those fields for StudentEntity in form.

User scenario	Which workflow activity is needed in the first form?	Is the FormInstance.UserInfo property populated in Workflow Composer?	Is the Login field populated on the form?	Notes
Existing user logged in	GetEntity – StudentEntity	Yes	Yes	
CampusNexus CRM				
Anonymous user	CreateEntity – ContactEntity	– NA –	– NA –	
New user registering as a contact	GetEntity – ContactEntity	No (get data from the ContactEntity)	No by default (get data from the ContactEntity)	
Existing user logged in	GetEntity – ContactEntity	No (get data from the ContactEntity)	No by default (get data from the ContactEntity)	

SaveEntity<>

The SaveEntity<> activity can be placed in the Trigger or Action area of the Next transition that leads to the End State. A SaveEntity<> activity must be paired with each CreateEntity<> or GetEntity<> activity to ensure that the data collected in the form sequence is persisted to the database.



Refer to Workflow Help for more details on the CreateEntity<>, GetEntity<>, and SaveEntity<> activities.

Best Practice to Prevent DbUpdateConcurrency Exceptions

A DbUpdateConcurrency error occurs when an attempt is made to update an instance of an entity via a Save activity, but that instance has been modified by another user in the time from when the instance was initially retrieved in the workflow to the point in time when the Save activity executes.

Example of a DbUpdateConcurrency exception in a Renderer log file:

2018-02-27 13:30:16.7645 54 Error Cmc.Nexus.Crm.Workflow.SaveDocument System.ServiceModel.FaultException`1[System.ServiceModel.ExceptionDetail]: Store update, insert, or delete statement affected an unexpected number of rows (0). Entities may have been modified or deleted since entities were loaded. See http://go.microsoft.com/fwlink/?LinkId=472540 for information on understanding and handling optimistic concurrency exceptions. (Fault Detail is equal to An ExceptionDetail, likely created by IncludeExceptionDetailInFaults=true, whose value is: System.Data.Entity.Infrastructure.DbUpdateConcurrencyException: Store update, insert, or delete statement affected an unexpected number of rows (0). Entities may have been modified or deleted since entities were loaded. See http://go.microsoft.com/fwlink/?LinkId=472540 for information on understanding and handling optimistic concurrency exceptions. ----> System.Data.Entity.Core.OptimisticConcurrencyException: Store update, insert, or delete statement affected an unexpected number of rows (0). Entities may have been modified or deleted since entities were loaded. See http://go.microsoft.com/fwlink/?LinkId=472540 for information on understanding and handling optimistic concurrency exceptions. at System.Data.Entity.Core.Mapping.Update.Internal.UpdateTranslator.ValidateRowsAffected(Int64 rowsAffected, UpdateCommand source) at System.Data.Entity.Core.Mapping.Update.Internal.UpdateTranslator.Update() at System.D...).

The best practice to avoid this error is to add a TransactionScope activity to the workflow. Use the defaults of IsolationLevel = Serializable, and a timeout of 1 minute.

Within that TransactionScope, add a **GetEntity** activity to retrieve the instance of the entity **prior to** the execution of the **SaveEntity** activity. Any property values that need to be updated prior to saving can be done so via Assign statements right after the Get activity and right before the Save activity.

A transaction locks the database to give the workflow a chance to read and update with no other process simultaneously doing the same. Read about the other less aggressive isolation levels as they may be adequate for the purpose based on the type of updates being done and produce less overhead. Google “TransactionScope IsolationLevel Activities”. A “RepeatableRead” may be sufficient.

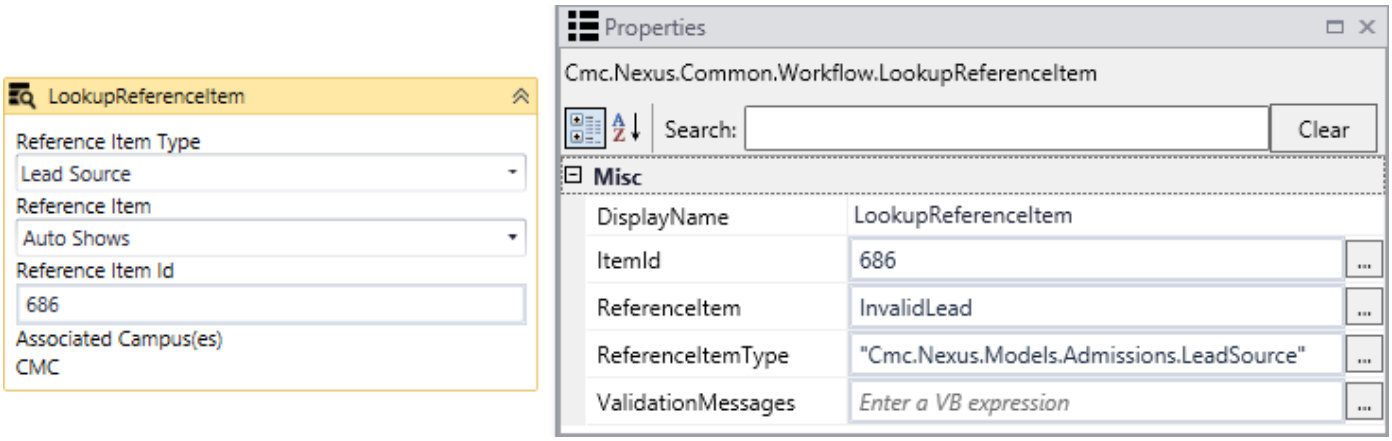
This pattern will eliminate any chance that another user will update this record in between the execution of the Get and Save activities within the workflow.

Custom Validations

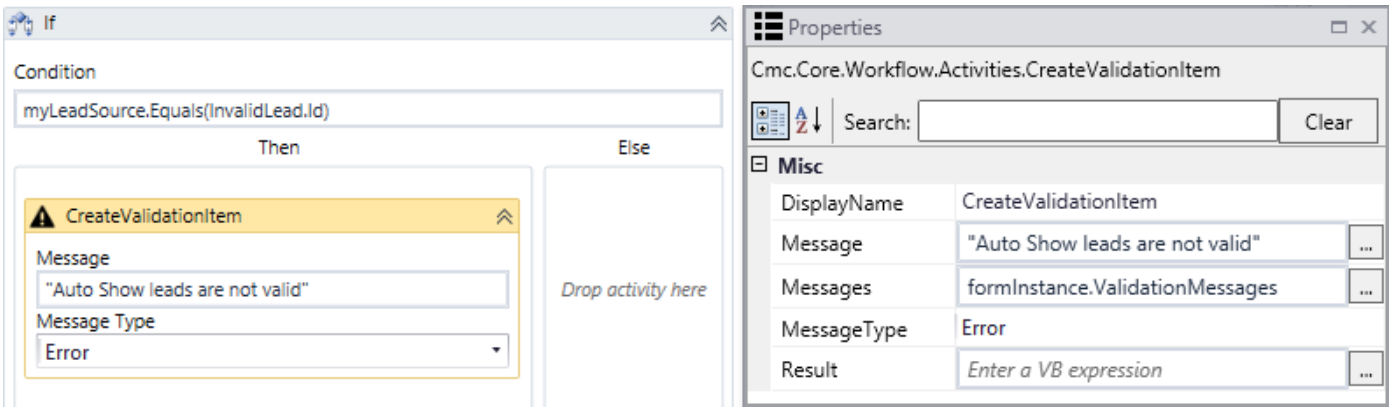
The workflow for a form sequence provides the ability to perform custom validations on the form input. Custom validations using the **CreateValidationItem** workflow activity can check for valid data input above and beyond the built-in validations for form fields within Forms Builder.

Single Validation

The workflow for a Request for Information (RFI) sequence includes a LookupReferenceItem activity that checks for a Reference Item Type of "Lead Source" with a Reference Item value of "Auto Shows". The out argument of the activity is a variable named "InvalidLead".



The "InvalidLead" variable is used in an If activity with a condition of "myLeadSource.Equals(invalidLead.Id)". If this condition is true (i.e., the lead is associated with a Reference Item value of "Auto Shows"), a custom validation message is displayed. This message is triggered by a CreateValidationItem activity, where the Messages variable is set to "formInstance.ValidationMessages".

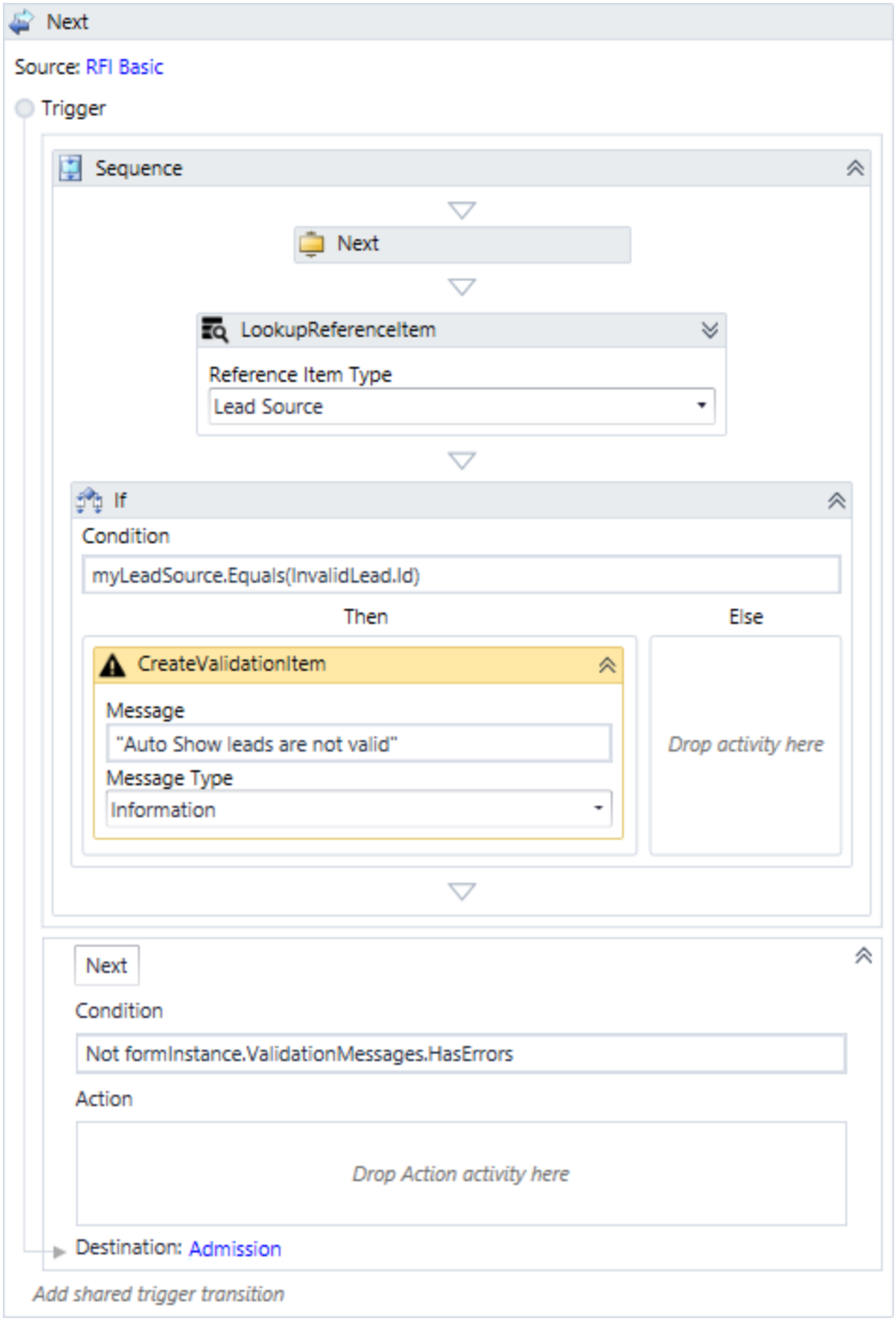


The validation item can be designed to use a message type of Error, Information, or Warning. On the rendered form, the message types are displayed as follows:

Validation Message Types

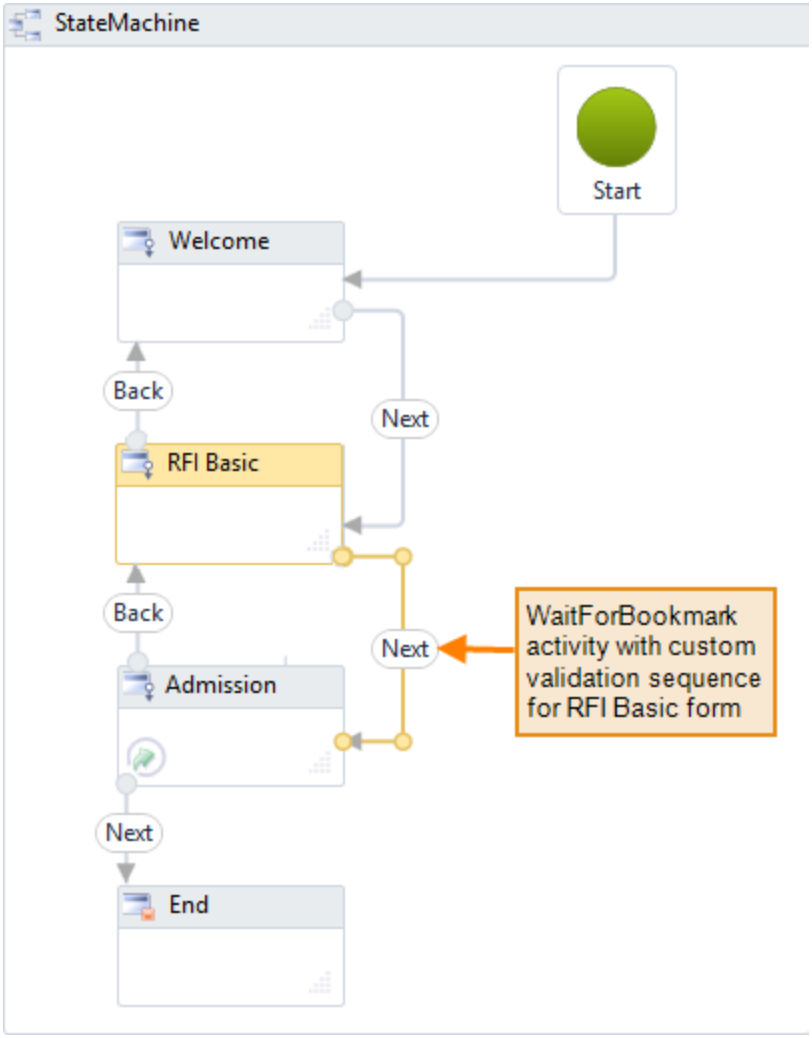
Message Type	Message Display in Renderer	Next Transition allowed?
Error	Red toaster popup	No, user must correct the form input
Warning	Orange toaster popup	Yes
Information	Blue toaster popup	Yes

For form input that meets the validation criteria (no errors) specify "Not formInstance.ValidationMessage.HasErrors" in the Next Condition field of the form transition. This allows users to continue to the subsequent form.



Placement of the Custom Validation

Place the custom validation sequence in the Trigger of the WaitForBookmark activity (labeled "Next") **from** the form on which you want to do the validation (Source: "RFI Basic" form in our example). The custom validation message will be displayed before the transition to the subsequent form (Destination: "Admission" form in our example).



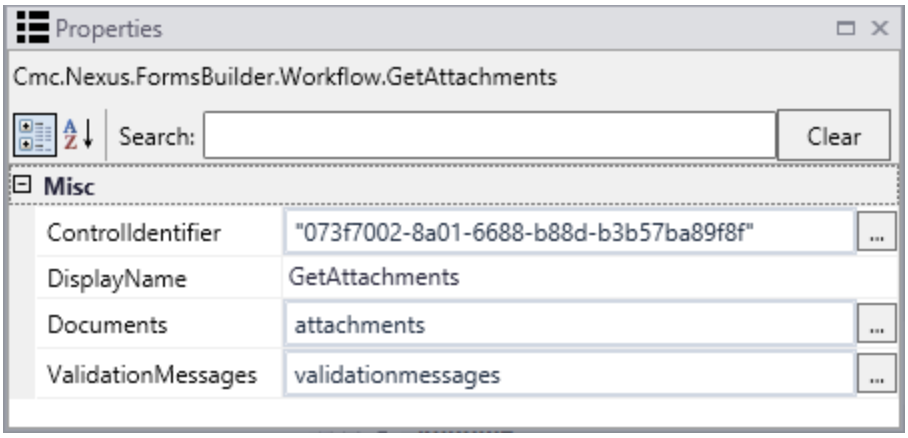
The WaitForBookmark activity clears any previous validation items. Therefore, the custom validation should be done **after** the transition from the form that is being validated.

Multiple Validations

You can also use a workflow to create multiple validations for a particular form in a sequence.

If any Forms Builder or Anthology Student activity is placed between two CreateValidationItem activities, a local variable must be created for the ValidationMessages in-argument of that activity instead of the usual formInstance.ValidationMessages argument. This is necessary because the activity will overwrite any already existing ValidationMessages in formInstance.ValidationMessages if that is supplied as an argument.

In our example, a GetAttachments activity is placed between two validations with CreateValidationItem activities.

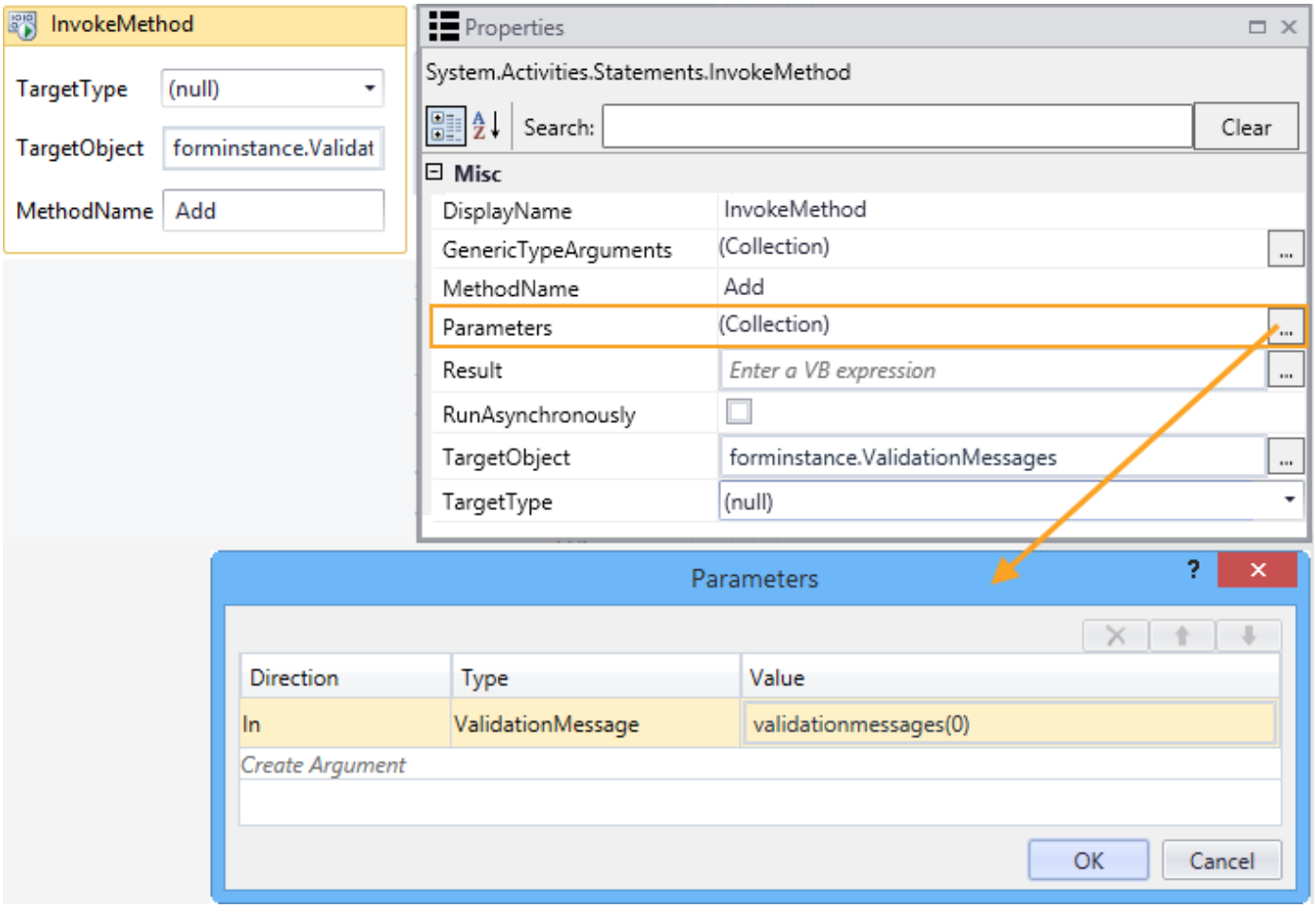


A local variable of type **ValidationMessageCollection** is created and supplied to the GetAttachments activity.

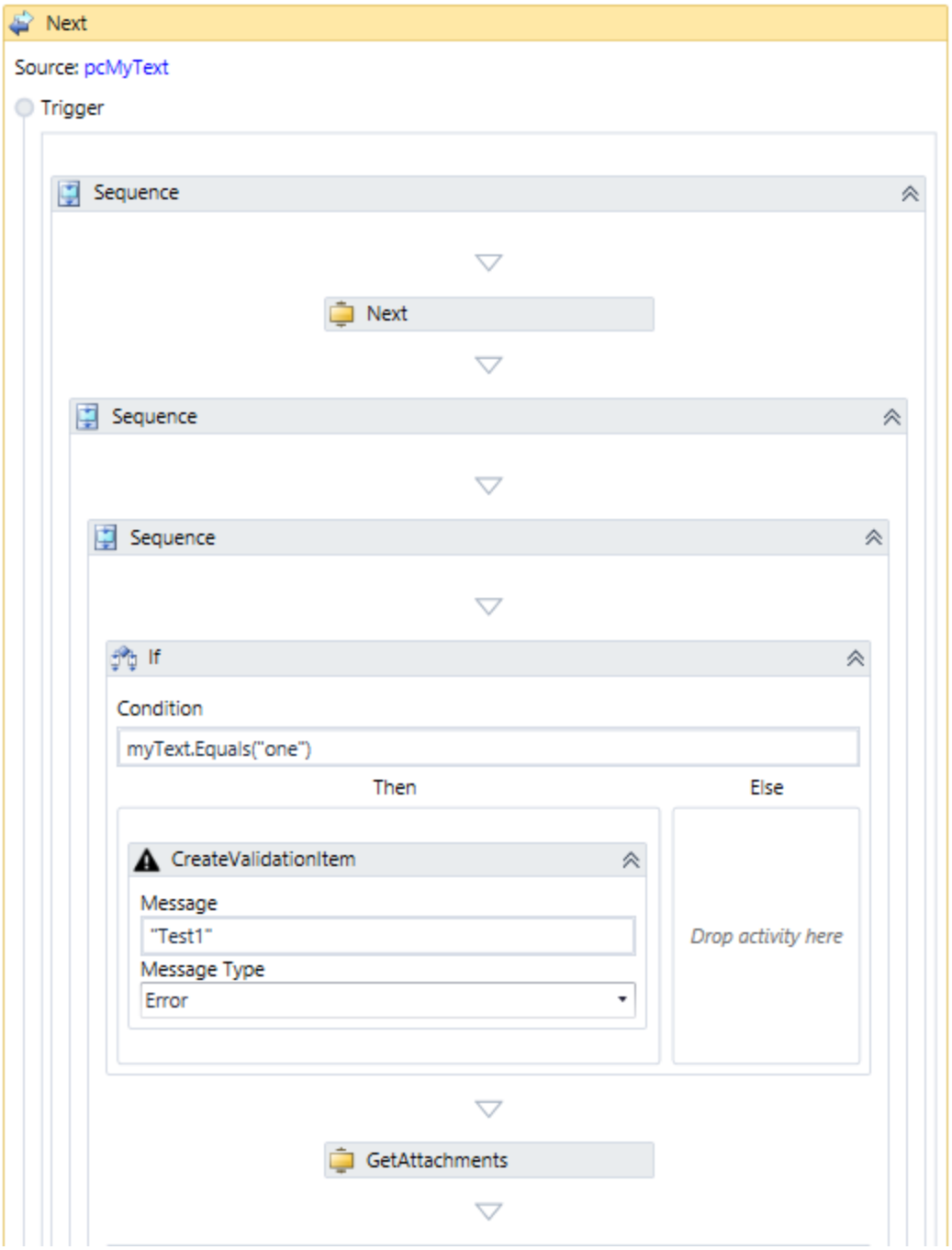
Name	Variable type	Scope	Default
validationmessages	ValidationMessageCollection	Sequence	*Enter a VB expression*
renderedFormImage	String	StateMachine	*Enter a VB expression*
attachments	UploadStorageEntity[]	Sequence	*Enter a VB expression*

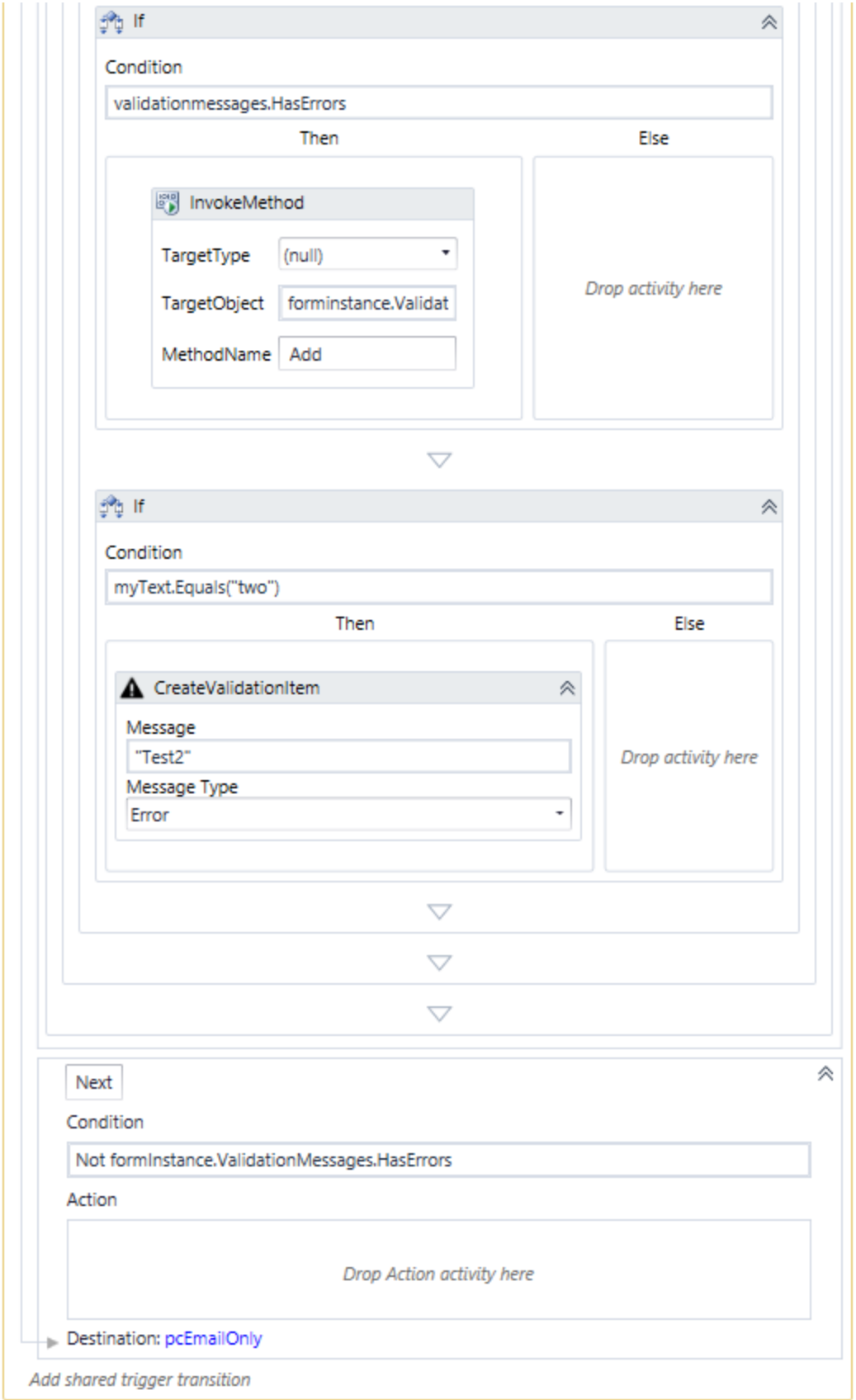
If the activity returns errors, the **InvokeMethod** activity adds the validation message to formInstance.ValidationMessages so it can be displayed in Renderer.

Supply a parameter for the Add method by selecting ... the Parameters attribute in the Properties window. Specify **validationmessage(0)** in the Value field for the Parameters attribute.



The following images show the workflow section that contains the validation steps. It is placed in the Next transition after the form that contains the fields to be validated.


Next
Source: pcMyText
Trigger
Sequence
Next
Sequence
Sequence
If
Condition
myText.Equals("one")
Then
Else
CreateValidationItem
Message
"Test1"
Message Type
Error
Drop activity here
GetAttachments

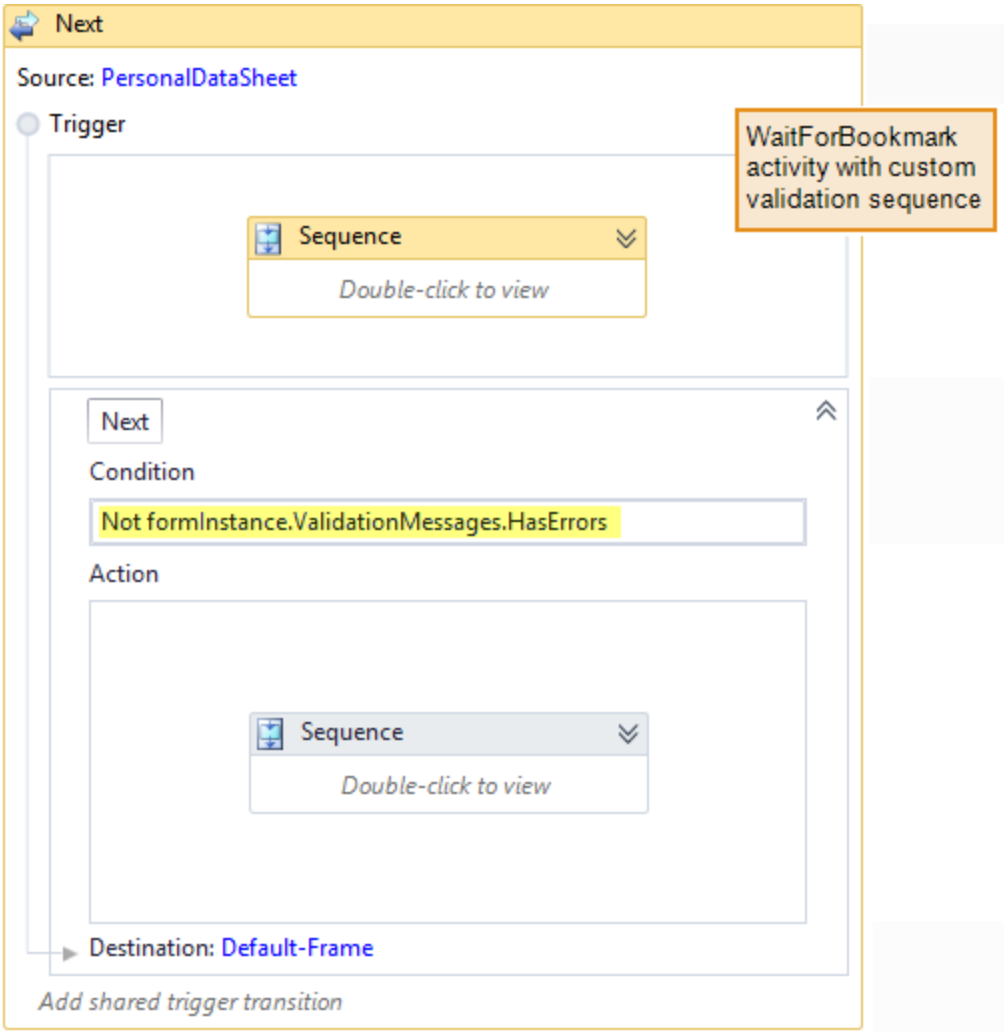

If
Condition
validationmessages.HasErrors
Then
Else
InvokeMethod
TargetType
(null)
TargetObject
forminstance.Validat
MethodName
Add
Drop activity here
If
Condition
myText.Equals("two")
Then
Else
CreateValidationItem
Message
"Test2"
Message Type
Error
Drop activity here
Next
Condition
Not formInstance.ValidationMessages.HasErrors
Action
Drop Action activity here
Destination: pcEmailOnly
Add shared trigger transition

Multiple Validations Items When Processing a Grid

You can use custom validations to validate multiple items when processing a grid. The custom validations can loop through an array of items and display all validation errors when the user attempts to save the data.

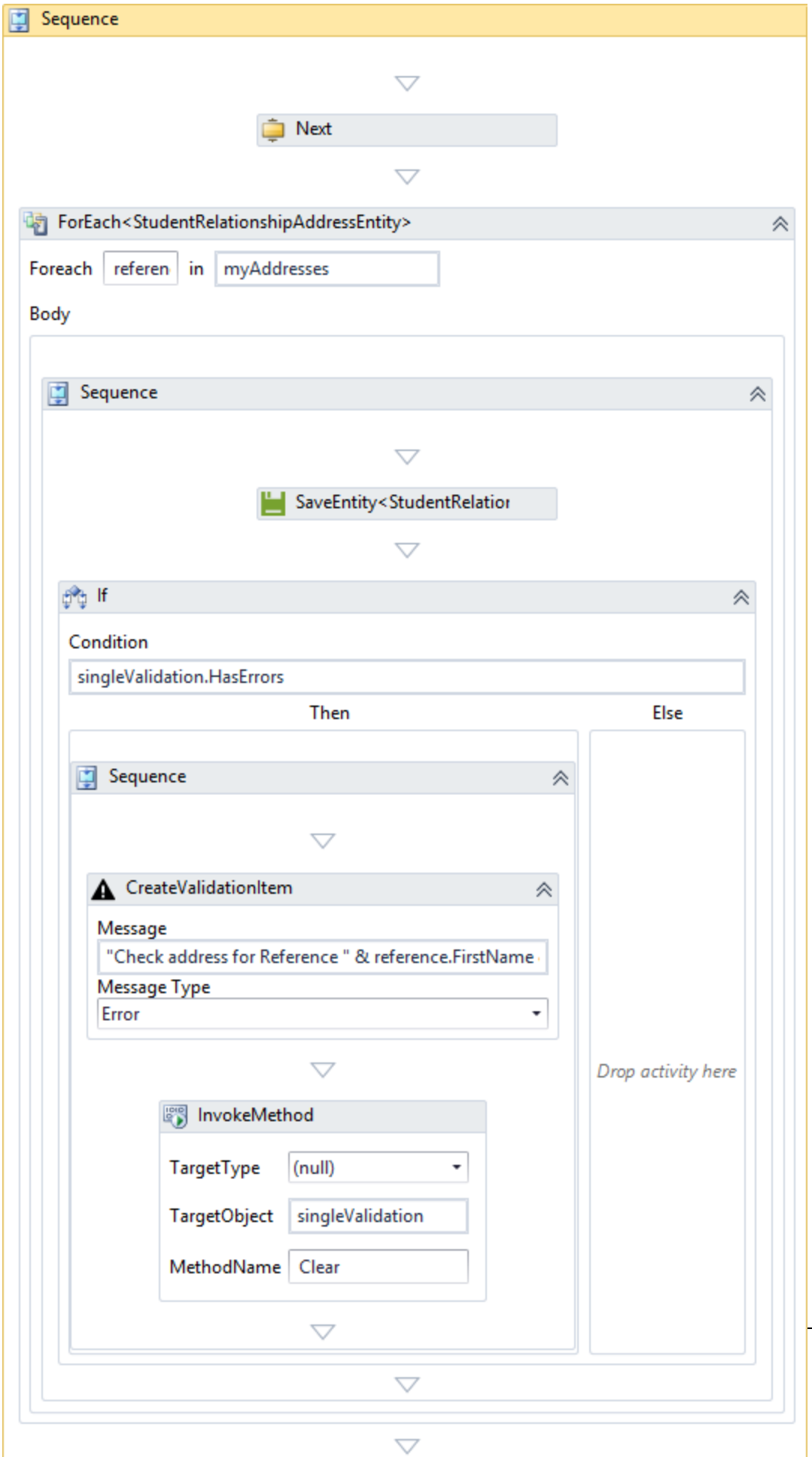
This example is based on a form in which the user enters an array of reference addresses. The workflow checks for a validation message on each item by setting the variable "singleValidation". The validation messages on the array items are then concatenated to the main formInstanceValidationMessages using the CreateValidationItem activity.

The custom validation sequence is placed in the Trigger section of the WaitForBookmark activity (labeled "Next") **from** the form on which you want to do the validation (Source: "PersonalDataSheet" in our example). The condition in the Next transition is set to "Not formInstance.ValidationMessages.HasErrors ". The custom validation message is displayed before the transition to the subsequent form (Destination: "Default-Frame" in our example).



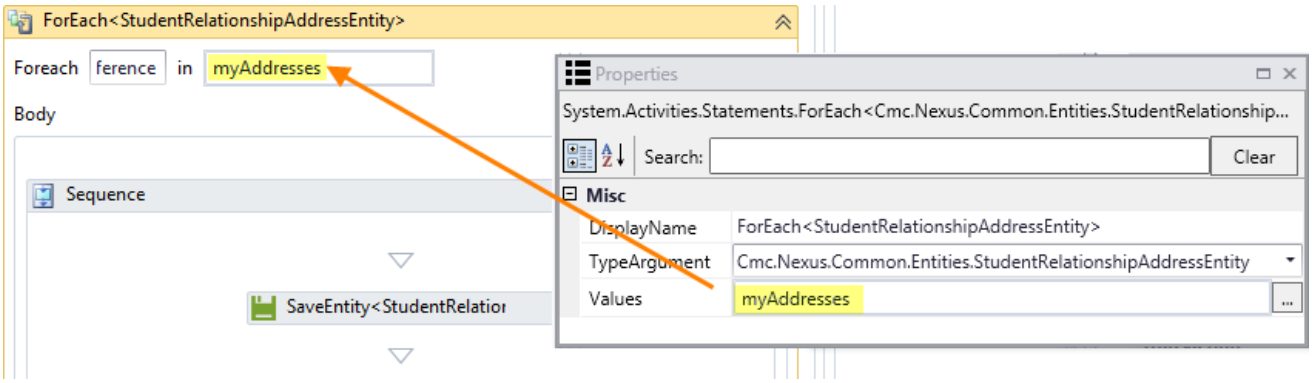
The following activities are used in the custom validation sequence:

- ForEach<>
- SaveEntity<>
- If
- CreateValidationItem
- InvokeMethod


Sequence
Next
ForEach<StudentRelationshipAddressEntity>
Foreach
referen
in
myAddresses
Body
Sequence
SaveEntity<StudentRelatior
If
Condition
singleValidation.HasErrors
Then
Else
Sequence
CreateValidationItem
Message
"Check address for Reference " & reference.FirstName
Message Type
Error
Drop activity here
InvokeMethod
TargetType
(null)
TargetObject
singleValidation
MethodName
Clear

- **ForEach<>**

  The ForEach<StudentRelationshipAddressEntity> activity loops through the reference addresses obtained through the "myAddresses" InArgument.



  The "myAddresses" argument is defined in the workflow as:

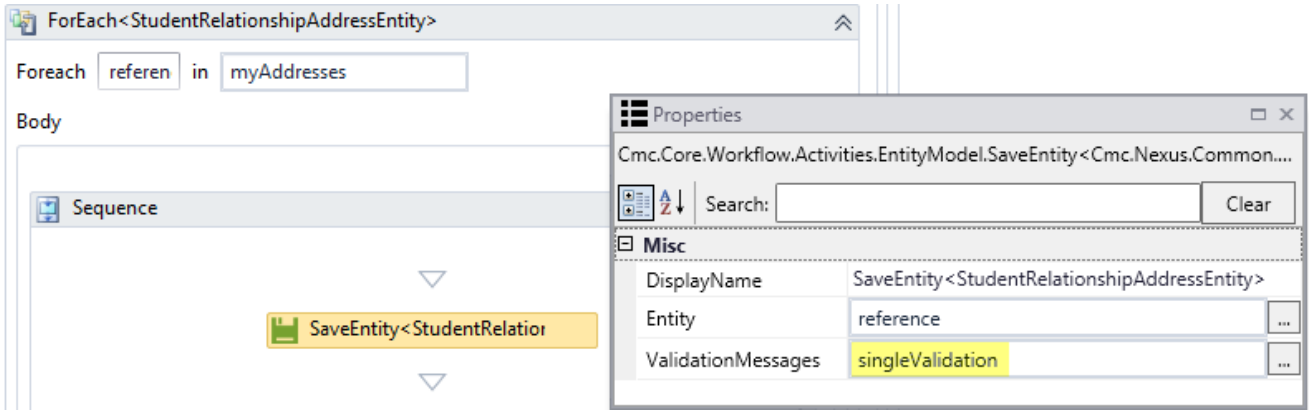
Name	Direction	Argument type	Default value
myAddresses	In/Out	StudentRelationshipAddressEntity[]	*Default value not supported*

  The myAddresses argument is set in the Model property on Text Box controls on the PersonalDataSheet form as:

  - vm.models.myAddresses[0].FirstName
  - vm.models.myAddresses[0].LastName
  - vm.models.myAddresses[0].RelationToStudent
  - vm.models.myAddresses[0].StreetAddress
  - vm.models.myAddresses[0].City
  - vm.models.myAddresses[1].FirstName, etc.

- **SaveEntity<>**

  The SaveEntity<StudentRelationshipAddressEntity> activity uses the "singleValidation" variable as InArgument for the ValidationMessages collection.

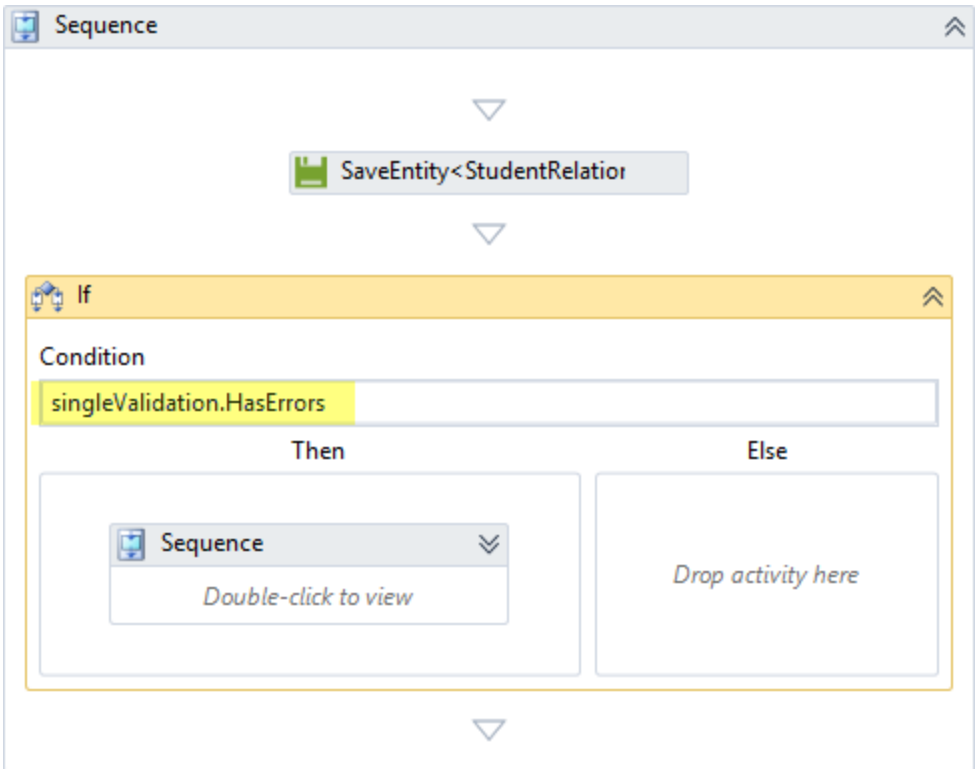


The "singleValidation" variable is defined in the workflow as:

Name	Variable type	Scope	Default
singleValidation	ValidationMessageCollection	SMPersonalDataSheet	New ValidationMessageCollection

- **If**

  The If activity uses the Condition "singleValidation.HasErrors". If errors are found, the sequence containing the CreateValidationItem activity is executed.



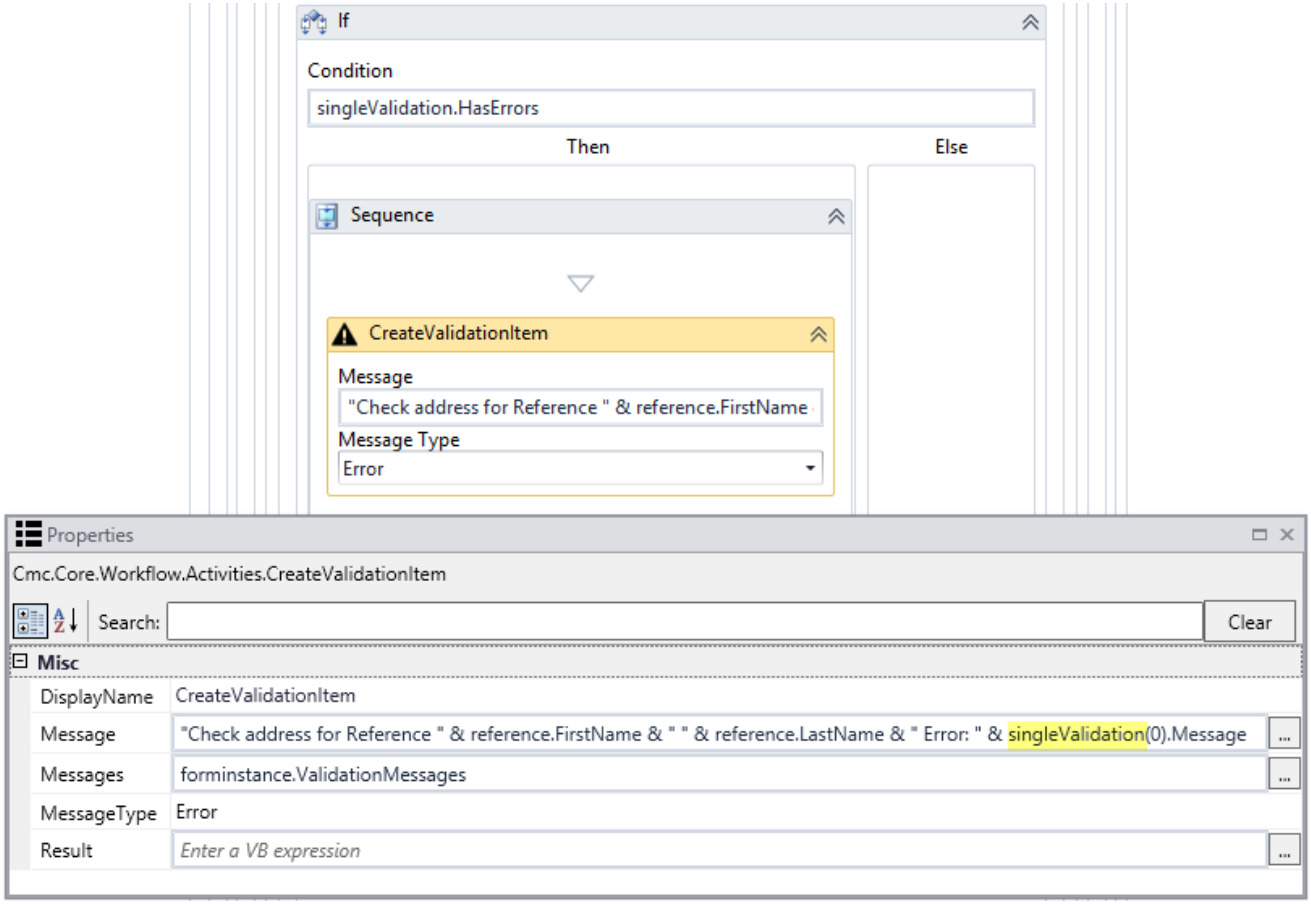
- **CreateValidationItem**

  The CreateValidationItem activity is set to a Message Type of "Error" for the InArgument of "formInstanceValidationMessages".

  The Message string displayed by the activity is defined as:

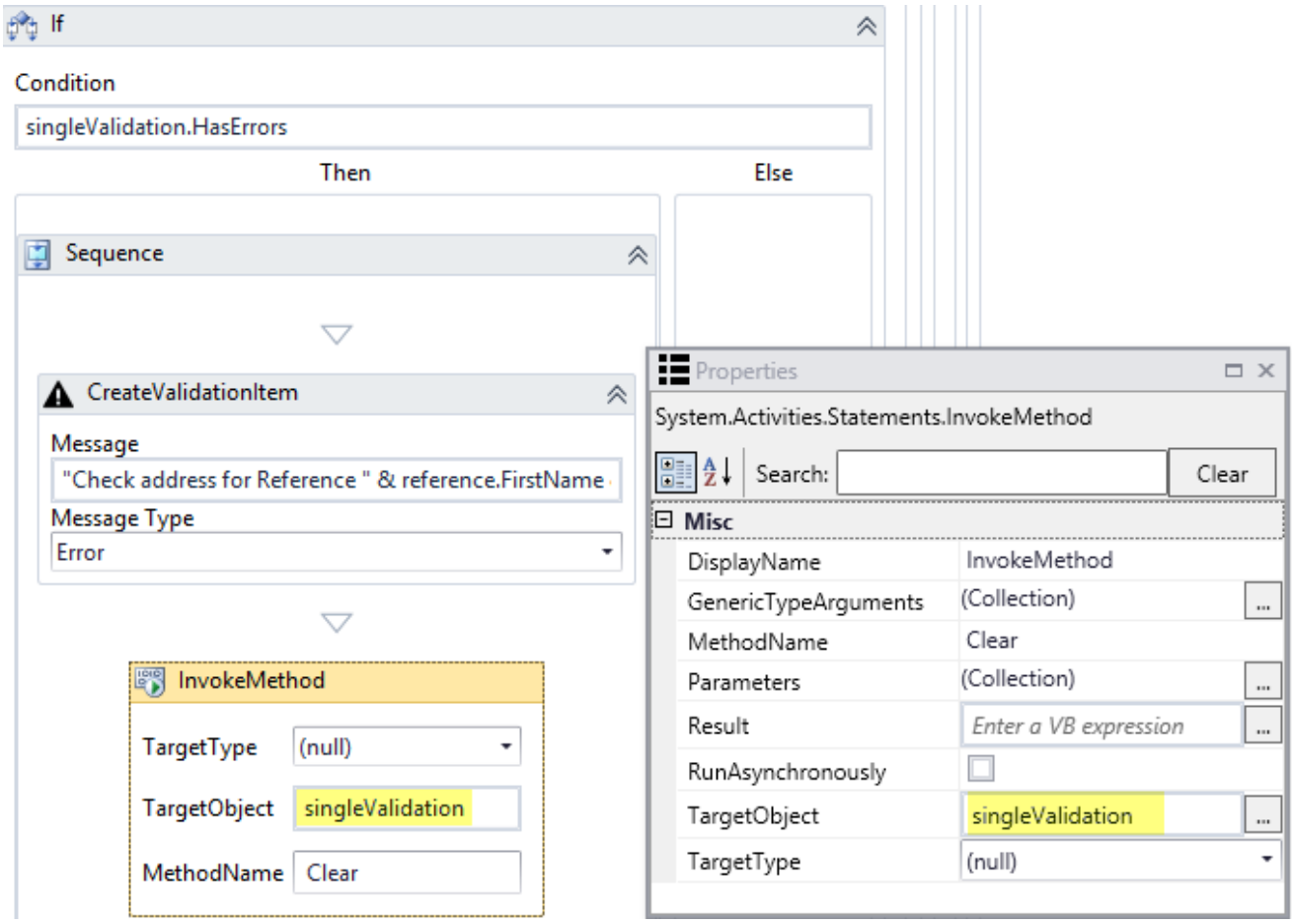
  "Check address for Reference " & reference.FirstName & " " & reference.LastName & " Error: " & **singleValidation**(0).Message

  Note that the "singleValidation" variable is used to hold the validation value for each item in the array.



- **InvokeMethod**

  The InvokeMethod activity is used to clear a singular collection for the next iteration of the loop. Otherwise the failure of a previous item in the array would remain and trigger the next item to show a failure as well.


If
Condition
singleValidation.HasErrors
Then
Else
Sequence
CreateValidationItem
Message
"Check address for Reference " & reference.FirstName
Message Type
Error
InvokeMethod
TargetType
(null)
TargetObject
singleValidation
MethodName
Clear
Properties
System.Activities.Statements.InvokeMethod
Search:
Clear
Misc
DisplayName
InvokeMethod
GenericTypeArguments
(Collection)
MethodName
Clear
Parameters
(Collection)
Result
Enter a VB expression
RunAsynchronously
TargetObject
singleValidation
TargetType
(null)

Grid Using Entity Collection Activities

When you create forms sequences that allow users to add or edit items in a grid (e.g., array of documents or addresses), you can use the GetEntityCollection<> and SaveEntityCollection<> workflow activities to retrieve an existing array from the database and save any new or modified items.

The GetEntityCollection<> and SaveEntityCollection<> activities are available in Workflow Composer version 2.7 and later and require the following **minimum** versions of activities and contracts:

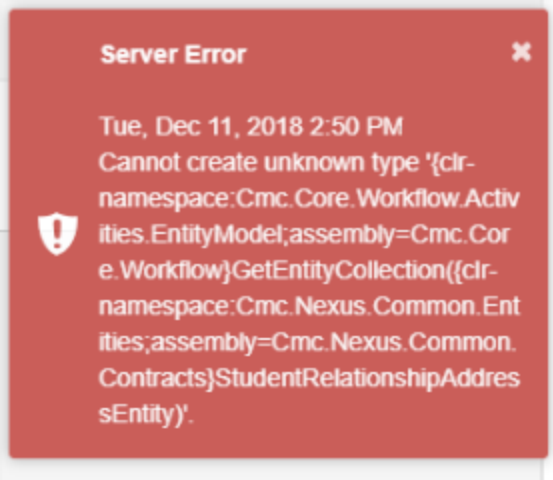
- Anthology Student version 20.0.x

  — OR —

- CampusNexus CRM version 12.2.x

The minimum Cmc.Core.dll version installed in Program Files (x86)\CMC\Workflow must be 5.1.167 or greater.

If you use these activities with Student 19.0 and Workflow Composer 2.7, you won't find any errors in Workflow Composer (because it has minimum Cmc.Core.dll version), but you'll have a server error at runtime.

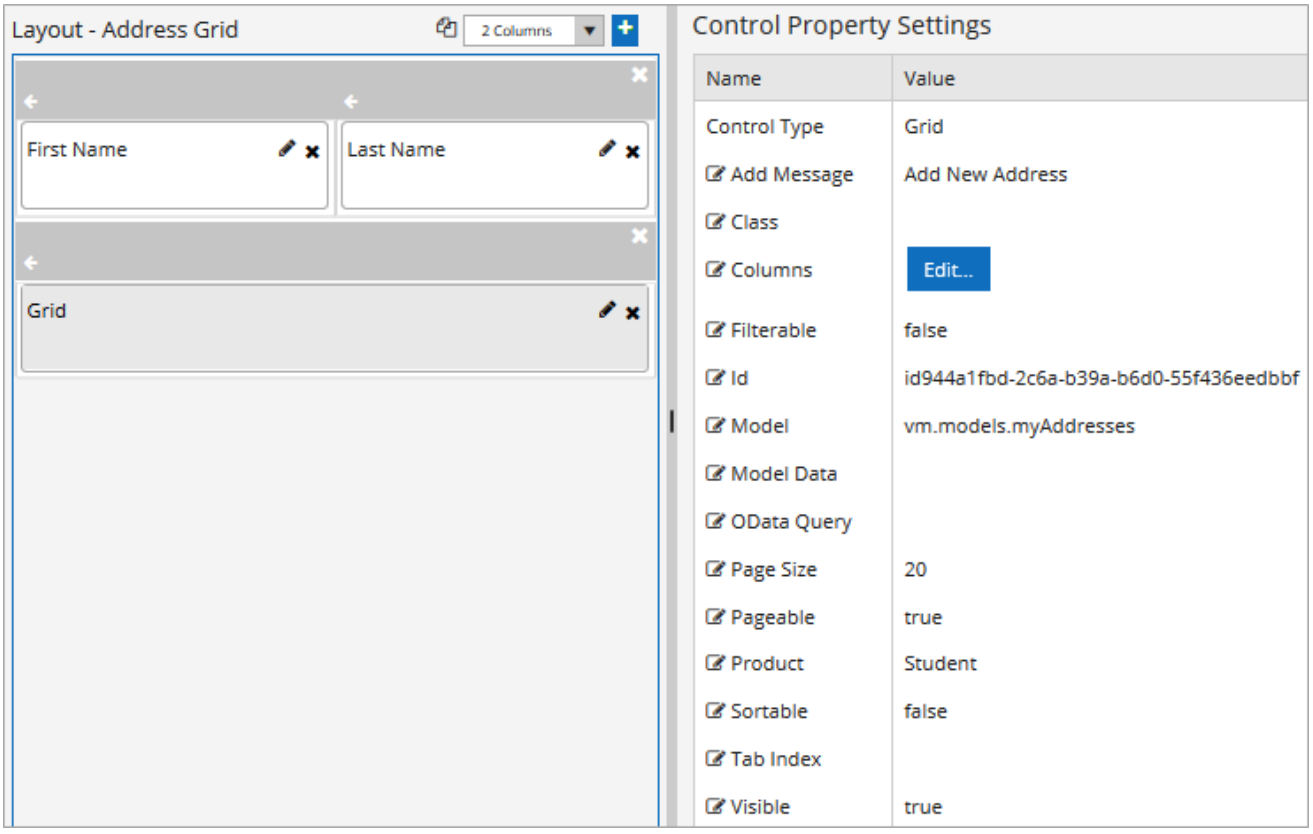


The following section provides a detailed example of the workflow activities required to add/edit and save records in an entity collection.

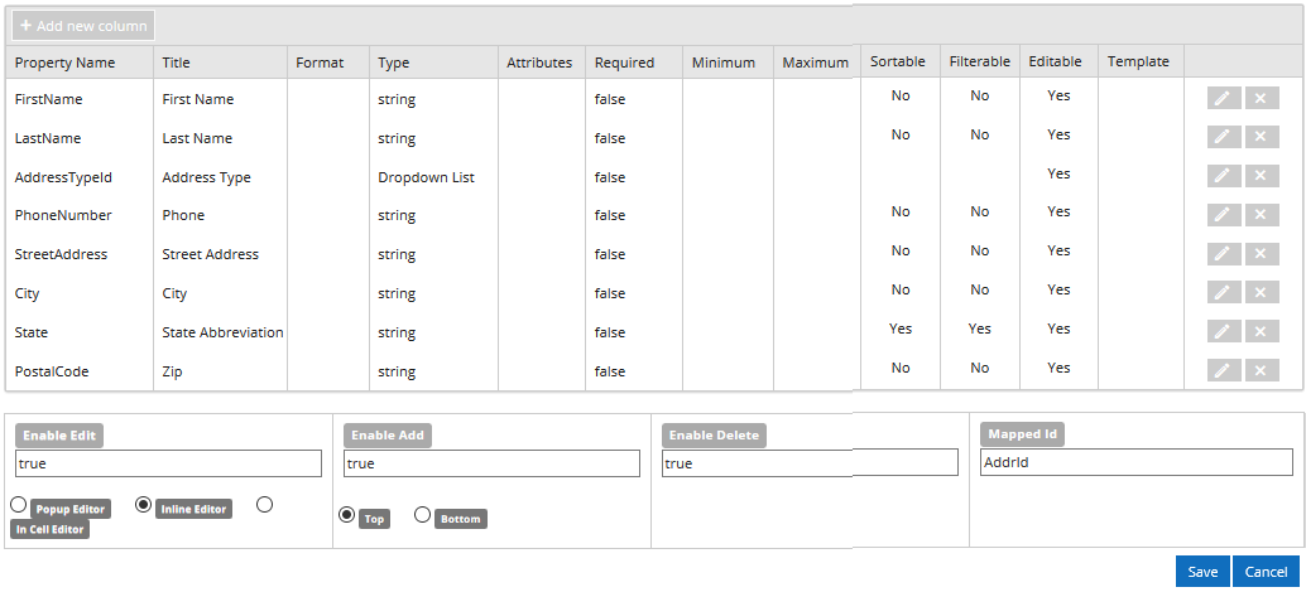
Add, Edit, and Save Records in a Collection

This workflow example is associated with a Forms Builder sequence that retrieves a collection of records for the StudentRelationshipAddressEntity and exposes the records in a grid control. The user of the form sequence is allowed to add and edit data in the grid. The new and modified records are saved to the database.

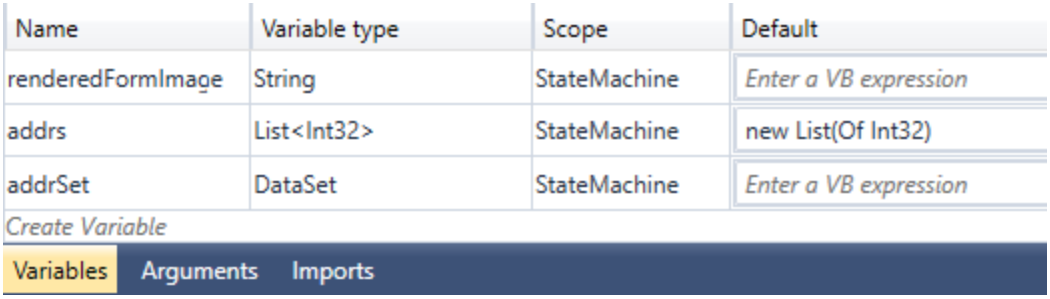
1. In Form Designer, create a form using the **Grid** component.
2. Bind the Grid component to the workflow using the **Model** property value **vm.models.myAddresses**.



3. Configure the **Columns** property to allow the user to add, edit, and delete data.



4. In Sequence Designer, add the form to a sequence.
5. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
6. In Workflow Composer, create the variables shown below.

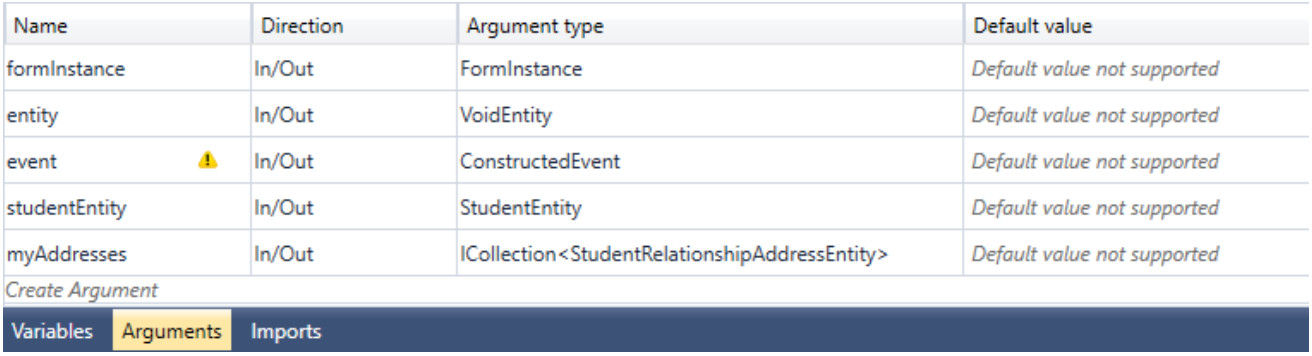


Name	Variable type	Scope	Default
renderedFormImage	String	StateMachine	*Enter a VB expression*
addrs	List<Int32>	StateMachine	new List(Of Int32)
addrSet	DataSet	StateMachine	*Enter a VB expression*

Create Variable

Variables | Arguments | Imports

7. Create an argument of type **ICollection<StudentRelationshipAddressEntity>** for the **myAddresses** model value that binds the grid to the workflow. The path to browse to the argument type is: System.Collections.Generic.ICollection<Cmc.Nexus.Common.Entities.StudentRelationshipAddressEntity>.



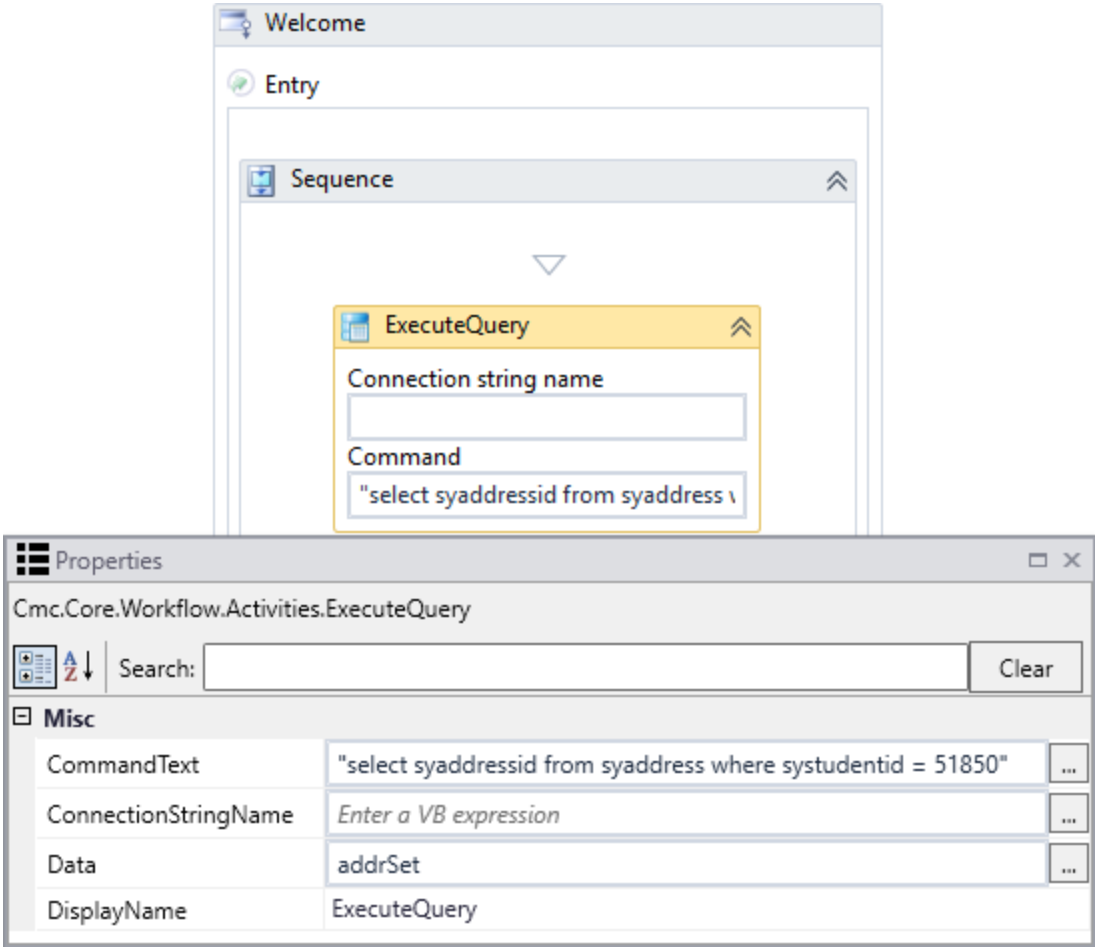
Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event	In/Out	ConstructedEvent	*Default value not supported*
studentEntity	In/Out	StudentEntity	*Default value not supported*
myAddresses	In/Out	ICollection<StudentRelationshipAddressEntity>	*Default value not supported*

Create Argument

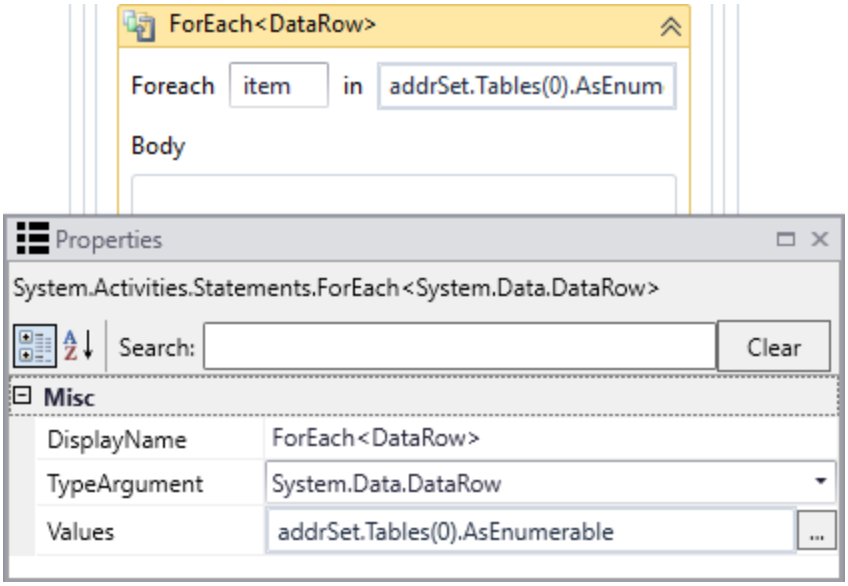
Variables | Arguments | Imports

8. The GetEntityCollection<> activity needs a list of ids for the collection of the same entity type to retrieve. To achieve this, drag an **ExecuteQuery** activity into the Entry section of the Welcome form. This activity retrieves a set of document ids for a student from the database and returns the data in a variable named **addrSet** (refer to variables created above).

   The Command property is defined as **"select syaddressid from syaddress where systudentid = 51850"** where the systudentid value is hard-coded. Use a variable for the systudentid as appropriate in your environment.



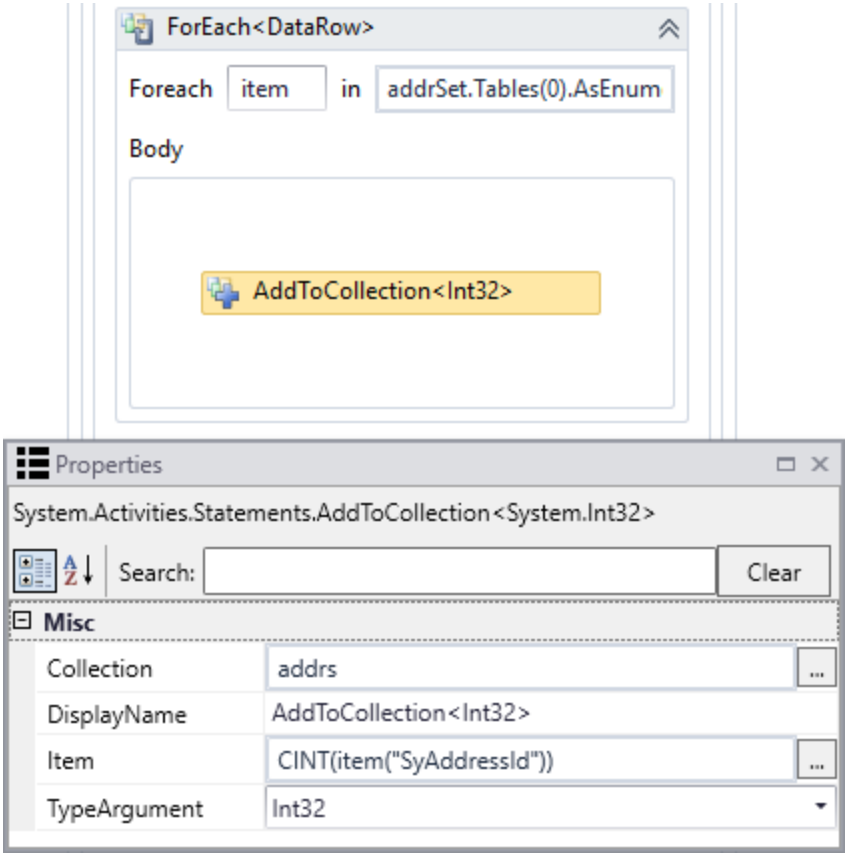
9. Drag a **ForEach<>** activity below the ExecuteQuery activity. The ForEach<> is activity converts the dataset type argument returned by ExecuteQuery to a collection of Int32 ids to pass to GetEntityCollection<> activity using the Values property **addrSet.Tables(0).AsEnumerable**.



10. Drag an **AddToCollection<>** activity into the Body section of the ForEach<> activity. The AddToCollection activity adds items to the collection when users enter new data on the form.

    The collection is defined by the variable **addrs** of type **List<Int32>** with a default value of **new List(Of Int32)**.

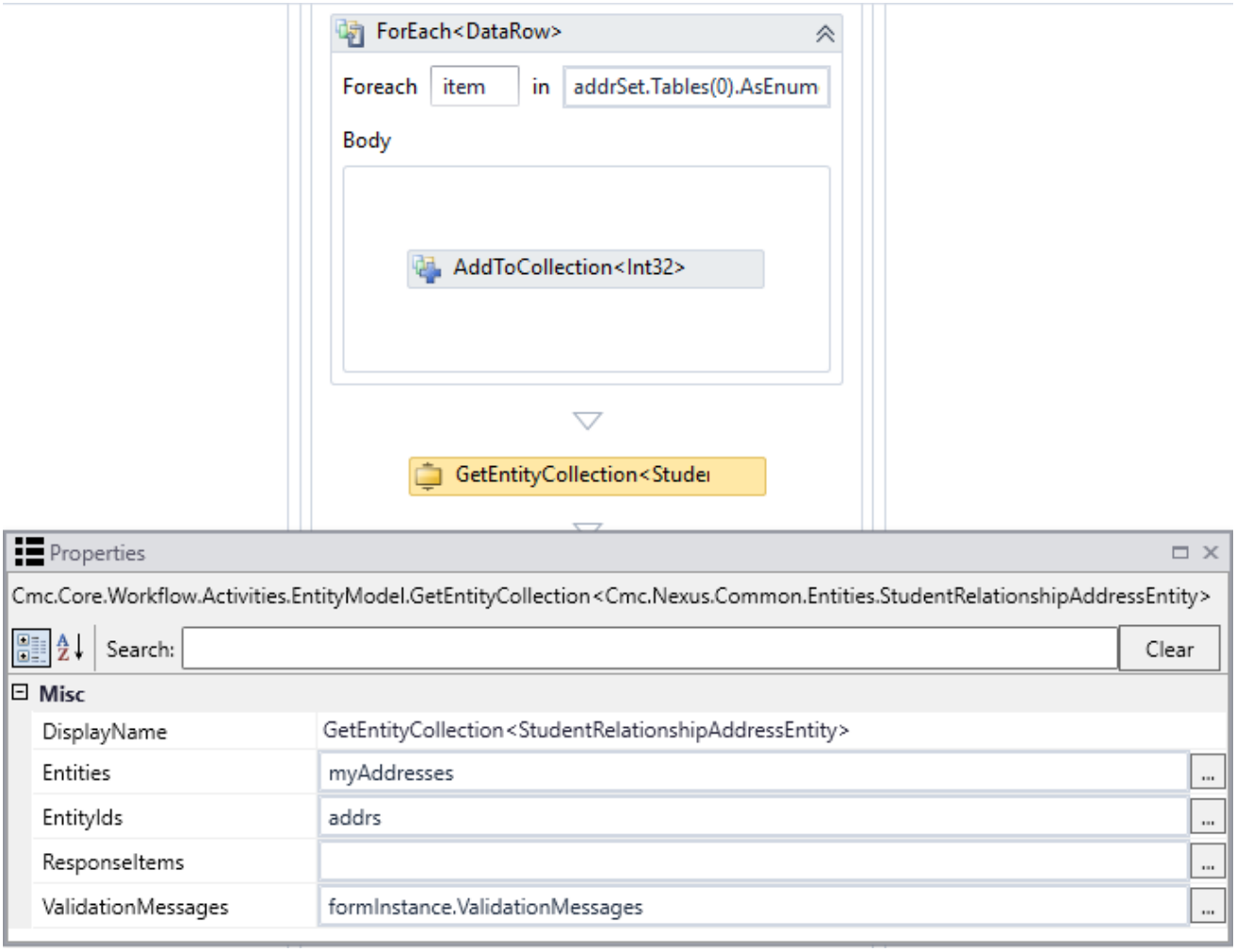
    The **Item** property value **CINT(item("SyAddressId"))** converts the data to integers.



11. Drag a **GetEntityCollection<>** activity below the ForEach<> activity. The GetEntityCollection<> activity uses the **StudentRelationshipAddressEntity**.

    The input argument is the **addrs** variable.

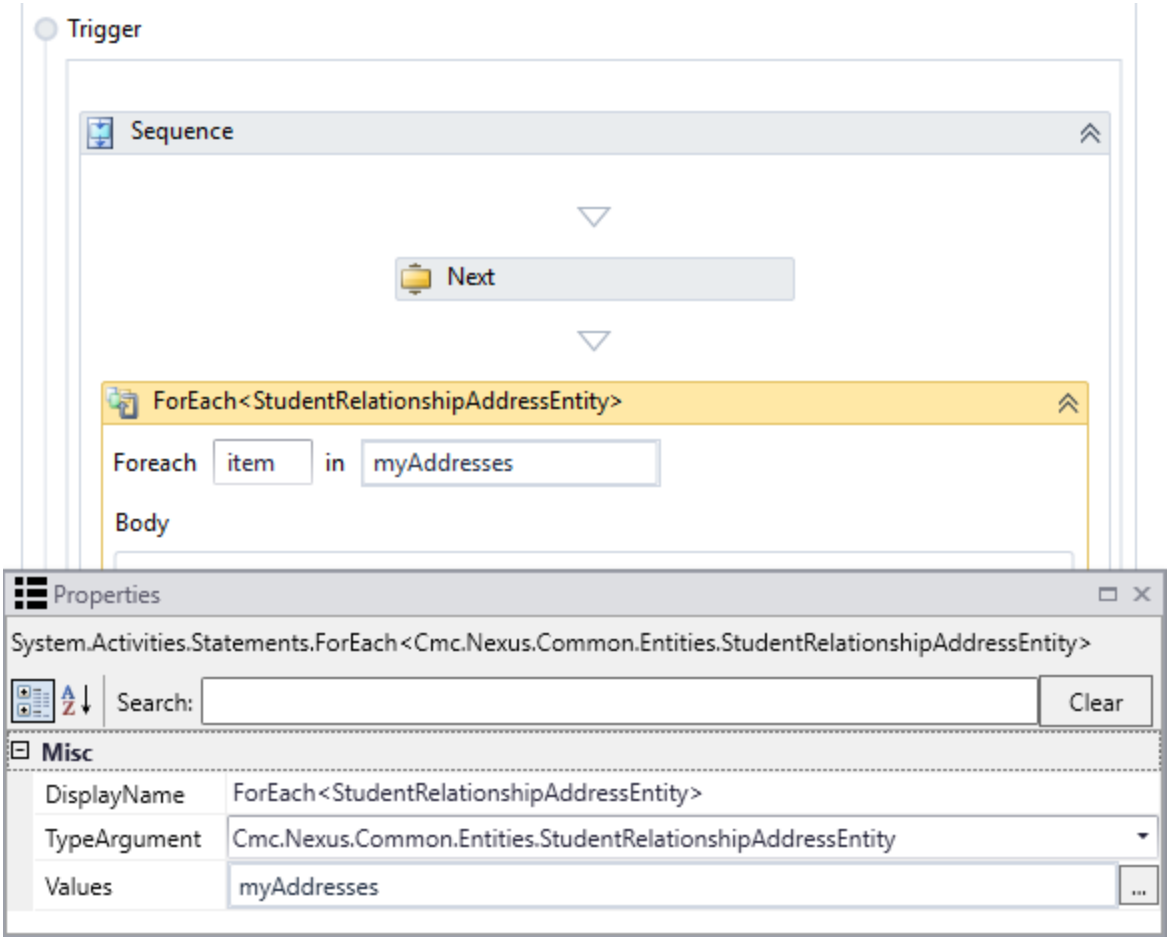
    The output argument is the **myAddresses** argument that binds the grid to the workflow.



12. Drag a **ForEach<>** activity into the Next transition following the form the contains the Grid component.

    The Values property holds the **myAddresses** argument that binds the grid to the workflow.

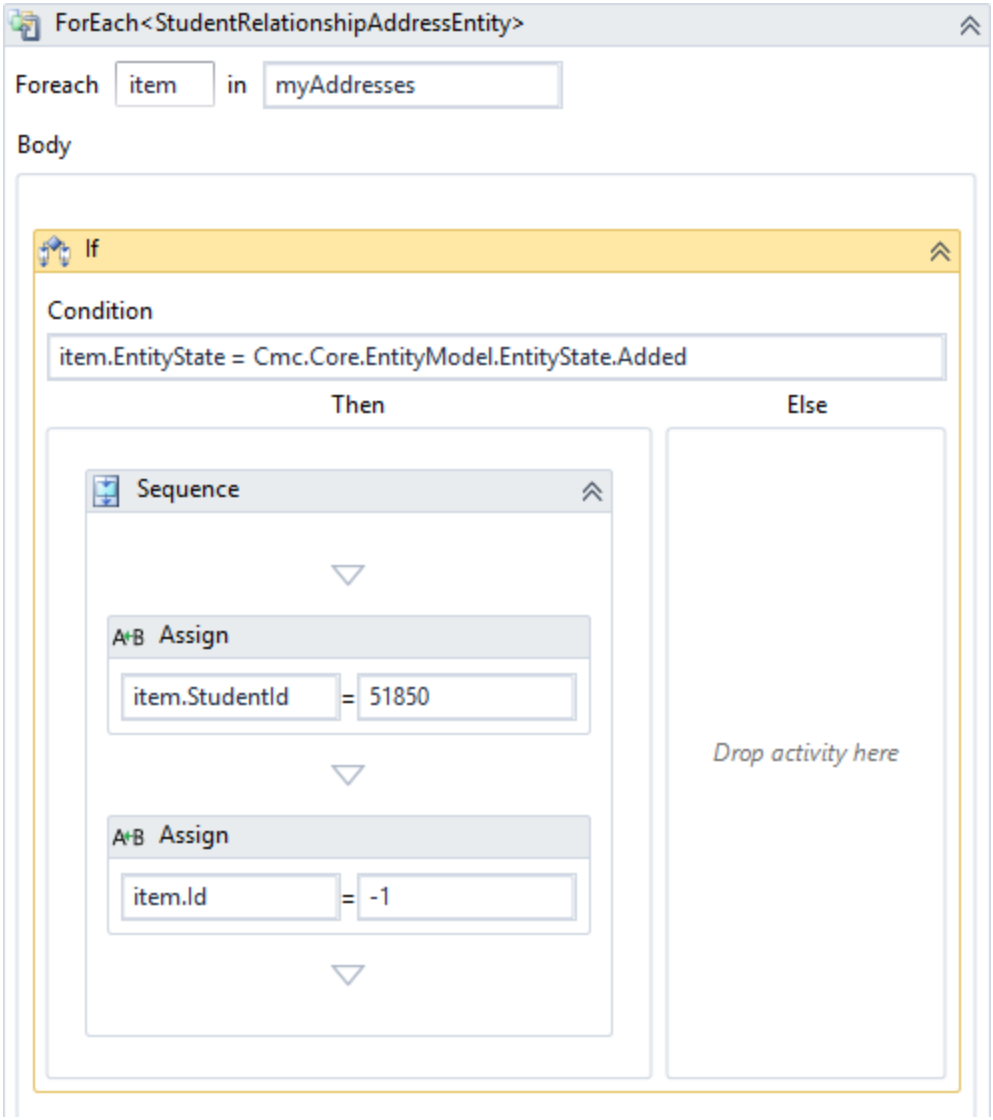
    This instance of the ForEach activity gathers all rows in the grid including rows that were added by the form user.



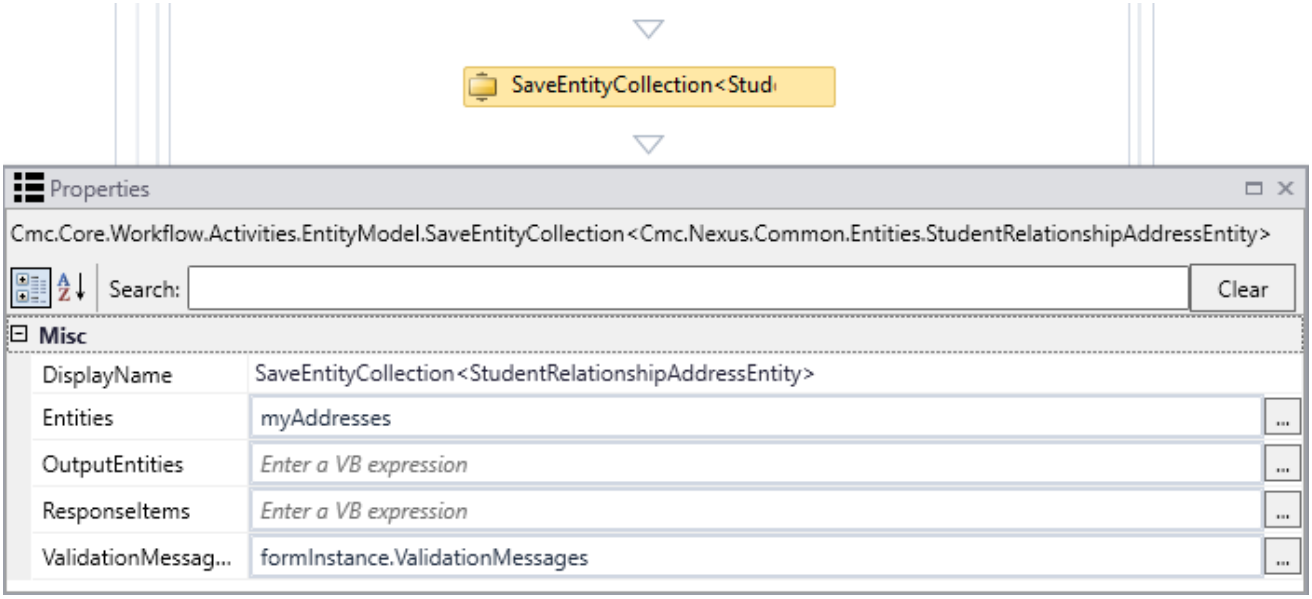
13. Drag an **If** activity into the Body section of the ForEach<> activity. Specify the following condition to detect if an item was added to the StudentRelationshipAddressEntity:
item.EntityState = Cmc.Core.EntityModel.EntityState.Added

    Drag an Assign activity into the **Then** branch to the associate the hard-coded studentid with the itemEntityState array.

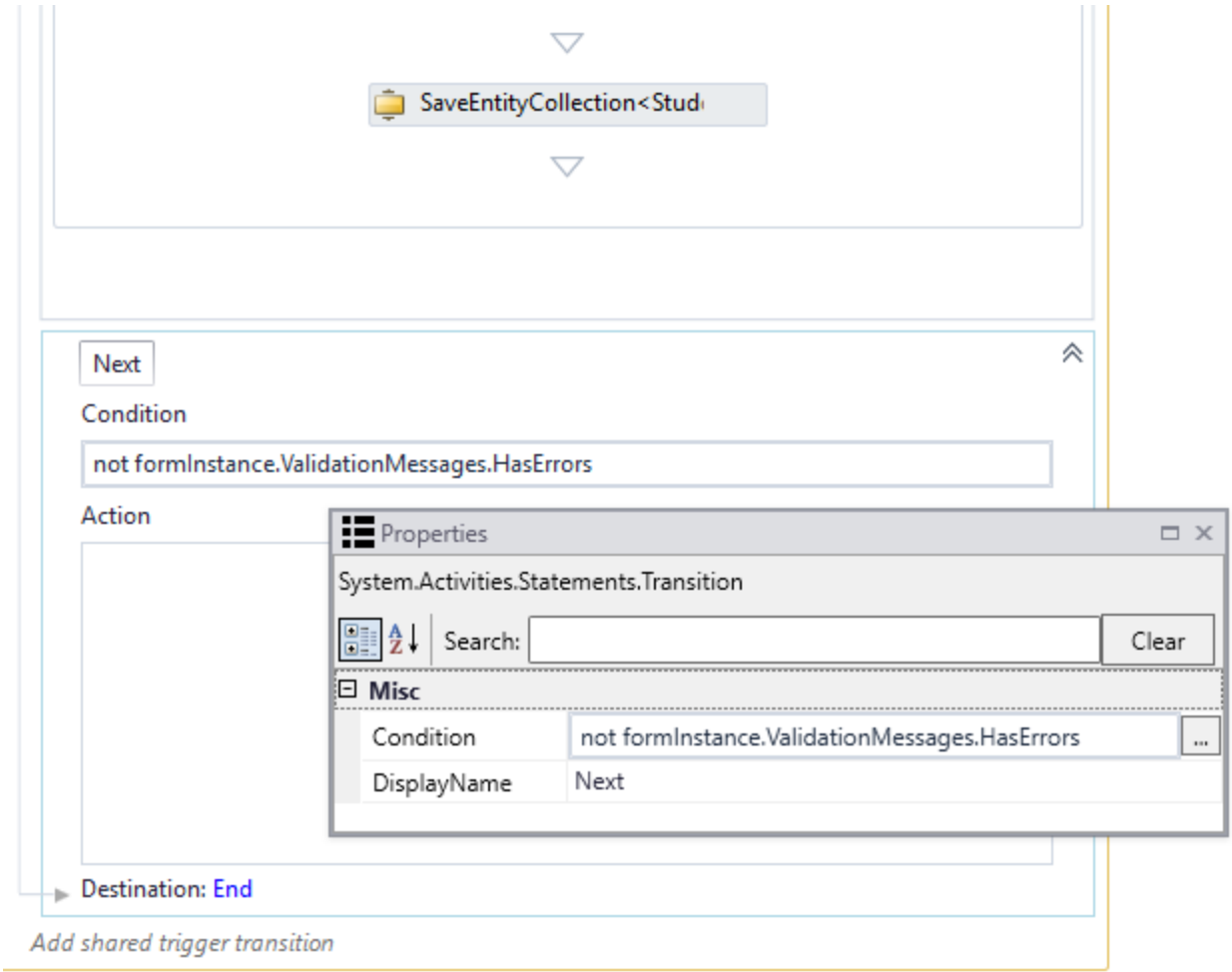
    Add another Assign activity to set the **item.Id** to **-1**. This assign statement ensures that a new item is appended to the array. The last element of an array is the length of the array - 1.



14. Drag a **SaveEntityCollection<>** activity into last Next transition of the sequence. The activity will handle add, edit and delete of any entity in the. In our example, the activity saves the changes passed in through **myAddresses** to the ICollection<StudentRelationshipAddressEntity>.



15. Finally, in the Condition field of the last Next transition, specify **not formInstance.ValidationMessages.HasErrors** to catch any validation errors.



Link a Portal Account to a Student Record

This procedure is applicable only for environments that use the Anthology Student product.

When applicants or students create new accounts via the CMCPortalSTS (refer to Renderer Authentication), a `wpUserId` (web portal user Id) is generated for them alongside the User Name they choose. The `wpUserId` must be linked to the `syStudent` record in the Anthology Student database.

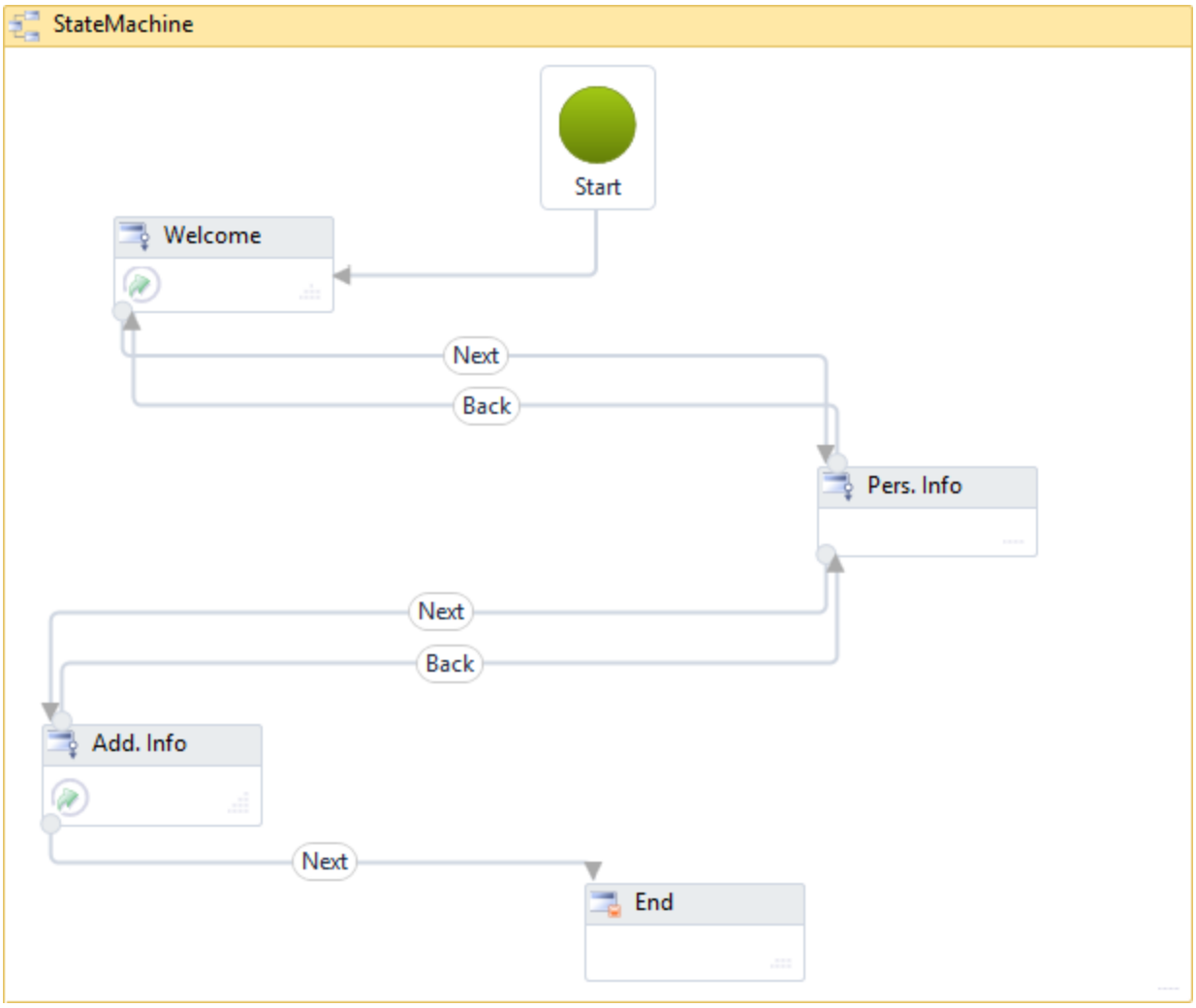
The workflow activity SaveStudentPortalUserAssociation (refer to Workflow Composer Help) is used to create a `wpUserRelation` record in the Portal database. The `wpUserRelation` record establishes a relation between a `wpUser` record in the Portal database and an `syStudent` record in the Anthology Student database.

The following example shows how the SaveStudentPortalUserAssociation activity can be used in a workflow for a form sequence:

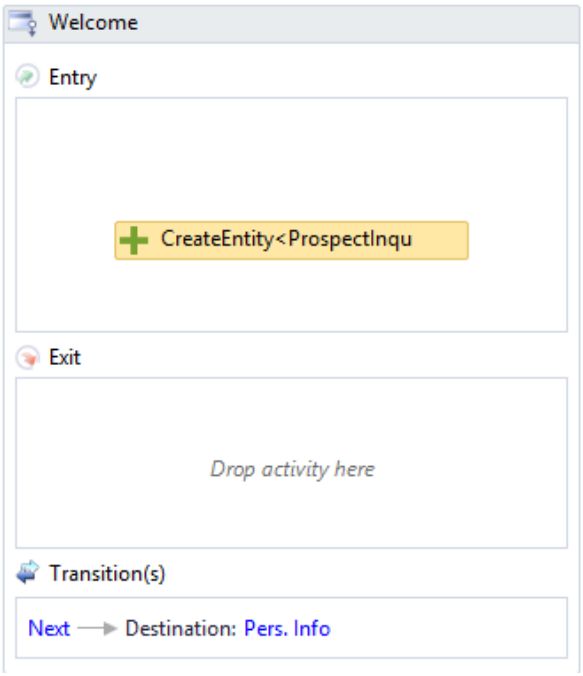
1. A sequence consisting of the following forms is created:

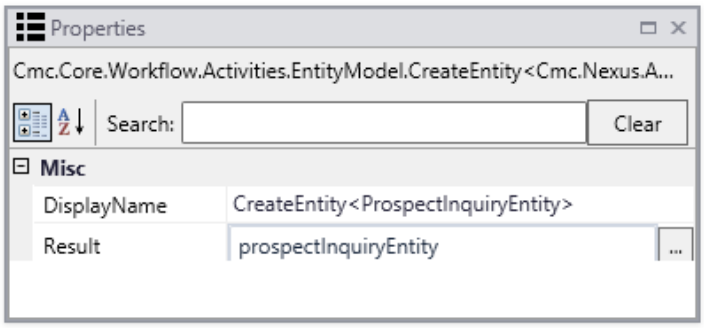
    - Welcome
    - Personal Information (Pers. Info)
    - Additional Information (Add. Info)
    - End

    The StateMachine workflow for the sequence contains the forms (States) and transitions (Next, Back).

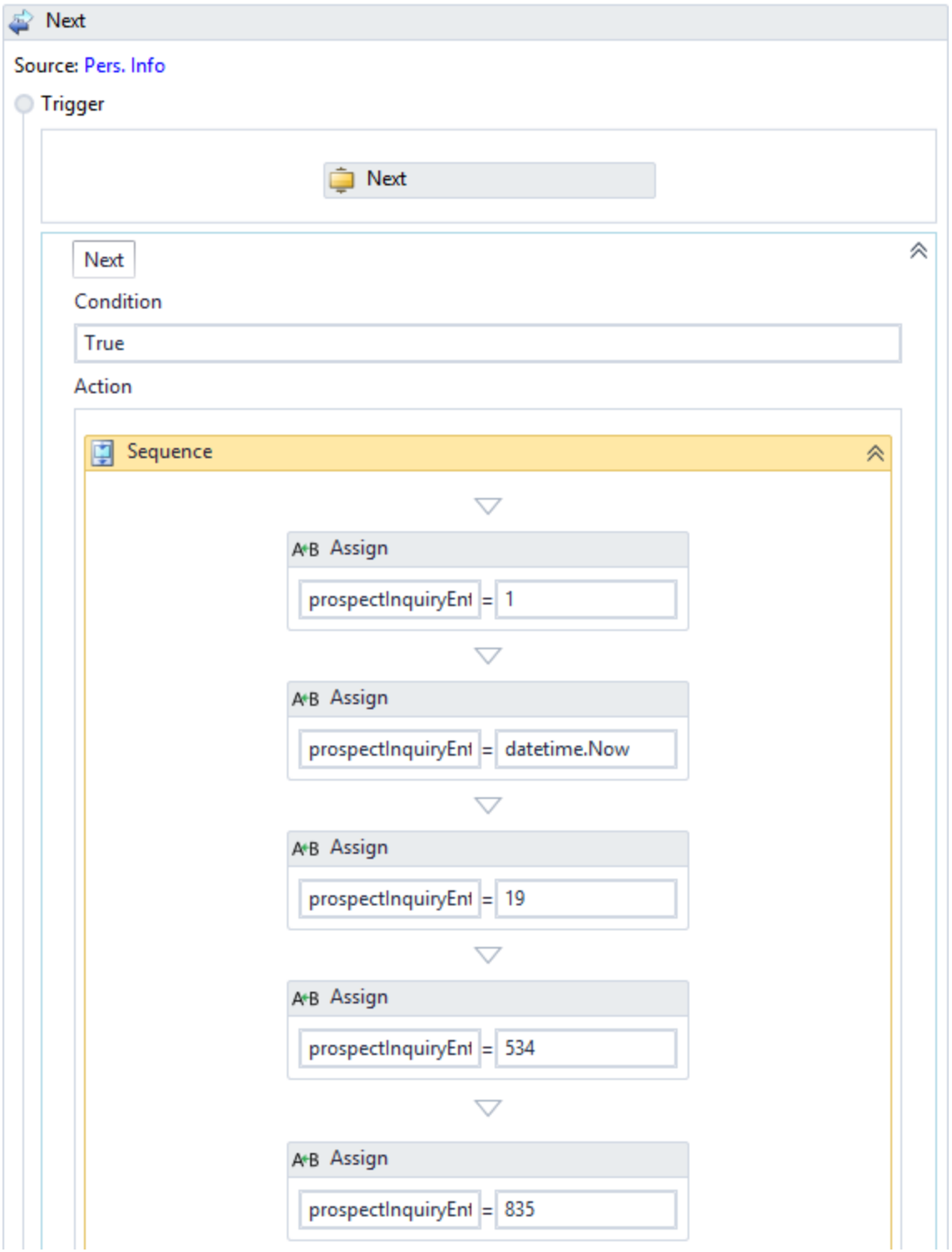


2. Add a **CreateEntity** activity in the Entry area of the Welcome form to create a **propectInquiryEntity**.



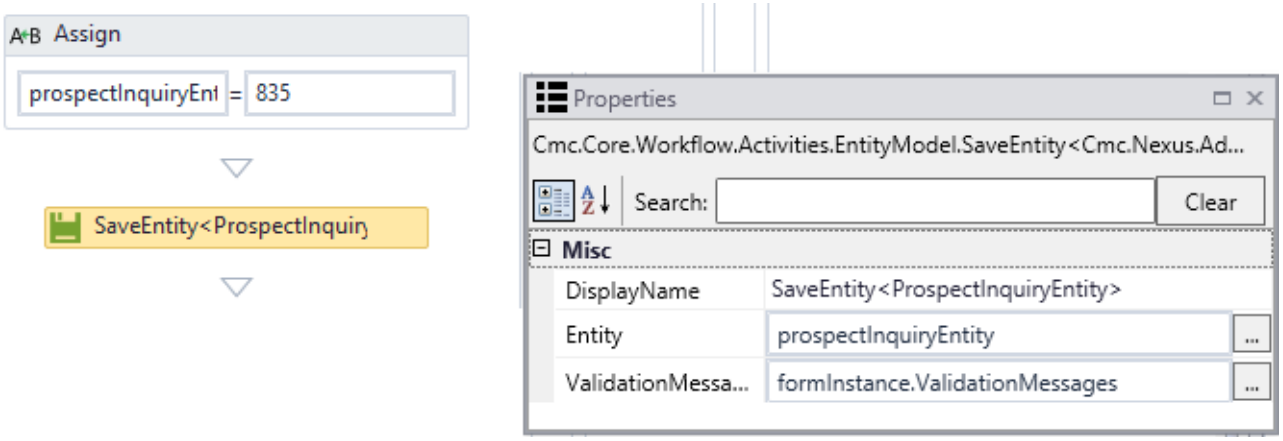


3. In the Action area of the Next transition from the Pers. Info form, set the Condition to True and add a Sequence with **Assign** activities to capture the **prospectInquiryEntity** attributes that are entered on the form, e.g., Student.SchoolStatusId, LeadDate, LeadTypeId, AssignedAdmissionsRepId, and LeadSourceId.

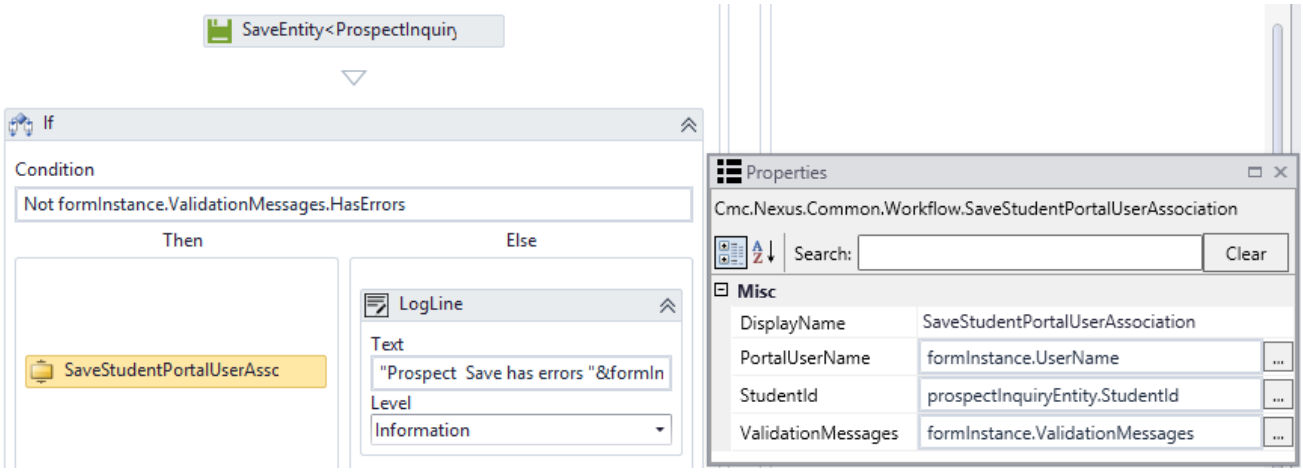


4. Add a **SaveEntity** activity below the Assign activities to save the prospectInquiryEntity record with the assigned values.

   This activity creates a `wpUser` record in the Portal database.



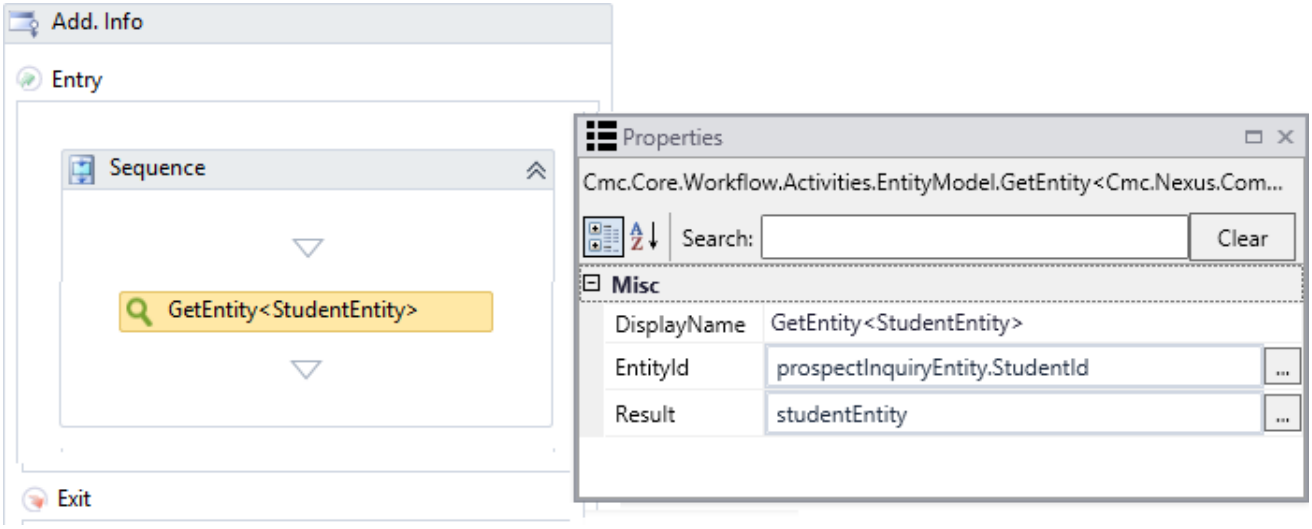
5. Add an **If** activity below the SaveEntity activity.



- In the Condition field, specify the following to capture form errors:
  - Not formInstance.ValidationMessages.HasErrors
- In the Then branch, add a **SaveStudentPortalUserAssociation** activity with the following properties:
  - PortalUserName = formInstance.UserName
  - StudentId = prospectInquiryEntity.StudentId
  - ValidationMessages = formInstance.ValidationMessages
- In the Else branch, add a **LogLine** activity to write the following error message to the log:
  - "Prospect Save has errors "&formInstance.ValidationMessages(0).Message

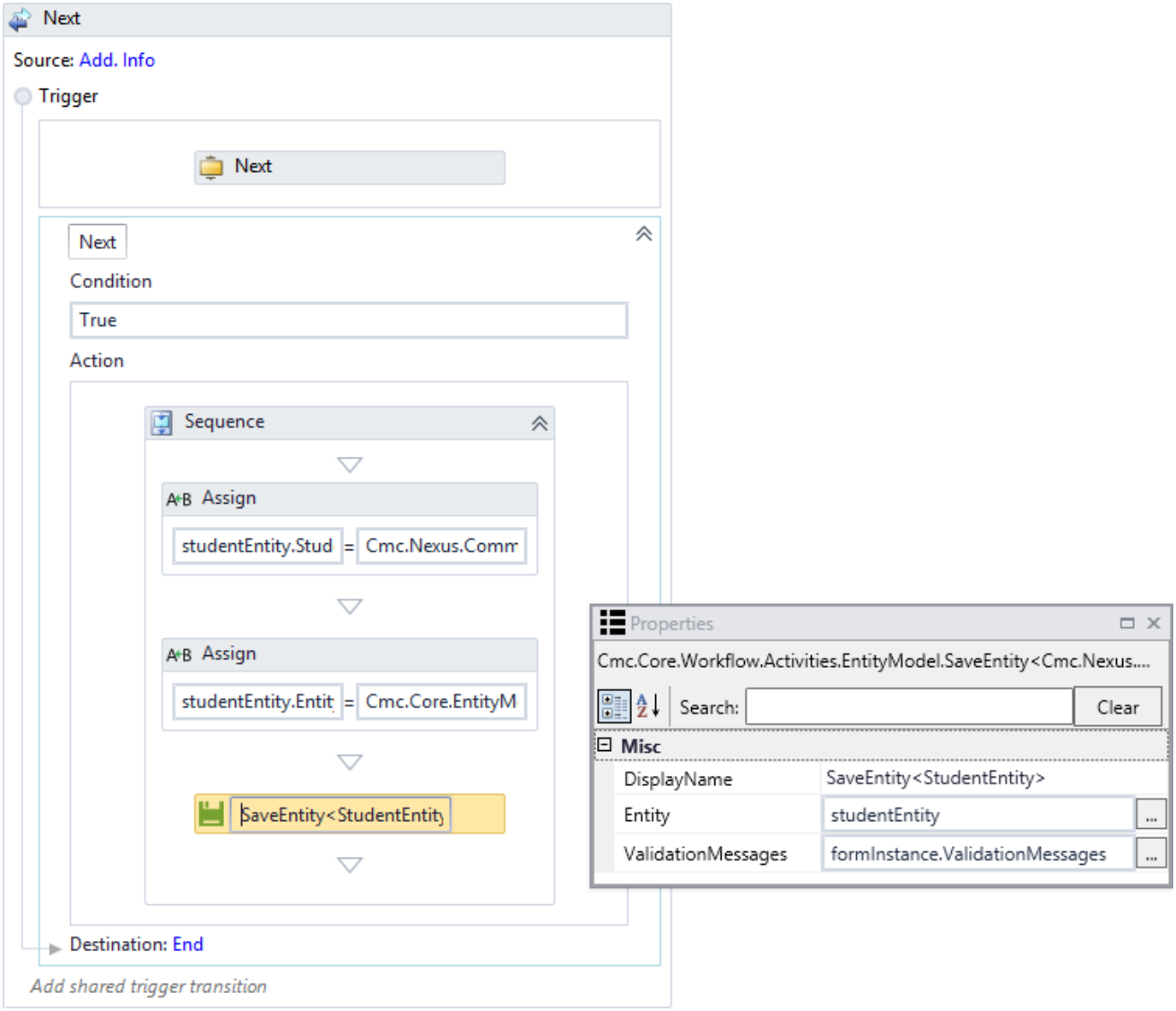
  We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

6. In the Entry area of the Add. Info form, add a Sequence with a GetEntity activity.



- In the **GetEntity** activity, specify the following properties:
    - EntityId = propspectInquiryEntity.StudentId
    - Result = studentEntity

7. In the Action area of the Next transition from the Add. Info form, set the Condition to True and add a Sequence with two **Assign** activities followed by a **SaveEntity** activity.



- In the first Assign activity, specify the following:
    - studentEntity.StudentAddressAssociation = Cmc.Nexus.Common.Entities.StudentAddressAssociation.IgnoreInStudentAssociation.
- In the second Assign activity, specify the following:
    - studentEntity.EntityState = Cmc.Core.EntityModel.EntityState.Modified
- In the SaveEntity activity, specify the following:
    - Entity = studentEntity
    - ValidationMessages = formInstance.ValidationMessages.

The SaveEntity activity creates a `syStudent` record in the Anthology Student database.

Multi Route Forms

In Forms Builder 2.x, the Multi Route Rule was used to control the flow of forms and sequences dynamically based on a user's selection in a form field. The Multi Route Rule enabled you to define which form or sequence was displayed when a user selected a specific value in a form field.

In Forms Builder 3.x and later, the multi route functionality is implemented using sequence workflows. For general information about workflows created through Sequence Designer, refer to State Machine Workflows.

Multi route sequences consists of multiple forms, at minimum a main form and two or more sub forms linked to the options available on the main form.

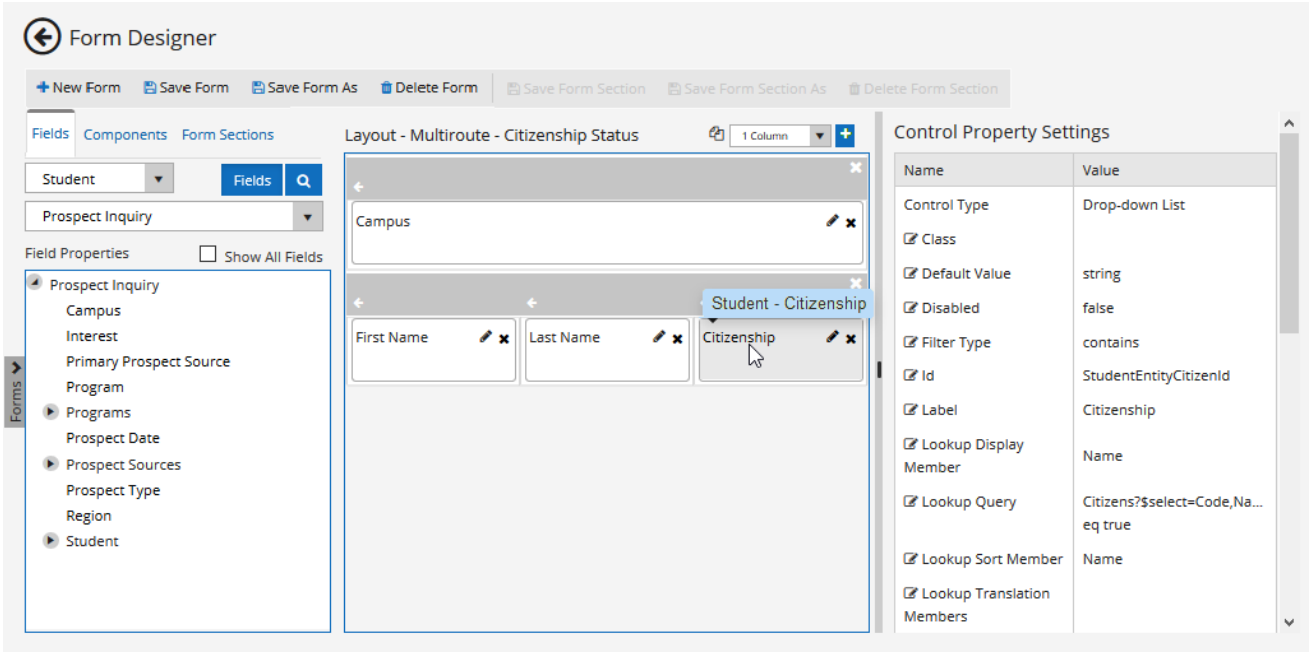
Example

The main form allows the user to select a citizenship status, e.g., US Citizen or Eligible Non-Citizen. If US Citizen is selected, a sub form presents a text field to enter the Social Security Number. If Eligible Non-Citizen is selected, a sub form presents field to enter the Alien Number.

Step 1: Create forms for the sequence

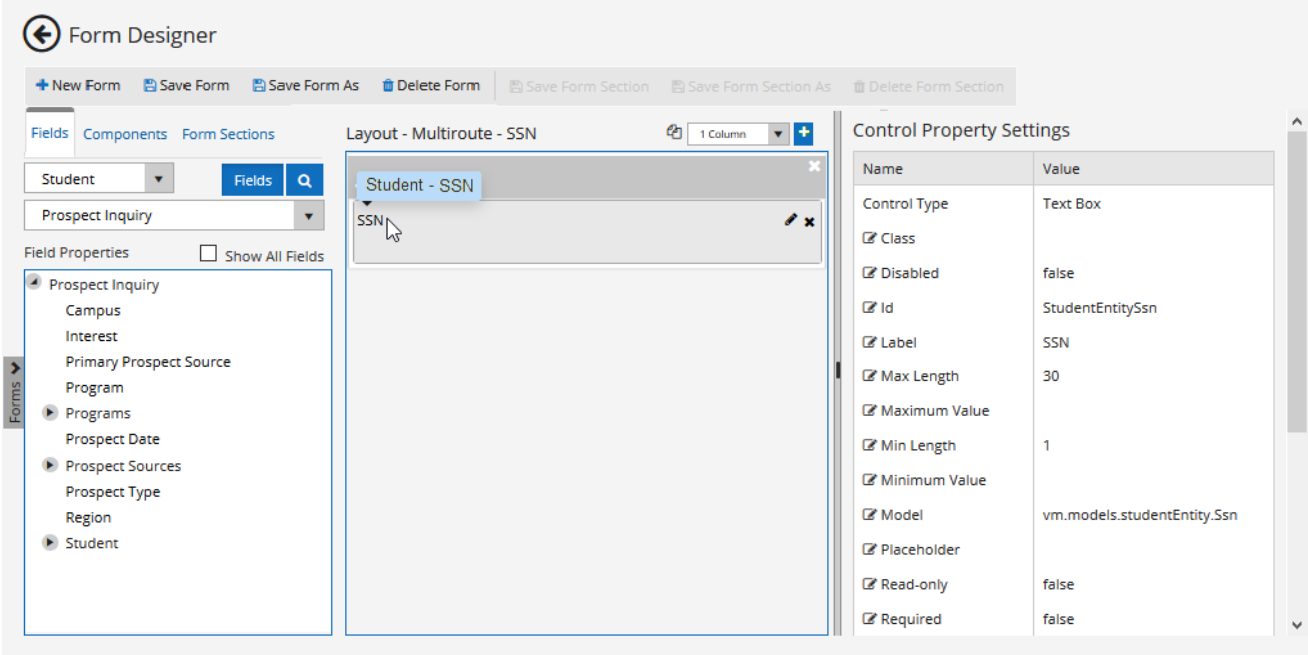
1. Browse to Form Designer to create and save the main form. The main form contains the field(s) that trigger the branching into multiple routes.

   In our example the value selected in the Student - Citizenship drop-down list will trigger the branching.

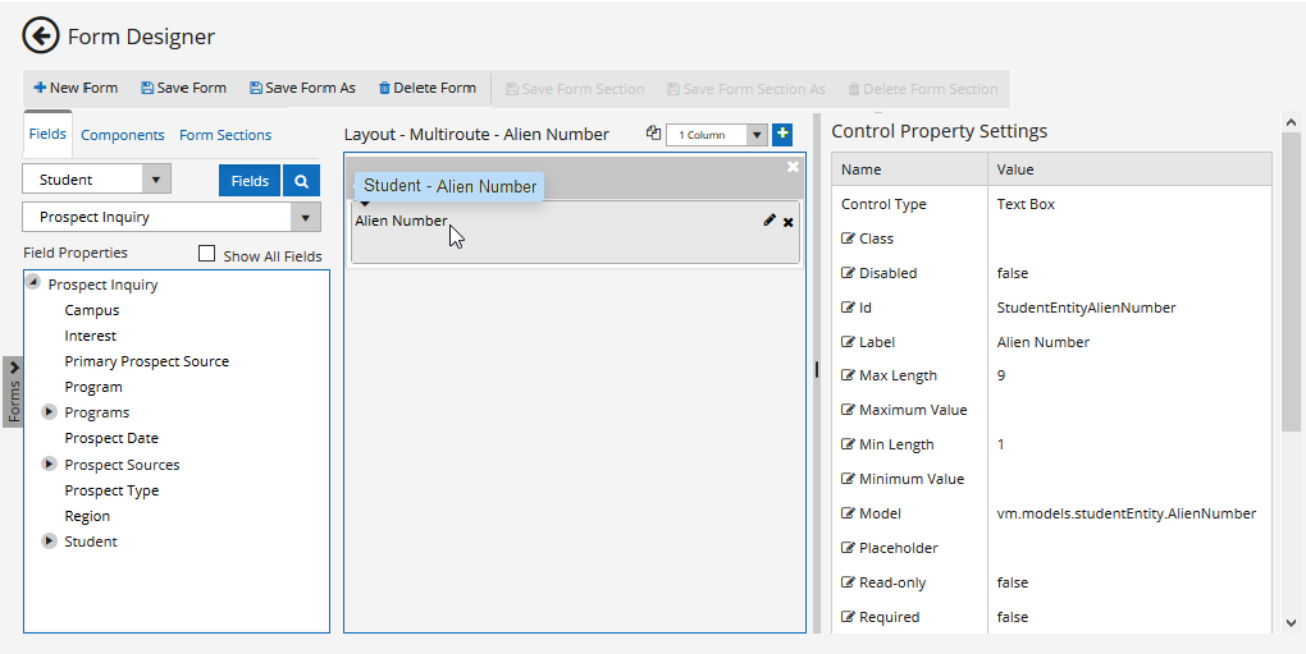


2. In Form Designer, create and save the sub forms. A sub form is required for each route.

In our example one sub form (form A) is created for the input of the social security number (SSN). This form is displayed when the user selects US Citizen in the Student - Citizenship Status drop-down list on the main form.



In our example a second sub form (form B) is created for the input of the Alien Number. This form is displayed when the user selects Eligible Non-Citizen in the Student - Citizenship Status drop-down list on the main form.



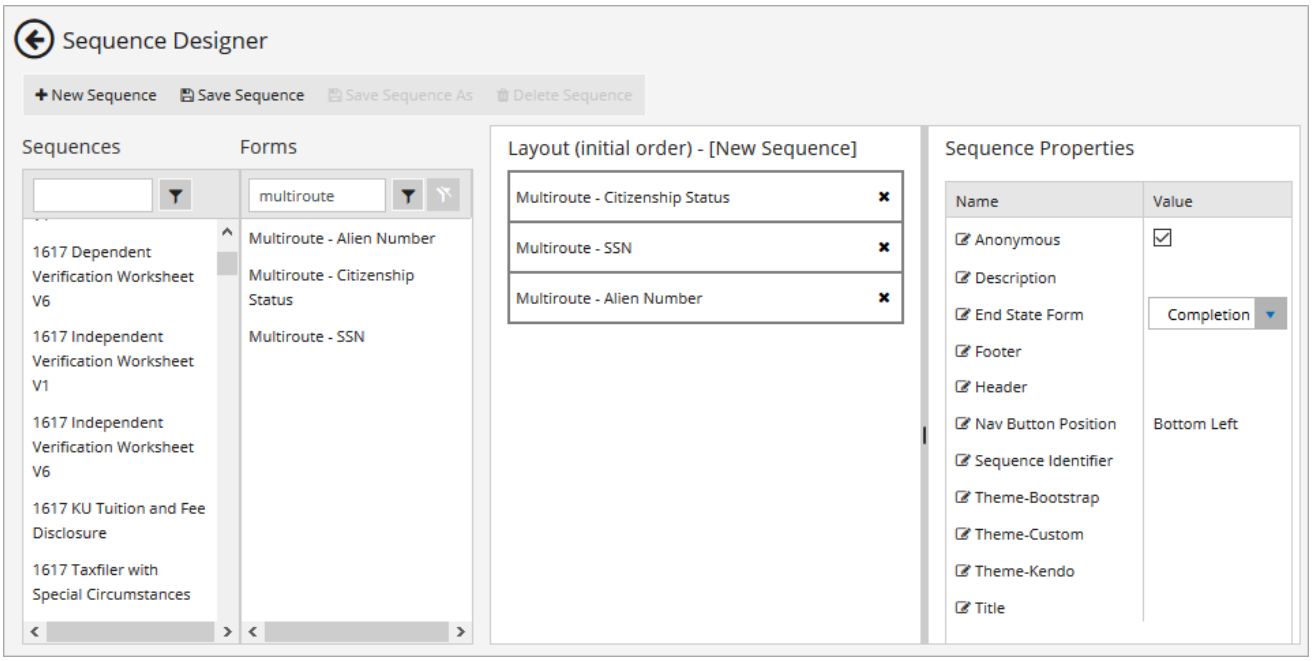
When all forms for all possible routes of a multi route sequence have been created, continue with Step B.

Step 2: Build the sequence

1. Browse to Sequence Designer and drag the main form and the sub forms to the Layout pane.

   The order of the sub forms in the Layout pane of the Sequence does not matter as long as they are grouped together after the main form is added. The branching/flow of the conditional sub forms will be specified when editing the workflow definition after the initial sequence is saved.

   Please note, however, that while workflow determines the order of the forms, when the initial workflow is created, the order of the forms in the sequence is used to create the order of states in the state machine workflow. It is easier to sort the sub forms in Sequence Designer than to change the order of states in the workflow after the sequence is created.



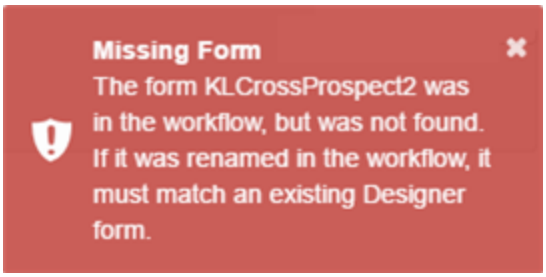
2. Save the sequence with all required forms.
3. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.

Step 3: Define transitions in Workflow Composer

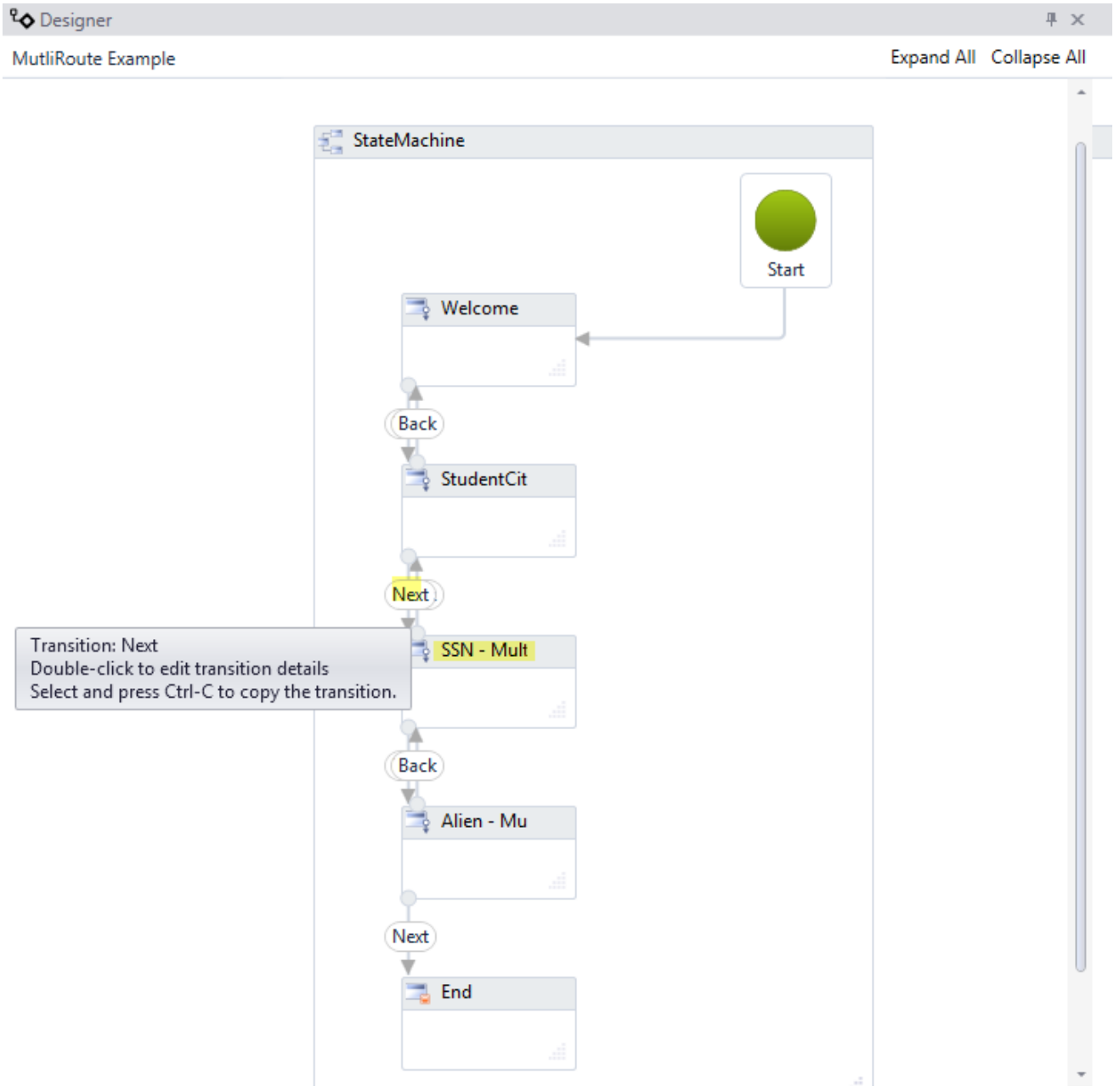
1. In Workflow Composer double-click on **Next** transition from the main form to form A.

   You can move the icons for states and transitions within the StateMachine sequence to get a clearer view of the links between the items.

   When editing a workflow definition, keep in mind that a state in the state machine workflow equates to a form within the sequence. The name of a **State** must match the name of a **Form** to be rendered properly. If Renderer encounters a State in workflow definition that does not match name of any Form created in Form Designer, an error similar to the following will be generated.

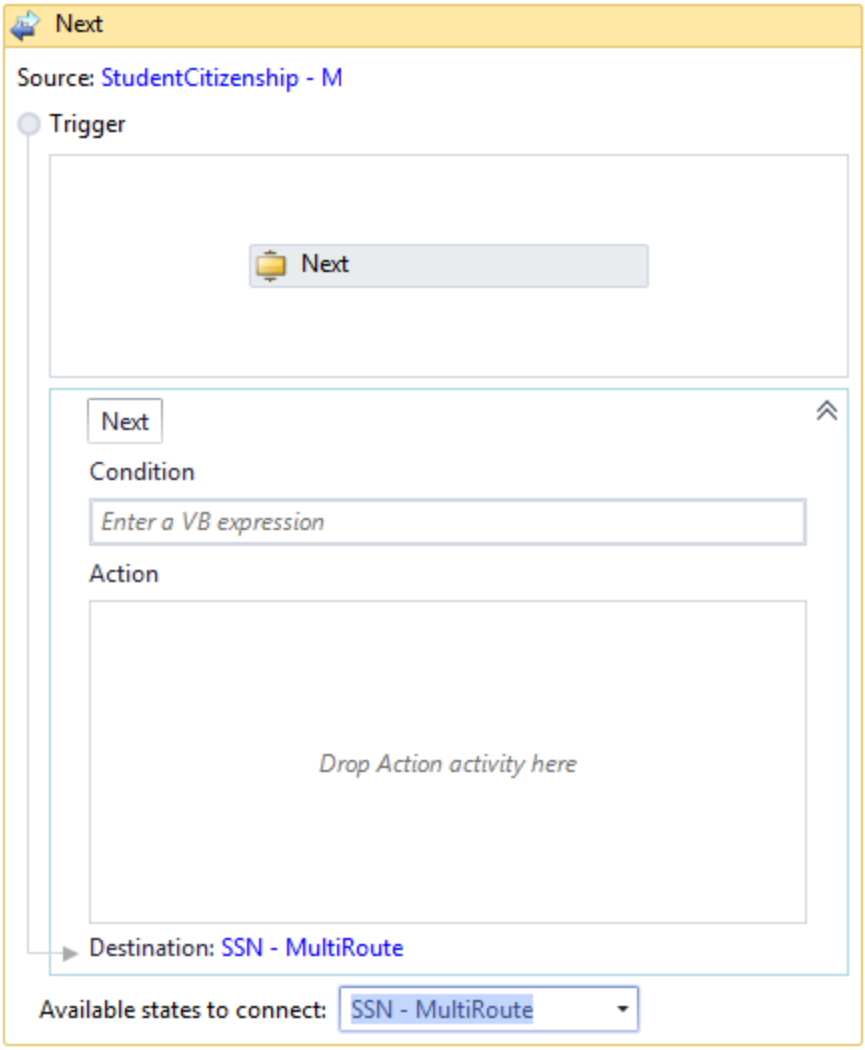


In our example the main form is named StudentCitizenship - MultiRoute.



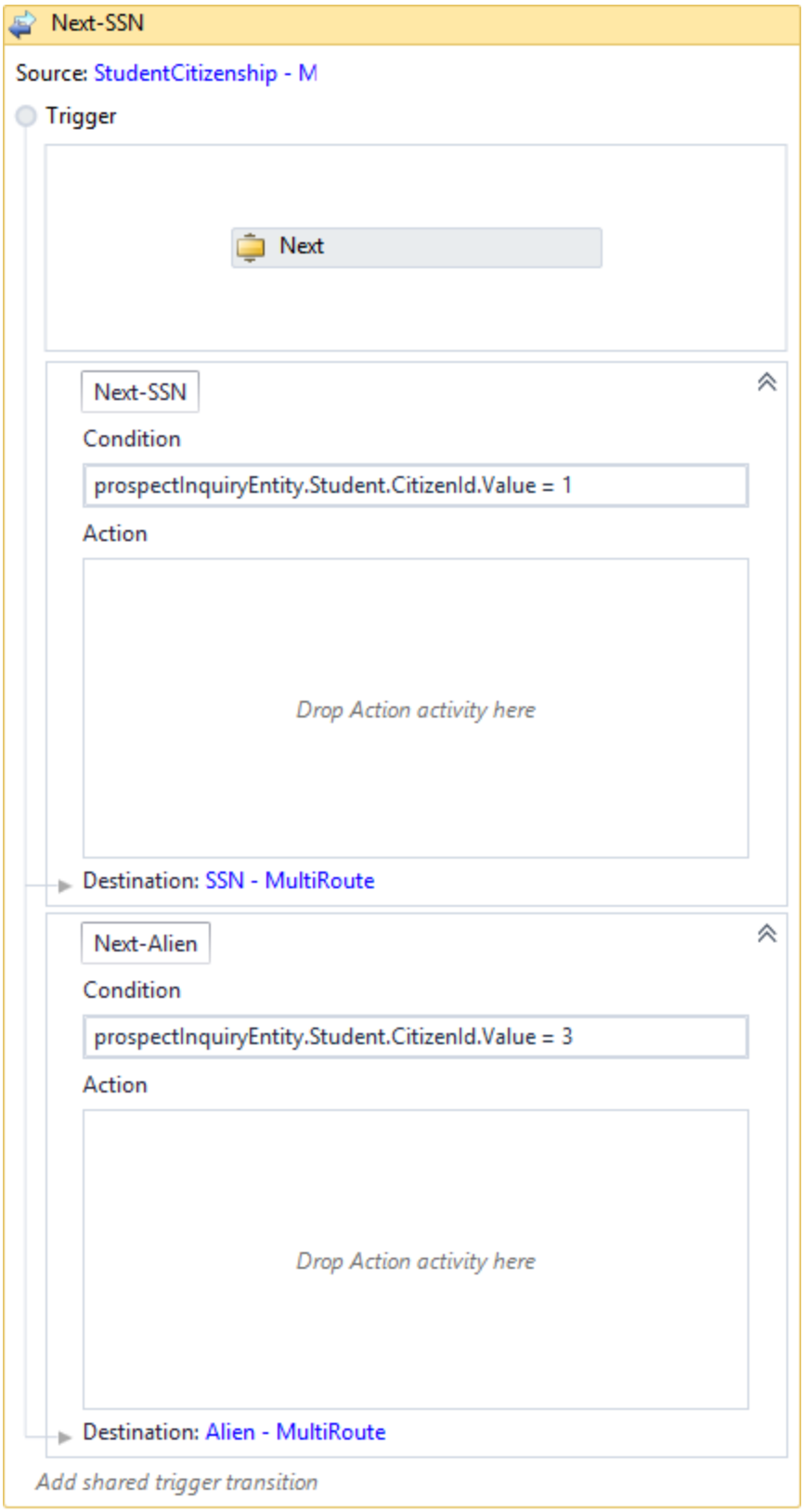
The transition is expanded and displays Trigger, Condition, and Action fields.

2. In the **Condition** field for the Next-SSN transition, enter the decision input property derived from the main form. In our example the condition is the US Citizen value selected in the Citizenship Status field, i.e., *prospectInquiryEntity.Student.CitizenId.Value = 1*.
3. At bottom of transition, select **Add Shared Trigger Transition**" and select **form B**. The name of form B is inserted into the Destination field.

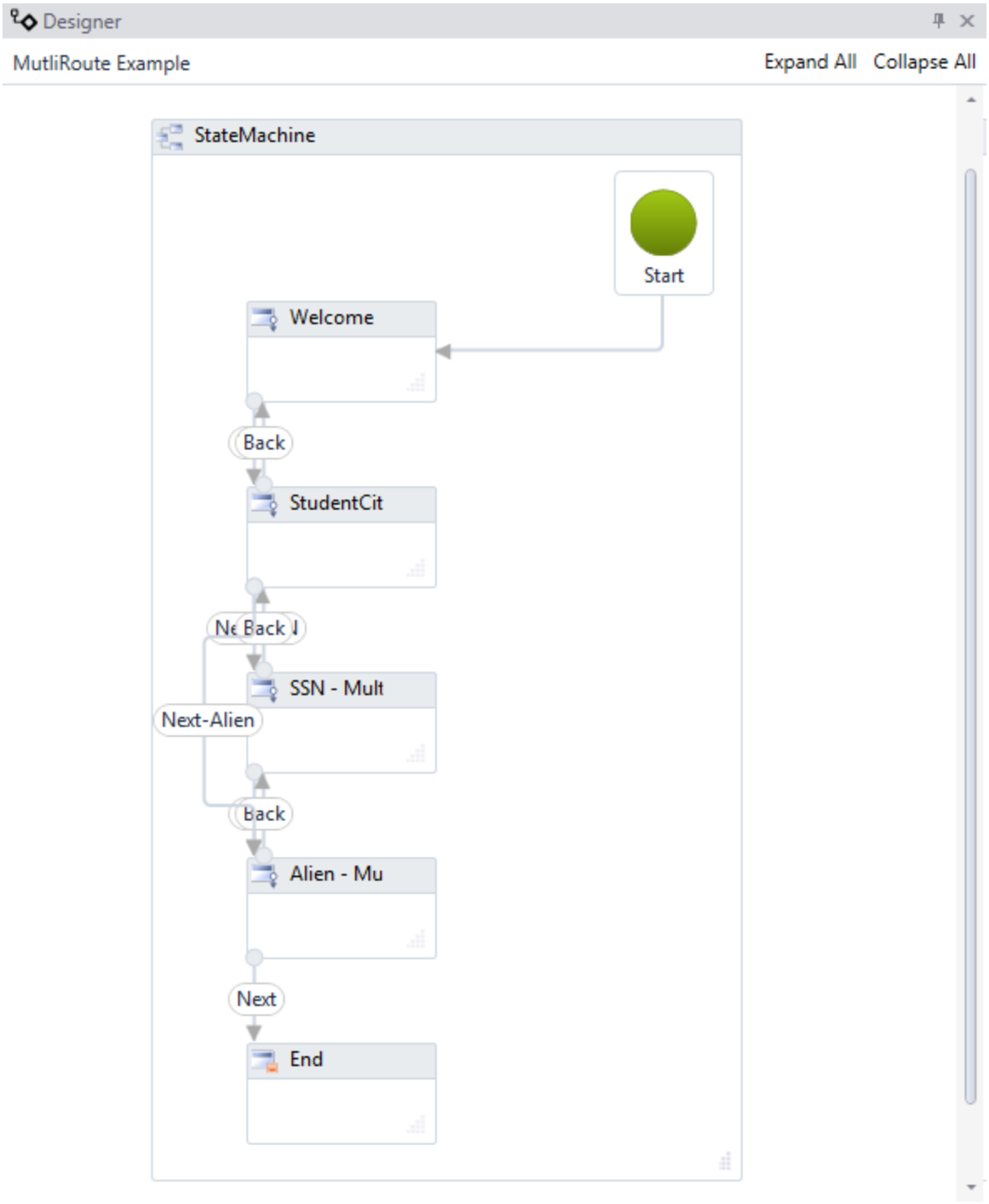


4. When created, change the default label on the transition from "T1" to "Next-Alien".
5. In the **Condition** field for the Next-Alien transition, enter the decision input property derived from the main form. In our example the condition is the Eligible Non-Citizen value selected in the Citizenship Status field, i.e., *prospectInquiryEntity.Student.CitizenId.Value = 3*.
6. Add a condition for each transition based on decision input property in the main form (i.e., prospectInquiryEntity.Student.CitizenId.Value = 1 and prospectInquiryEntity.Student.CitizenId.Value = 3).

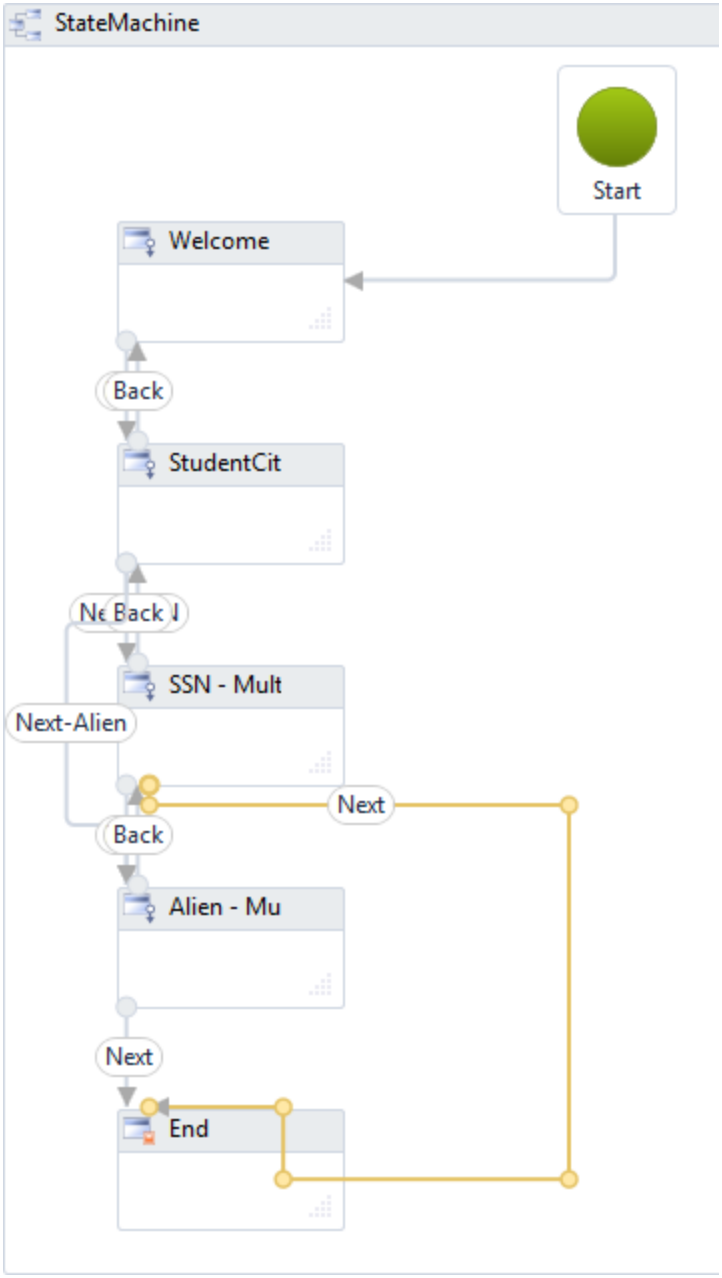
   At this point shared transition is created for the Next button on the main form. Based on the selected CitizenId.Value value, the Next button will route the user to form A or form B.


Next-SSN
Source: StudentCitizenship - M
Trigger
Next
Next-SSN
Condition
prospectInquiryEntity.Student.CitizenId.Value = 1
Action
Drop Action activity here
Destination: SSN - MultiRoute
Next-Alien
Condition
prospectInquiryEntity.Student.CitizenId.Value = 3
Action
Drop Action activity here
Destination: Alien - MultiRoute
Add shared trigger transition

7. In the breadcrumbs at top of Designer window, select your **sequence name** (leftmost item). The Designer displays the entire StateMachine workflow view.

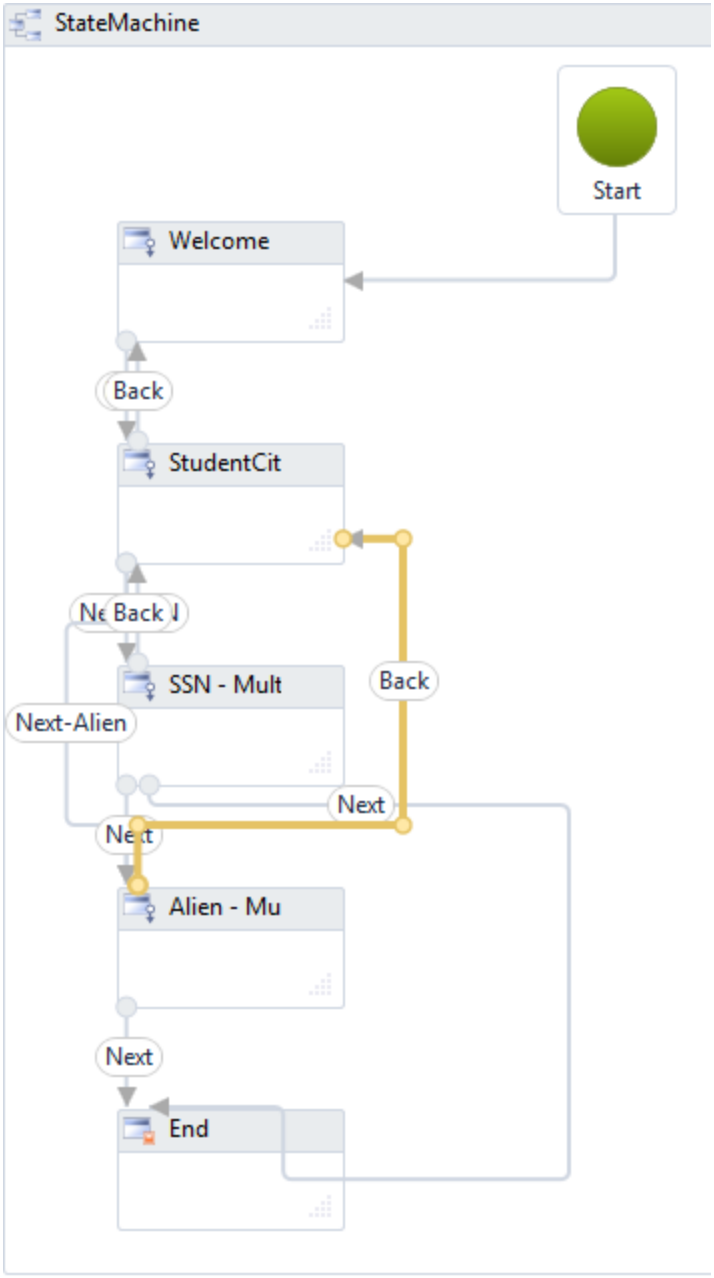


8. Select the **Next transition** on form A to form B and move the endpoint to the **End state**.



The Next transition on form A now points to the End state instead of form B.

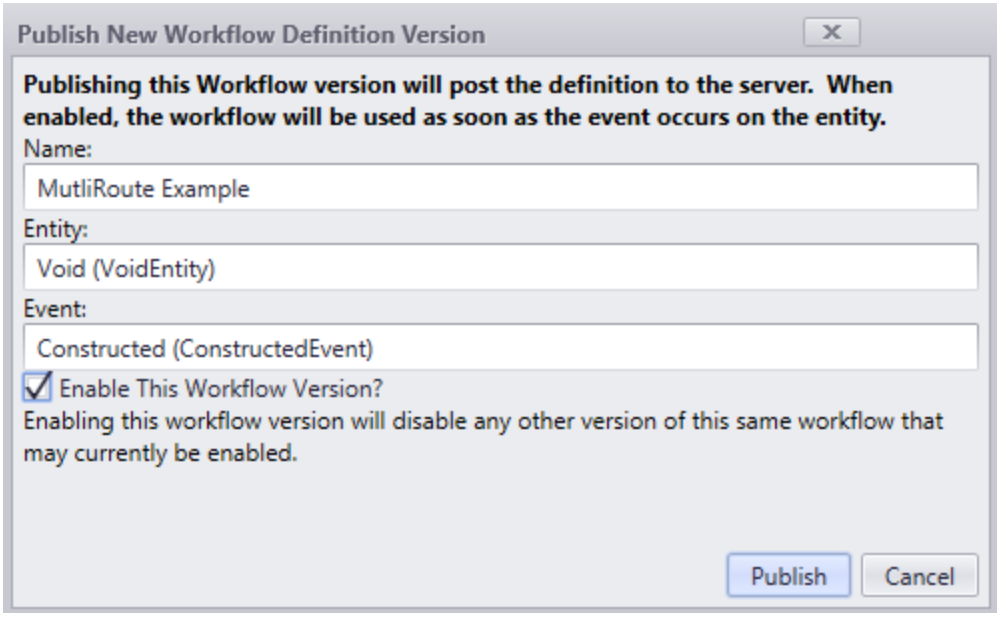
9. Select the **Back** transition on form B to form A and move the endpoint to the **main** form.



The Back transition on form B now points to the main form instead of form A.

If it is necessary to add new transitions, be sure to drag/drop the WaitForFormBookmark activity to the Trigger section of the transition.

10. **Publish** the updated workflow for the multi route form sequence. Be sure to select the **Enable This Workflow** option.



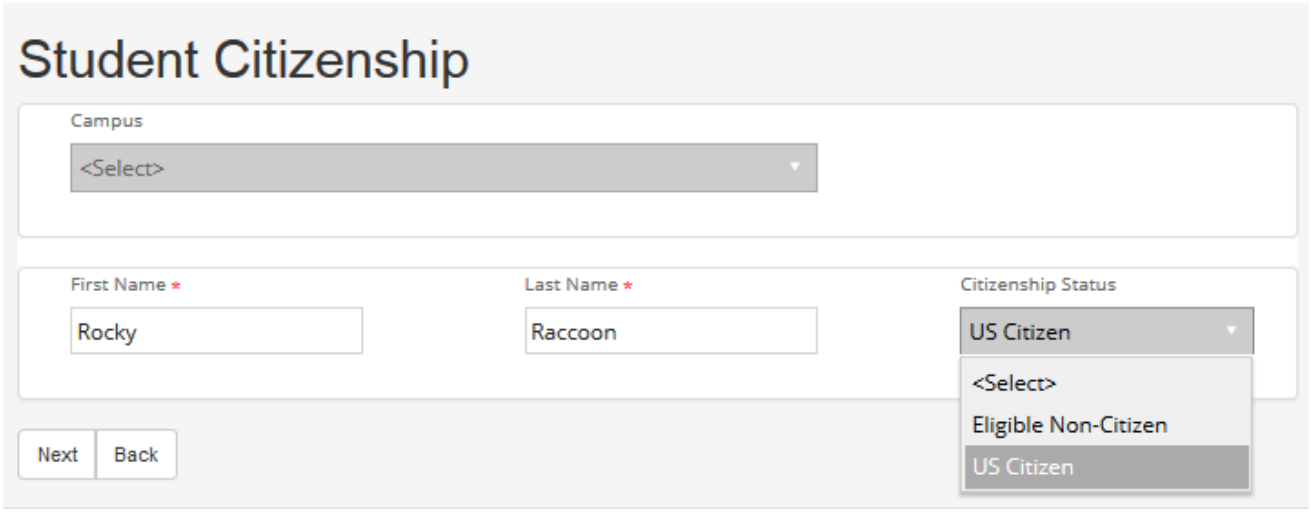
11. Select **OK** on the *Publish Succeeded* message.

    The updated version of the sequence workflow is now enabled in your test environment.

Step 4: Render and test the sequence

1. Browse to the Forms Renderer site and select the sequence just created.

   The main form is displayed.



2. Fill out the form and select the value that takes the user to form B and select **Next**.

   Form B is displayed.

3. Select the **Back** button on form B.

The main form is displayed.

4. Modify the value on the main form and select the value that takes the user to form A and select **Next**.

   Form A is displayed.

5. Fill out the form and select **Next**.

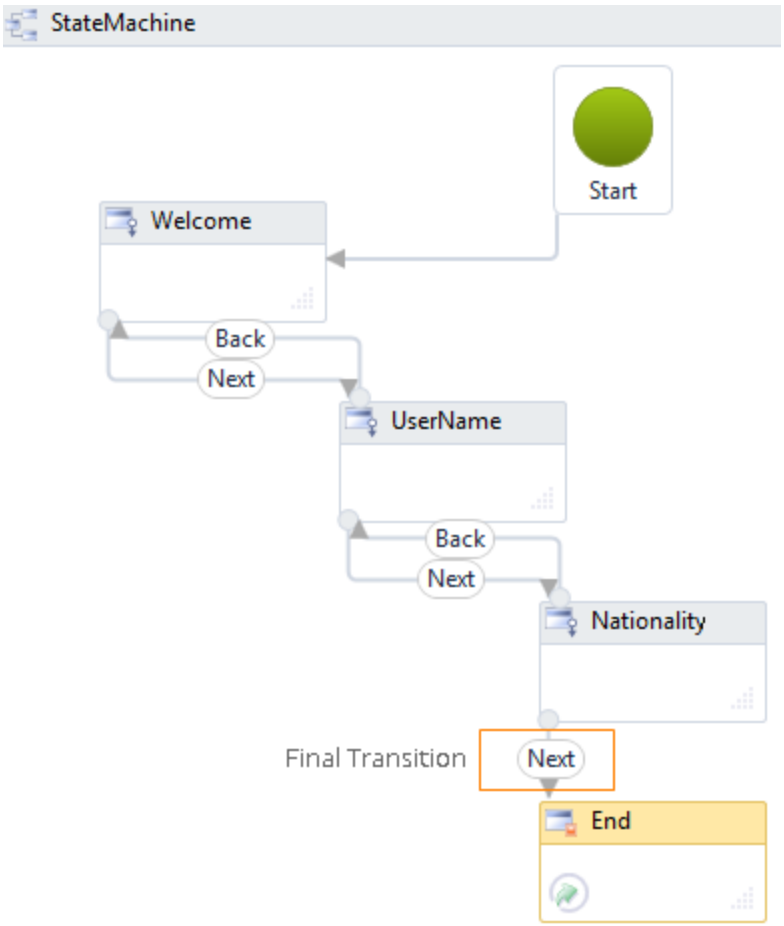
   The final form/end state is displayed with no buttons.

Passing Values to an End State Form

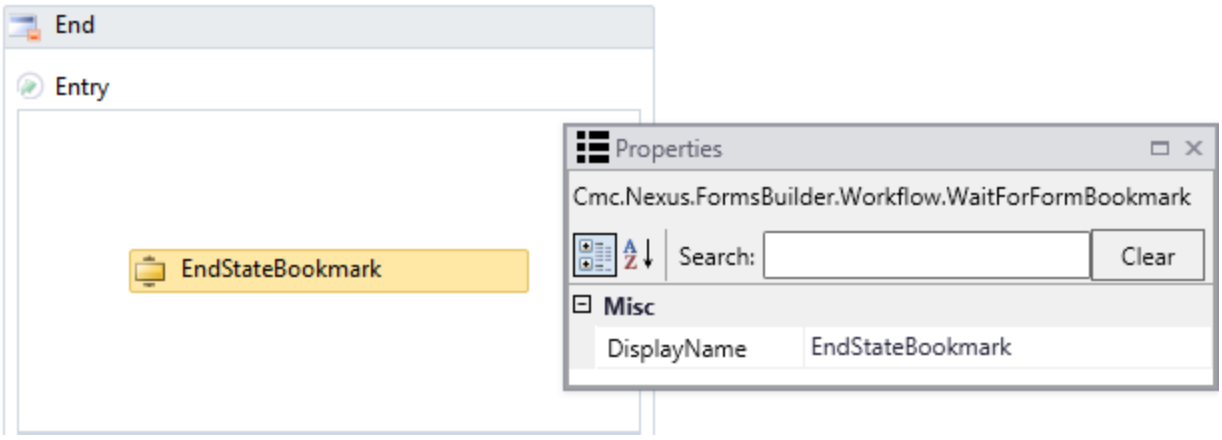
A customized end state form or confirmation form can display values that are assigned on previous forms within a sequence; however, any values that are assigned in the final transition before the end state form are not passed to the end state form. If you need values from the last transition to be available as part of the model and passed to the end state form, the Entry section of the final State in the workflow must contain a WaitForFormBookmark activity.

Example

The sequence contains the following forms: Welcome, UserName, Nationality, and End. The user enters a user name and selects a nationality value. The user name and nationality value are displayed on the End form.



An Assign activity in the final transition assigns a value to the nationality argument. This argument is bound to `vm.models.nationality`. To pass this assignment to the End form, a WaitForFormBookmark activity is inserted into the Entry section of the End form.

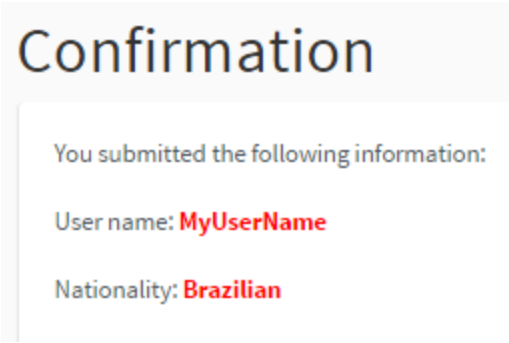


The customized End State form contains an HTML component with the following HTML code:

 You submitted the following information:

 User name: {{vm.models.userName}}

 Nationality: {{vm.models.nationality}}



Without the WaitForFormBookmark activity in the Entry section of the End form, the value of the nationality argument would not be available for display in the End form.

State Machine Workflows

When you save a form sequence, Forms Builder automatically creates a new state machine workflow definition. The new workflow definition contains a state for each form included in the sequence. The order in which the states appear in the workflow definition match the order in which the forms were dropped onto the Layout pane when creating the sequence in Sequence Designer.

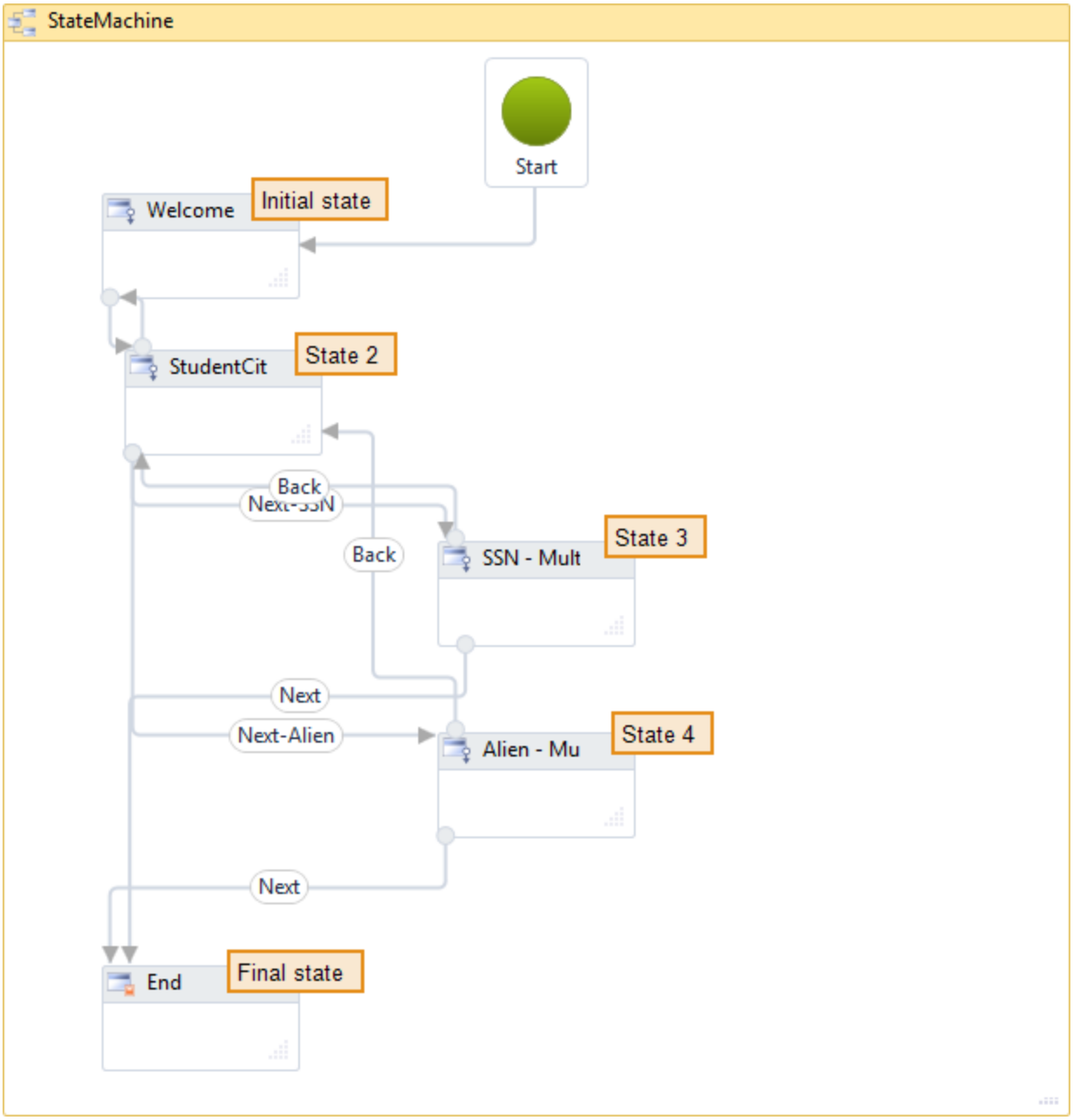
You can access the workflow using Workflow Composer For more information, refer to Open the Workflow for a Sequence. You can then edit, save, and publish the workflow using Workflow Composer.

State machine workflows provide a modeling style with which you can model your workflow in an event-driven manner. A StateMachine activity contains the states and transitions that make up the logic of the state machine and can be used anywhere an activity can be used. To create a state machine workflow, *States* are added to a StateMachine activity, and *Transitions* are used to control the flow between states.

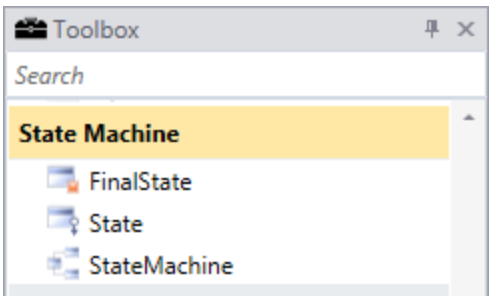
State

A state machine workflow must have one and only one initial state and at least one final state. Each state that is not a final state must have at least one transition.

The following image shows a state machine workflow with five states and multiple transitions. The **initial state** named *Welcome* represents the first state in the workflow. This is designated by the line leading to it from the **Start** node. The **final state** named *End* represents the point at which the workflow is completed.



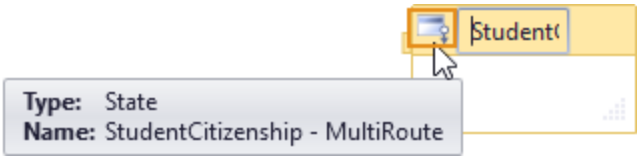
- To add a state to a workflow, drag the **State** activity from the State Machine section of the Toolbox and drop it onto a state machine workflow in the Designer pane.



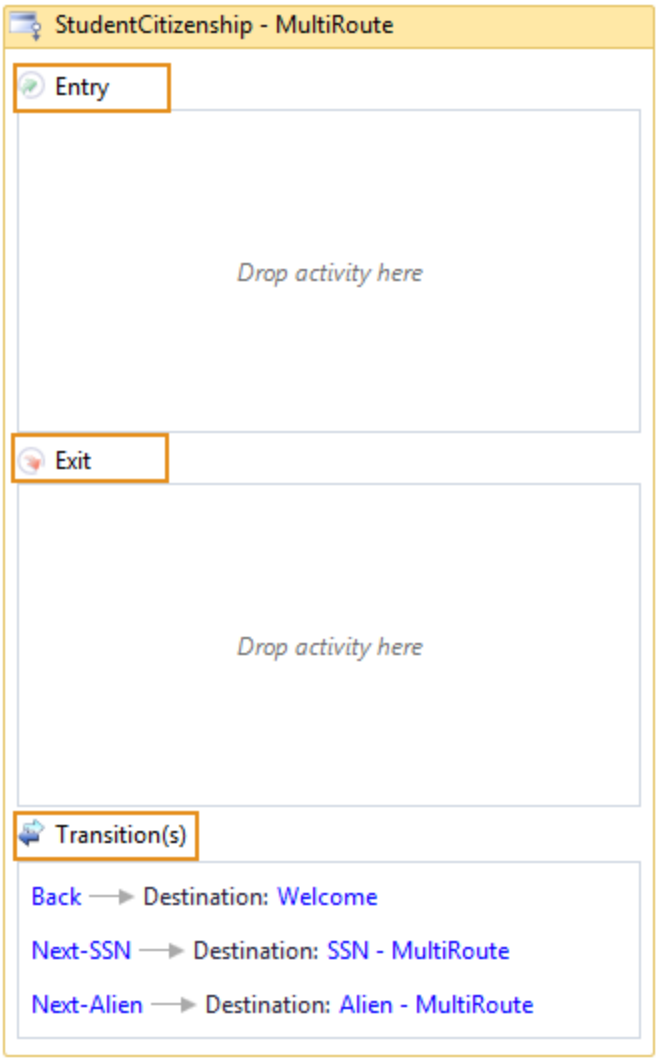
- To configure a state as the Initial State, right-click the State in the Designer pane and select **Set as Initial State**. If there is no current initial state, the initial state can be designated by dragging a line from the Start node to the state.
- To add a final state to a workflow, drag a **FinalState** activity from the State Machine section of the Toolbox and drop it onto a state machine workflow in the Designer pane. A final state is a state that has its *IsFinal* property set to true, has no Exit activity, and no transitions originating from it.

A state can have an **Entry** and an **Exit** action. (A state configured as a final state only has an entry action). When a workflow instance enters a state, any activities in the Entry action execute. When the Entry action is complete, the triggers for the state's **Transitions** are checked. When a transition to another state is confirmed, the activities in the Exit action are executed. After the Exit action completes, the activities in the transition's action execute, and then the new state is transitioned to, and its Entry actions are executed.

Double-click the icon of a State activity to view its details.



The following image shows the details of a State activity.



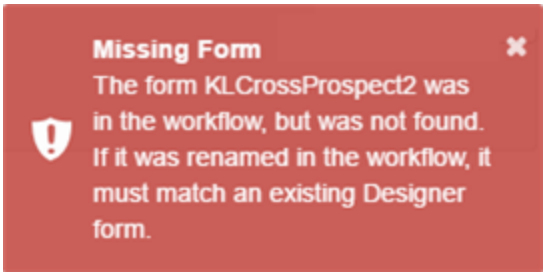
Notes

- When debugging a state machine workflow, breakpoints can be placed on the root state machine activity and states within the state machine workflow. Breakpoints may not be placed directly on the transitions, but they may be placed on any activities contained within the states and transitions.

  When editing a workflow definition, keep in mind that a state in the state machine workflow equates to a form within the sequence. The name of a **State** must match the name of a **Form** to be rendered properly. If Renderer encounters a State in workflow definition that does not match name of any Form created

-

in Form Designer, an error similar to the following will be generated.

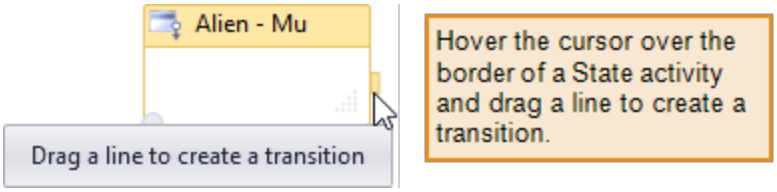


Transition

All states must have at least one transition, except for a final state which may not have any transitions. Transitions may be added after a state is added to a state machine workflow, or they can be created as the State activity is dropped into the Designer pane.

To add a state and create a transition in one step, drag a **State** activity from the State Machine section of the Toolbox and hover it over another State in the Designer pane. When the dragged State is over another State, four triangles will appear around the other State. If the State is dropped onto one of the four triangles, it is added to the state machine and a transition is created from the source State to the destination State.

To create a transition after a state is added, drag a line from one state to another state. The yellow box indicates the start point of the line.



To edit the transition details, double-click the transition line.

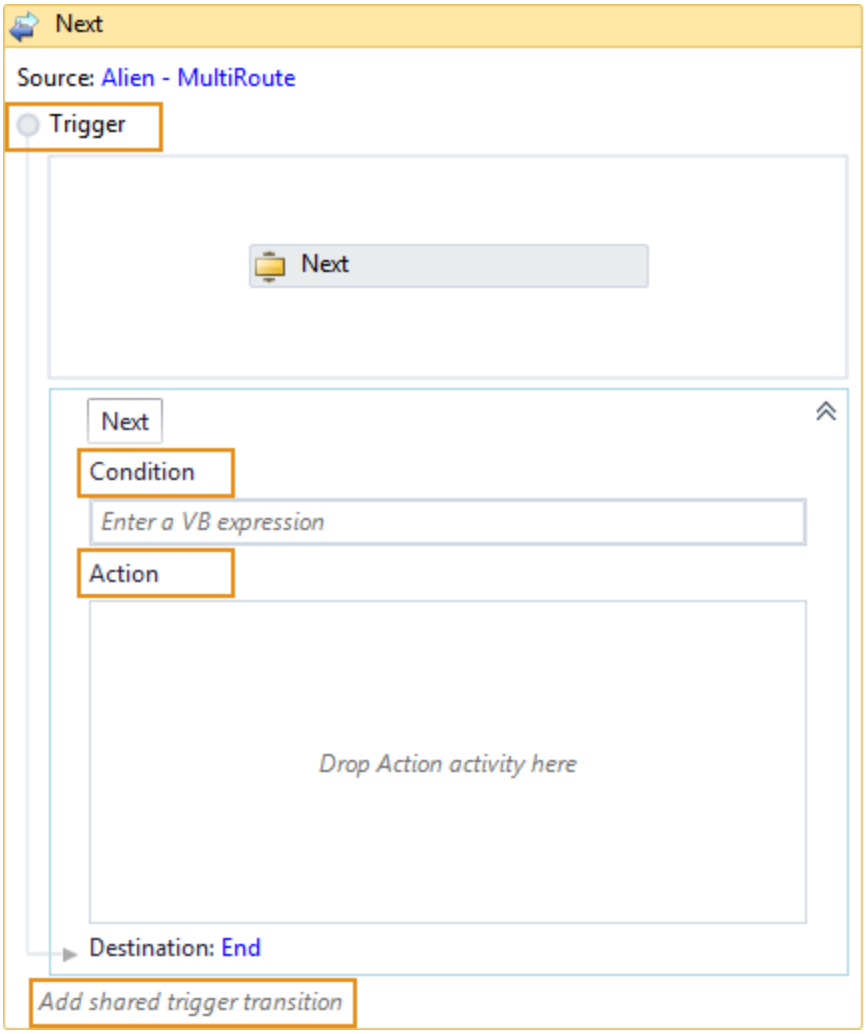
A transition may have a **Trigger**, a **Condition**, and an **Action**.

A transition's Trigger is checked when the transition's source state's Entry action is complete. Typically, the Trigger is an activity that waits for some type of event to occur, but it can be any activity, or no activity at all. Once the Trigger activity is complete, the Condition if present, is evaluated.

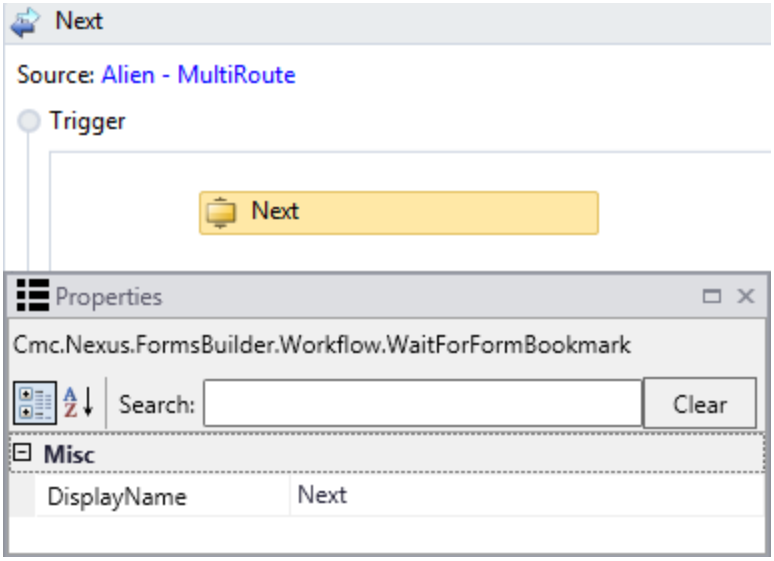
- If there is no Trigger activity, then the Condition is immediately evaluated.
- If the Condition evaluates to false, the transition is cancelled, and the Trigger activity for all transitions from the state are rescheduled.
- If there are other transitions that share the same source state as the current transition, those Trigger actions are cancelled and rescheduled as well.

- If the Condition evaluates to true, or there is no condition, then the Exit action of the source state is executed, and then the Action of the transition is executed. When the Action completes, control passes to the Target state.

The following image shows the transition designer for a transition with a *Next* trigger that was automatically inserted by Forms Builder when the state machine workflow was created. Forms Builder will also insert *Back* triggers to enable users to navigate forward and back through a sequence.



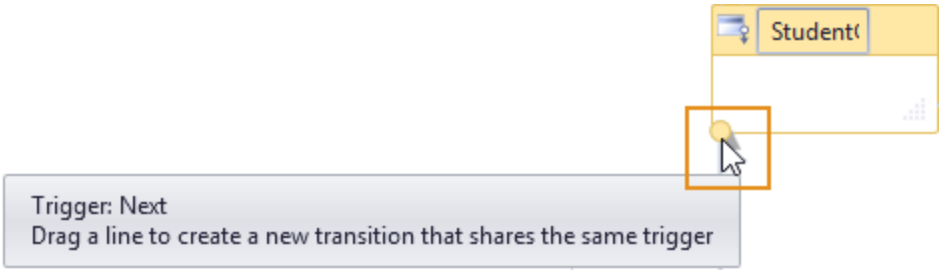
Note that the *Next* and *Back* triggers created by Forms Builder are **WaitForFormBookmark** activities. To add this trigger to a workflow, drag a WaitForFormBookmark activity from the Toolbox to the Designer pane. When you modify an existing workflow to add new transitions, be sure to add a WaitForFormBookmark activity as a trigger.



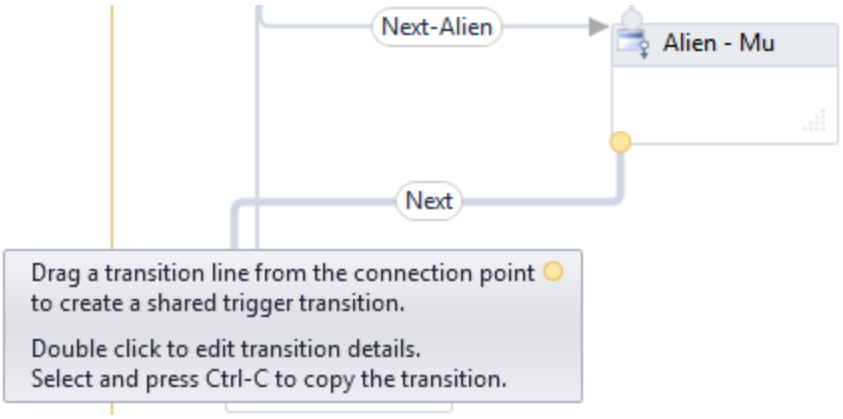
Shared Trigger Transition

Transitions that share a common trigger are known as shared trigger transitions. Each transition in a group of shared trigger transitions has the same trigger, but a unique Condition and Action. An example for shared trigger transitions is a form with multiple value selection options which routes to different sub forms depending on a value selection (refer to Multi Route Forms).

To create a share trigger transaction, drag a line from the connection point (yellow circle) of the trigger.



To add additional actions to a transition and create a shared transition, select the yellow circle that indicates the start of the desired transition and drag it to the desired state. The new transition will share a same trigger as the initial transition, but it will have a unique condition and action. Shared transitions can also be created from within the transition designer by selecting **Add shared trigger** transition at the bottom of the transition designer (refer to transition designer image above), and then selecting the desired target state from the **Available states to connect** drop-down.



If the Condition of a transition evaluates to false (or all of the conditions of a shared trigger transition evaluate to false), the transition will not occur and all triggers for all the transitions from the state will be rescheduled.

State Machine Terminology

Term	Description
State	The basic unit that composes a state machine. A state machine can be in one state at any particular time.
Entry Action	An activity which is executed when entering the state.
Exit Action	An activity which is executed when exiting the state.
Transition	A directed relationship between two states which represents the complete response of a state machine to an occurrence of an event of a particular type.
Shared Transition	A transition that shares a source state and trigger with one or more transitions but has a unique condition and action.
Trigger	A triggering activity that causes a transition to occur.
Condition	A constraint which must evaluate to true after the trigger occurs in order for the transition to complete.
Transition Action	An activity which is executed when performing a certain transition.
Conditional Transition	A transition with an explicit condition.
Self-transition	A transition which transits from a state to itself.
Initial State	A state which represents the starting point of the state machine.
Final State	A state which represents the completion of the state machine.

Update a Form After Creation of a Sequence

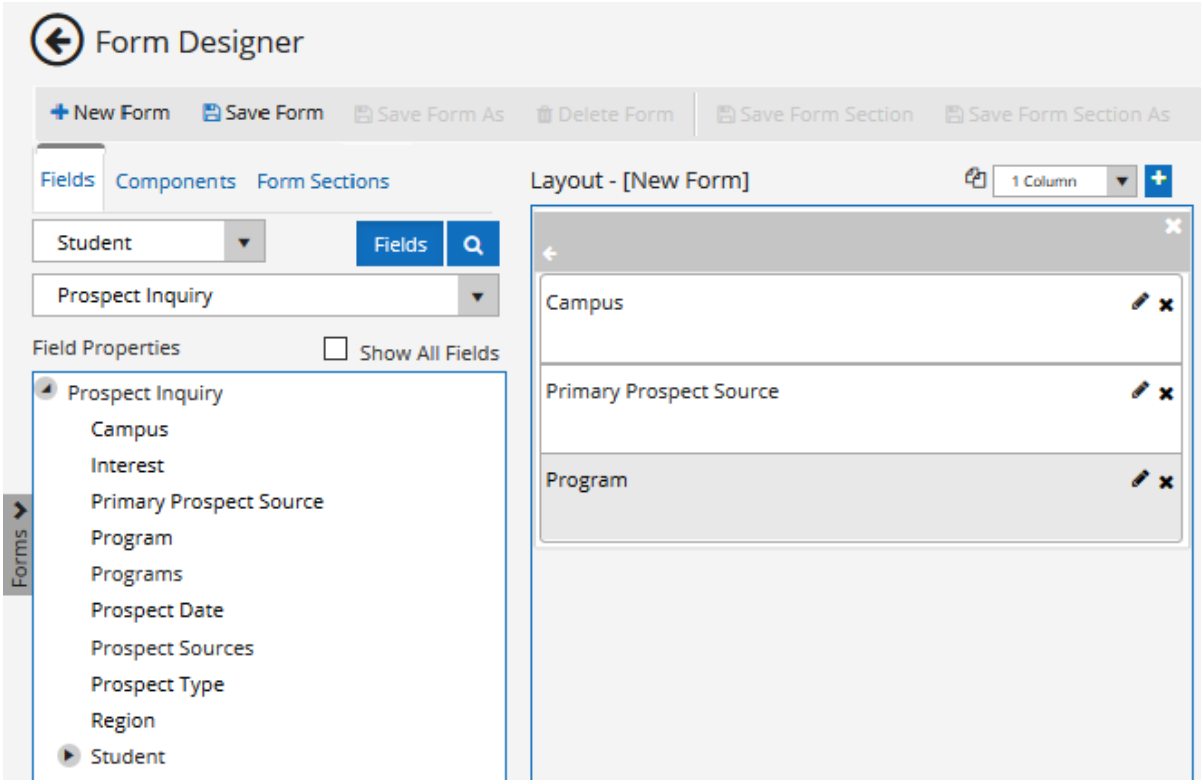
When a new sequence is saved, and the workflow definition is initially created, arguments for all entities referenced within all forms in the sequence will be automatically added in the workflow definition.

However, when a property associated with an entity is added to a form, and the sequence initially did not have any properties for that entity, arguments for that entity must be manually added to the workflow definition.

When a form is modified (fields added or deleted) and saved, the changes will be reflected in existing sequences that contain that form.

Adding an Entity to a Workflow

1. In Form Designer, create a form with several fields from the Prospect Inquiry entity and save the form.

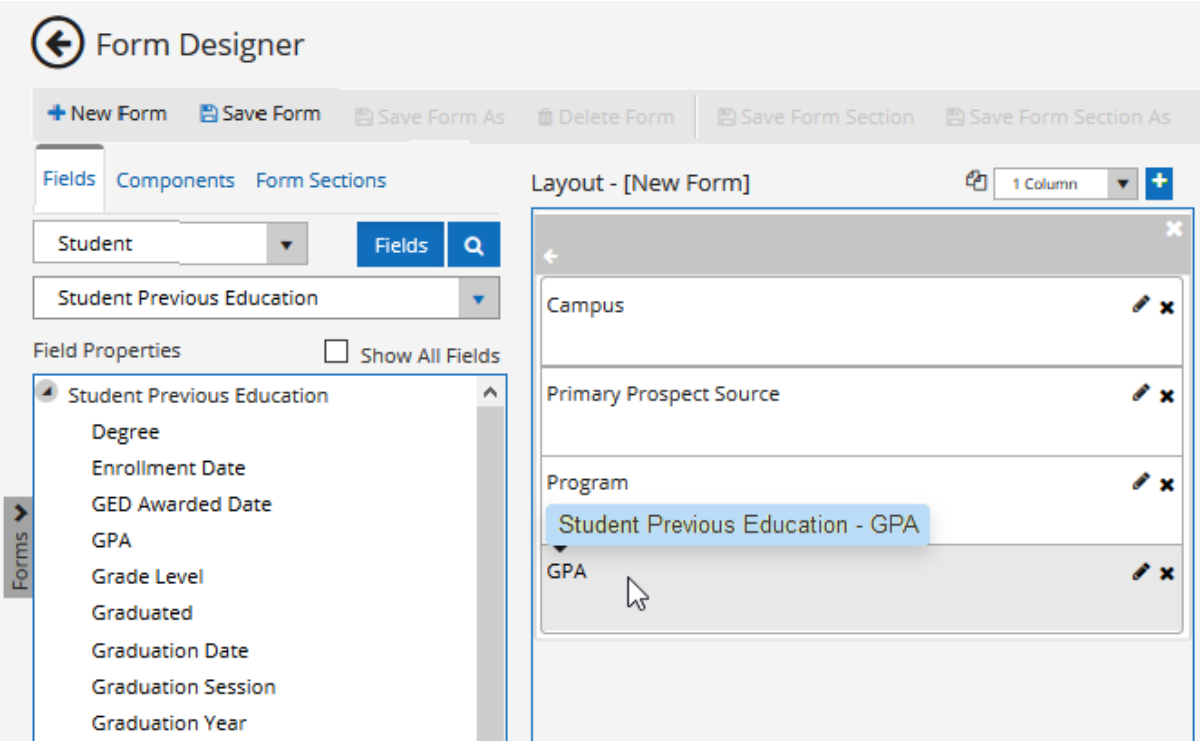


2. In Sequence Designer, create a sequence using the form and save the sequence.
3. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.

   Note that an argument was created for the Prospect Inquiry entity.

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event	In/Out	ConstructedEvent	*Default value not supported*
prospectInquiryEntity	In/Out	ProspectInquiryEnt	*Default value not supported*
Create Argument			

4. In Form Designer, add a field from another entity, e.g., Student Previous Education, and save the form.



5. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence. Note that no argument was created for the Pending Applicant Previous Education entity.

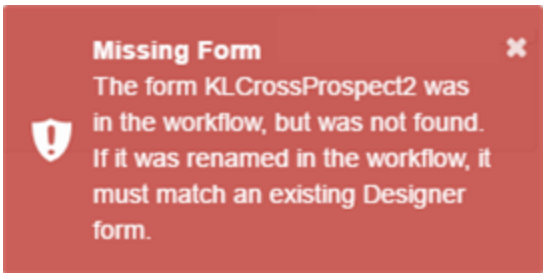
6. In Workflow Composer, create an argument for entity that was added to the form by specifying the Name, Direction (In/Out), and Argument type.

   The casing of argument names is significant. Use camel case as shown in the examples below.

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event	In/Out	ConstructedEvent	*Default value not supported*
prospectInquiryEntity	In/Out	ProspectInquiryEntity	*Default value not supported*
prospectInquiryProgramEntity	In/Out	ProspectInquiryProgramEntity	*Default value not supported*
studentPreviousEducationEntityGpa	In/Out	StudentPreviousEducation	*Default value not supported*

7. Save the workflow.

When editing a workflow definition, keep in mind that a state in the state machine workflow equates to a form within the sequence. The name of a **State** must match the name of a **Form** to be rendered properly. If Renderer encounters a State in workflow definition that does not match name of any Form created in Form Designer, an error similar to the following will be generated.



Workflows for CampusNexus CRM

Forms Builder can be set up to use the Anthology Student database, the CampusNexus CRM database, or both. When both databases are used, Form Designer displays the Service Provider option to enable you to select a database and the associated entities. When only one of the two databases is used, the entities available for the selection of form fields will correspond to the database, and the Service Provider option is not displayed.

When CampusNexus CRM is the Service Provider, you can build forms that use CampusNexus CRM entities and design workflows for the associated form sequences. These workflows rely on specific events and objects that are generated in CampusNexus CRM.

CampusNexus CRM Events and Objects

All operational and reference objects in CampusNexus CRM are wrapped in the assembly file Cmc.NexusCrm.Contracts.dll. Whenever new properties are created in CampusNexus CRM or an existing property definition (metadata) is changed, this assembly is regenerated. Workflows for CampusNexus CRM require the events and objects contained in the Cmc.NexusCrm.Contracts.dll to be available in Workflow Composer and Forms Renderer.

To regenerate the assembly after any metadata changes, perform the following steps:

1. On the IIS Server of the Web Client for CampusNexus CRM, **restart** the **Cmc.Crm.Workspaces** application.
2. Navigate to the URL of the Web Client for CampusNexus CRM.
3. Copy the regenerated **Cmc.NexusCrm.Contracts.dll** from the \bin folder of the Web Client to the installation path of Workflow Composer and to the \bin folder of Forms Renderer.

As a best practice, when CampusNexus CRM metadata is changed, the generated contracts assembly file (Cmc.NexusCrm.Contracts.dll) must be copied from the bin folder of Web Client to the installation path of **Workflow Composer** and to the \bin folder of **Forms Renderer**.

> ⚠ **Do not copy the Cmc.NexusCrm.Contracts.dll to the \bin folder of Forms Builder Designer.**

If an existing workflow includes a property that is not available in the current generated contracts, the administrator needs to manually edit the workflow and remove the property.

The .dll file can be copied while you remain logged on to Workflow Composer. Any updates will be reflected in Workflow Composer after you log off and on again.

Workflow Activities for CampusNexus CRM

The **Cmc.NexusCrm.Common.Workflow** namespace provides activities designed specifically for CampusNexus CRM, e.g., GetAttachment, GetRelatedEntity, and LookupContact. These activities become available when the

Activities and Contracts (CRM) package is installed using the Package Manager within Workflow Composer.

In addition to the workflow activities from the Cmc.NexusCrm.Common.Workflow namespace, workflows for form sequences can use other activities, such as generic activities (Assign, If, StateMachine, etc.) and workflow activities designed for CampusNexus packages.

For detailed information about these activities, please refer to Workflow Help.

Renderer

Forms Builder Renderer constructs the web pages for the sequences that were designed using Designer.

Renderer is installed on your web server under wwwroot\CMCFormsRenderer_V3.

`http://<server>.<domain>:<port>/#/Sequencelist`

Renderer is installed on port 9003 by default. The port number can be customized during installation. It can be installed on port 443 with HTTPS.

`https://<server>.<domain>:443/#/Sequencelist`

Access your Forms Builder URL and append **/#/Sequencelist** to view the Sequence List.

The Enable Sequence List option in Settings determines if the Sequence List is displayed or not. Change this setting to "false" if you do not want your users to view the Sequence List at this URL.

Azure AD Authentication

When Forms Builder is installed on premises, authentication of users is handled by the Security Token Service (STS) component.

- Users of sequences associated with the Student/Contact role are authenticated by the Student STS for Anthology Student or by the Contact STS for CampusNexus CRM.
- Users of sequences associated with the Staff role are authenticated by the Staff STS.

The authentication token returned by the STS identifies the user's role. The token enables the user to log in to Portal and access authenticated sequences as Student/Contact or Staff.

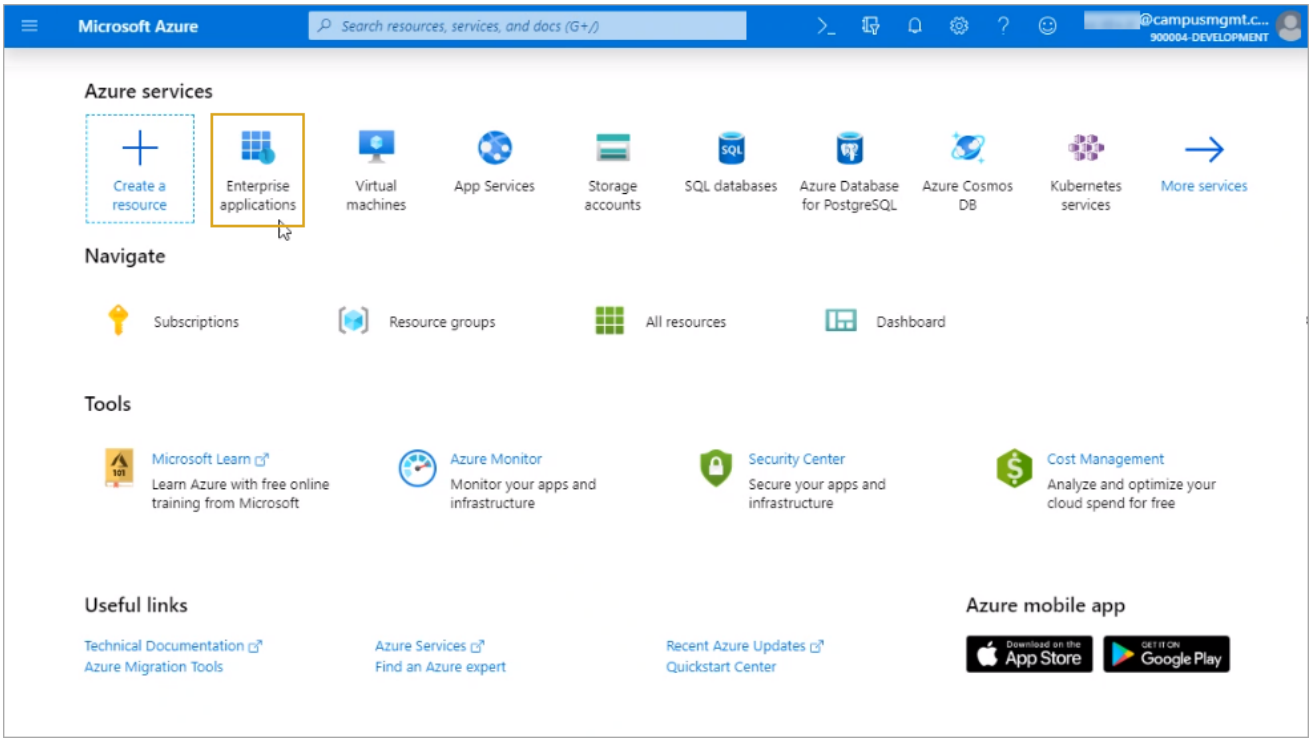
Forms Builder 3.4 and later can also be deployed in a cloud (Azure) environment with Active Directory (AD) authentication. In this environment, the STS component is not used. In an Azure AD environment, the authentication process does not allow for role verification.

The logic to verify the user's role must be embedded in **all** form sequences that are deployed in an Azure AD environment. Each workflow **must** include a LookupUser activity with UserType=Student or UserType=Staff as appropriate. The LookupUser activity with a proper UserType value ensures that student users cannot access staff sequences and vice versa. Refer to Workflow Examples below.

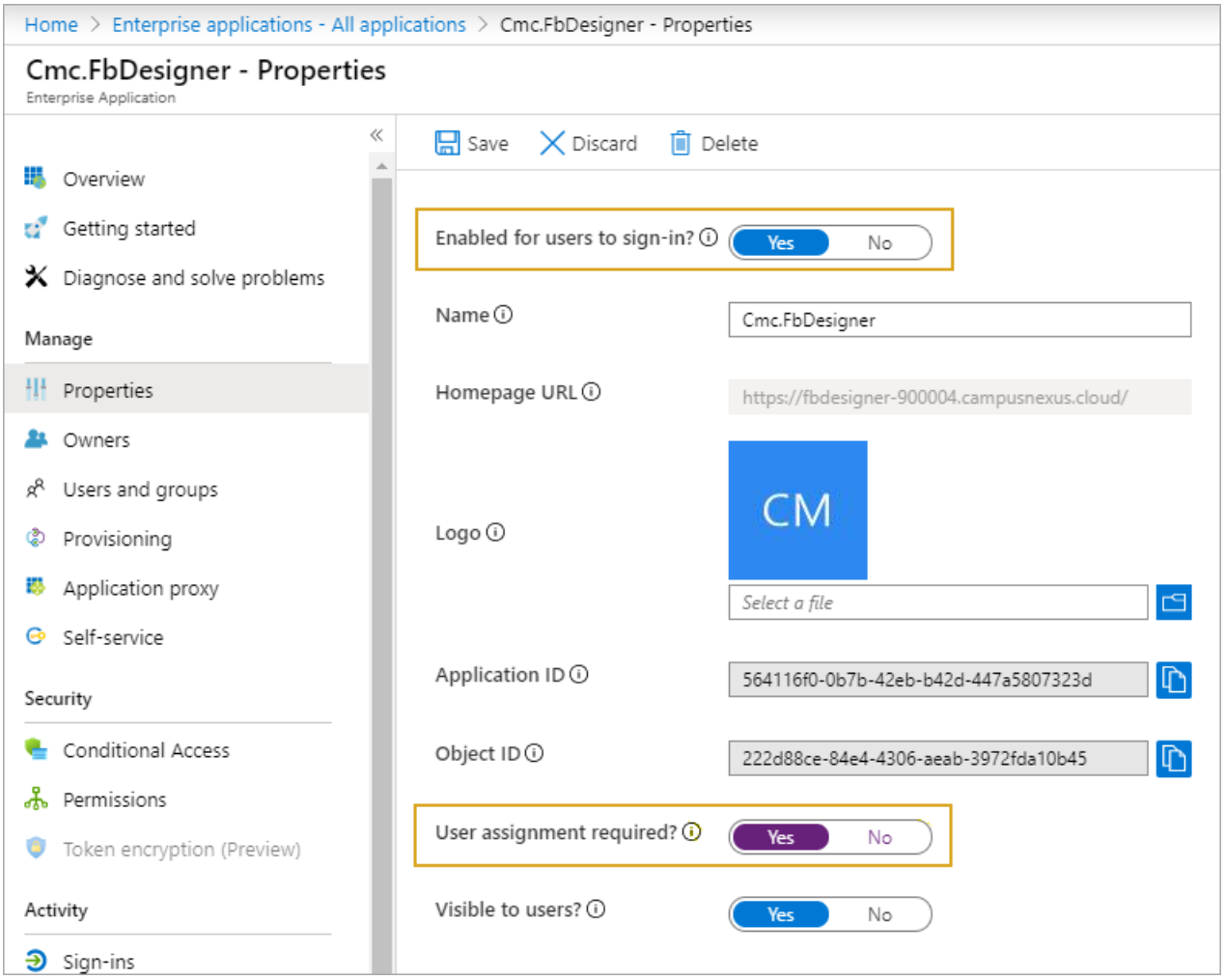
Prevent Student Users from Accessing Form Designer

When student and staff users share the same Azure AD group, you need to take the following steps to ensure that only staff members can access Form Designer.

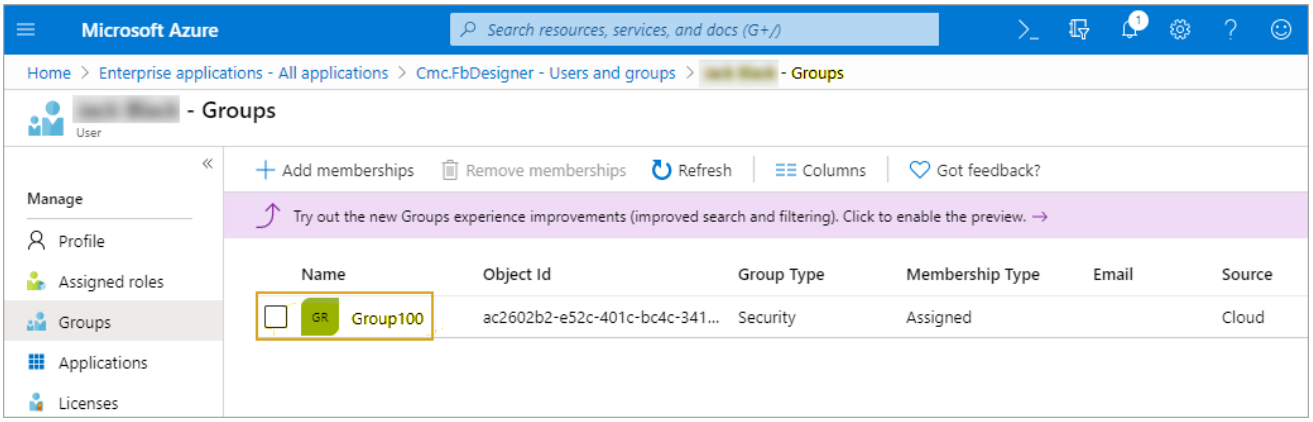
1. Log in to your Azure tenant directory as administrator.
2. Select **Home** > **Enterprise applications**.



3. Select **All applications** and select the link on **Cmc.FBDesigner**.
4. In the **Manage** menu, select **Properties**.
5. In the **Enabled for users to sign-in** property, select **Yes**.
6. In the **User assignment required** property, select **Yes**.



7. In the **Users and groups** menu, select **Add Users**.
8. In the **Add Assignment** form, select **Users and groups**.
9. In the **Users and groups** drop-down list, find the staff group to which you want to give permissions to access Form Designer and select **Select**.

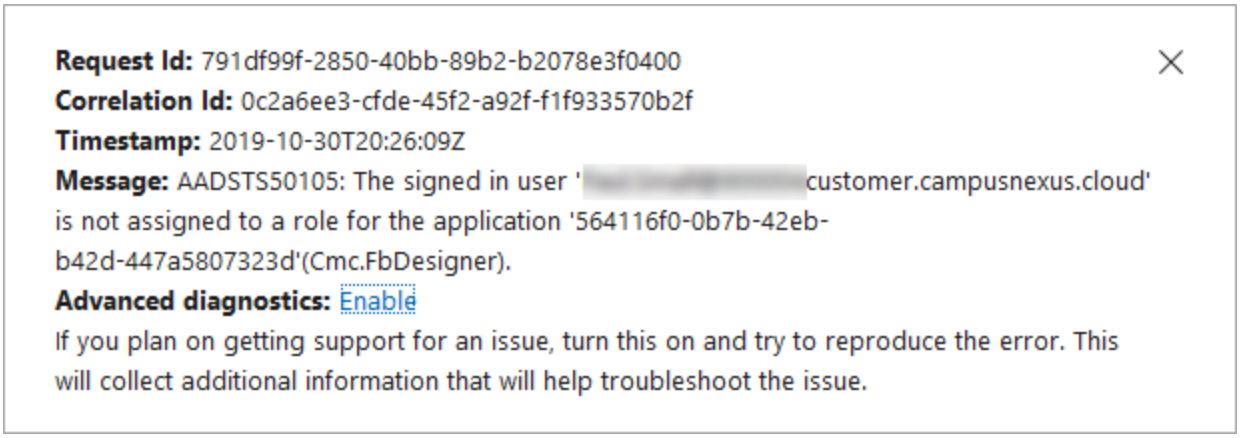


All Anthology Student staff members will be maintained within the selected group. When new staff members are added, they need to be added to that group.

Any member of the staff group can access Form Designer, can create and publish forms, but cannot modify workflows. Workflow Composer authentication and authorization are controlled separately. Refer to Workflow Composer in Installation Manager help.

Users that are not in the staff group, for example students users created via the CreatePortalAccount workflow activity, will receive a messages similar to the examples below when they try to access the URL for Form Designer





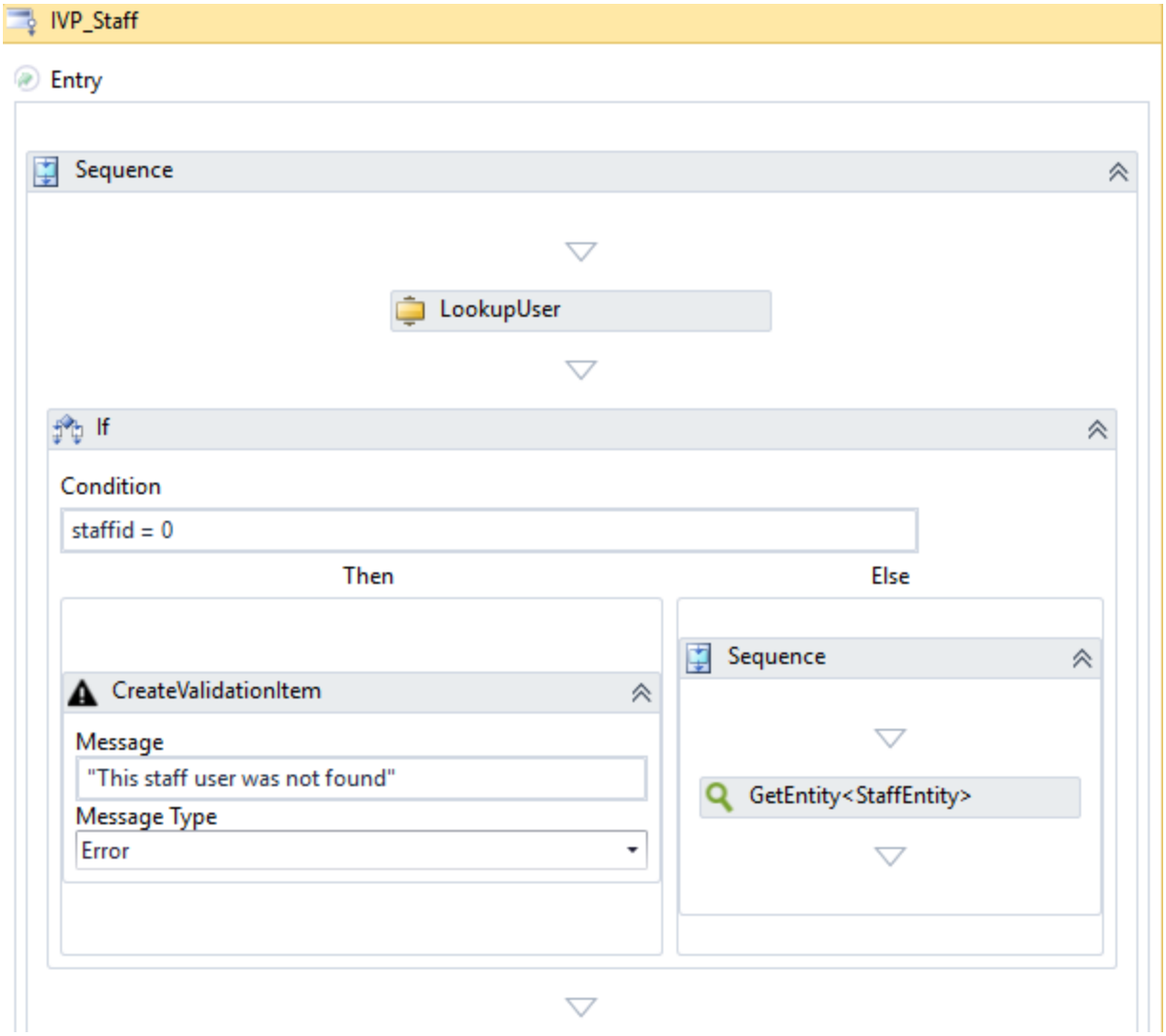
Workflow Examples

When sequences for Staff and Student roles are deployed in an Azure AD environment, the workflows need to include logic as detailed below. These workflow segments need to be placed in the first form of a sequence.

Verification of Staff Role in Azure AD

For staff sequences we recommend creating a landing page that does not display any information intended for staff users only. The landing page should display a validation error if an unauthorized user (i.e., student) tries to access the staff sequence.

The following workflow segment checks if a user is permitted to access a staff sequence. It checks for a valid staffid.



1. The Entry section of the first form contains a **LookupUser** activity with the following properties:

UserId	staffid
UserName	formInstance.UserName
UserType	Cmc.Nexus.FormsBuilder.Entities.UserType.Staff
ValidationMessages	formInstance.ValidationMessages

2. An **If** condition checks if **staffid = 0**.

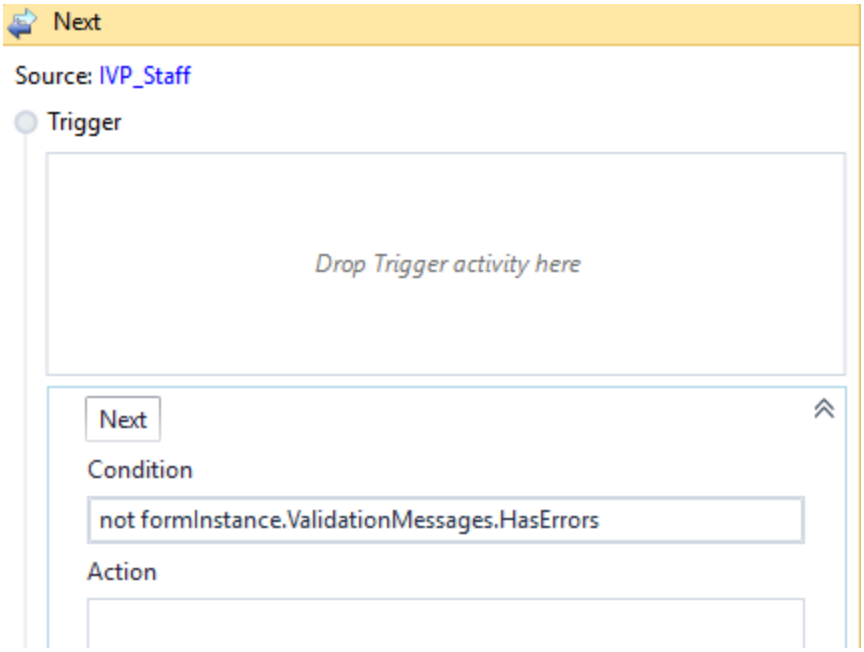
a. The **Then** branch contains a **CreateValidationItem** activity with the following properties:

Message	"This staff user was not found"
Messages	formInstance.ValidationMessages
MessageType	Error
Result	<blank>

b. The **Else** branch contains a **GetEntity<StaffEntity>** activity with the following properties:

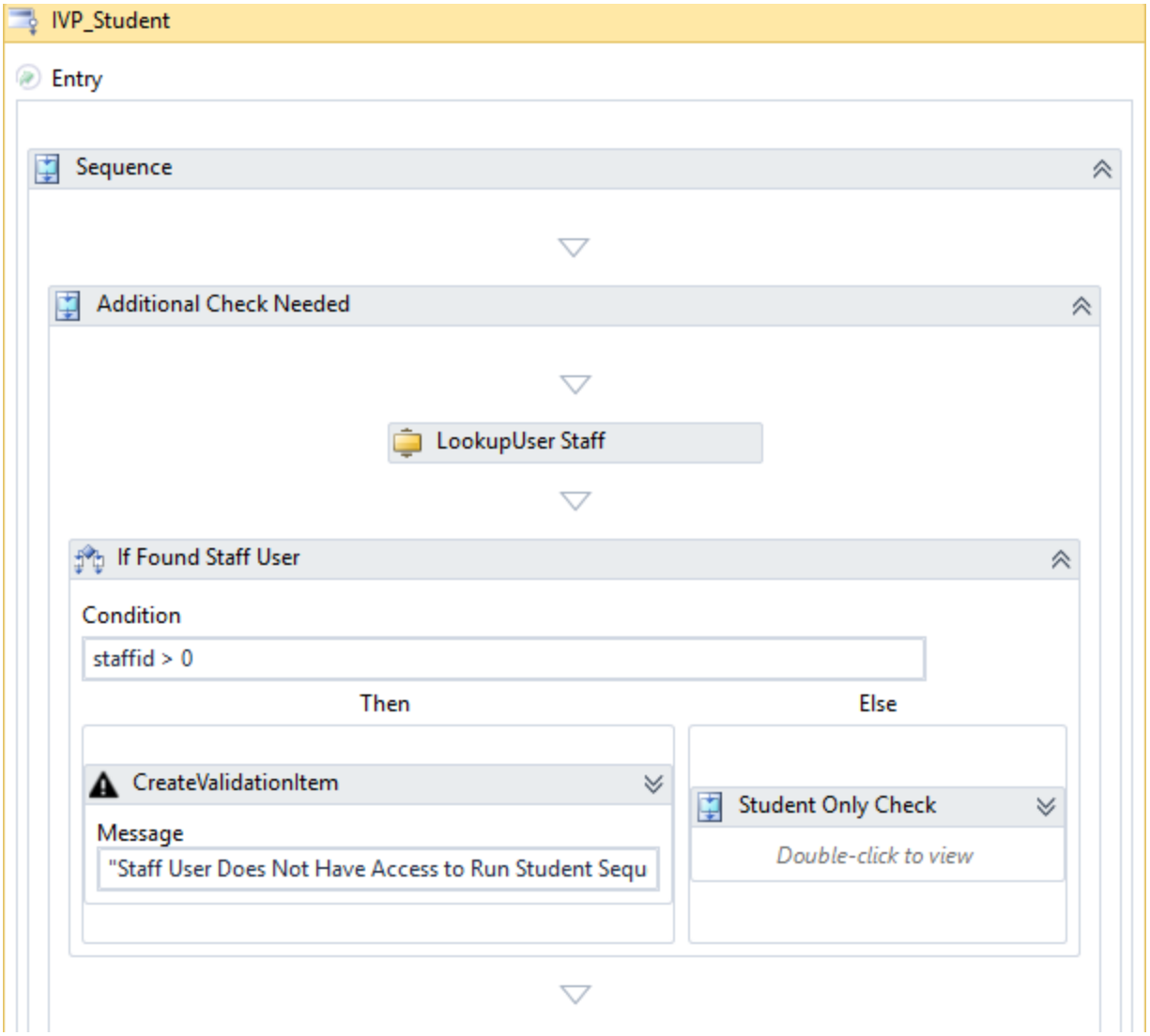
EntityId	staffid
Result	staffEntity

3. In the **Next** transition, check for a validation error to prevent the user from proceeding if the user's role does not match the role associated with the sequence. In the Condition field, specify **not formInstance.ValidationMessages.HasErrors**.



Verification of Student Role in Azure AD

The following workflow segment checks if a user is permitted to access a student sequence. It creates a new student record if needed. The workflow denies access for staff users and prevents creating student records for users who logged in as staff.



1. The Entry section of the first form contains a **LookupUser** activity with the following properties:

UserId	staffId
UserName	formInstance.UserName
UserType	Cmc.Nexus.FormsBuilder.Entities.UserType.Staff
ValidationMessages	formInstance.ValidationMessages

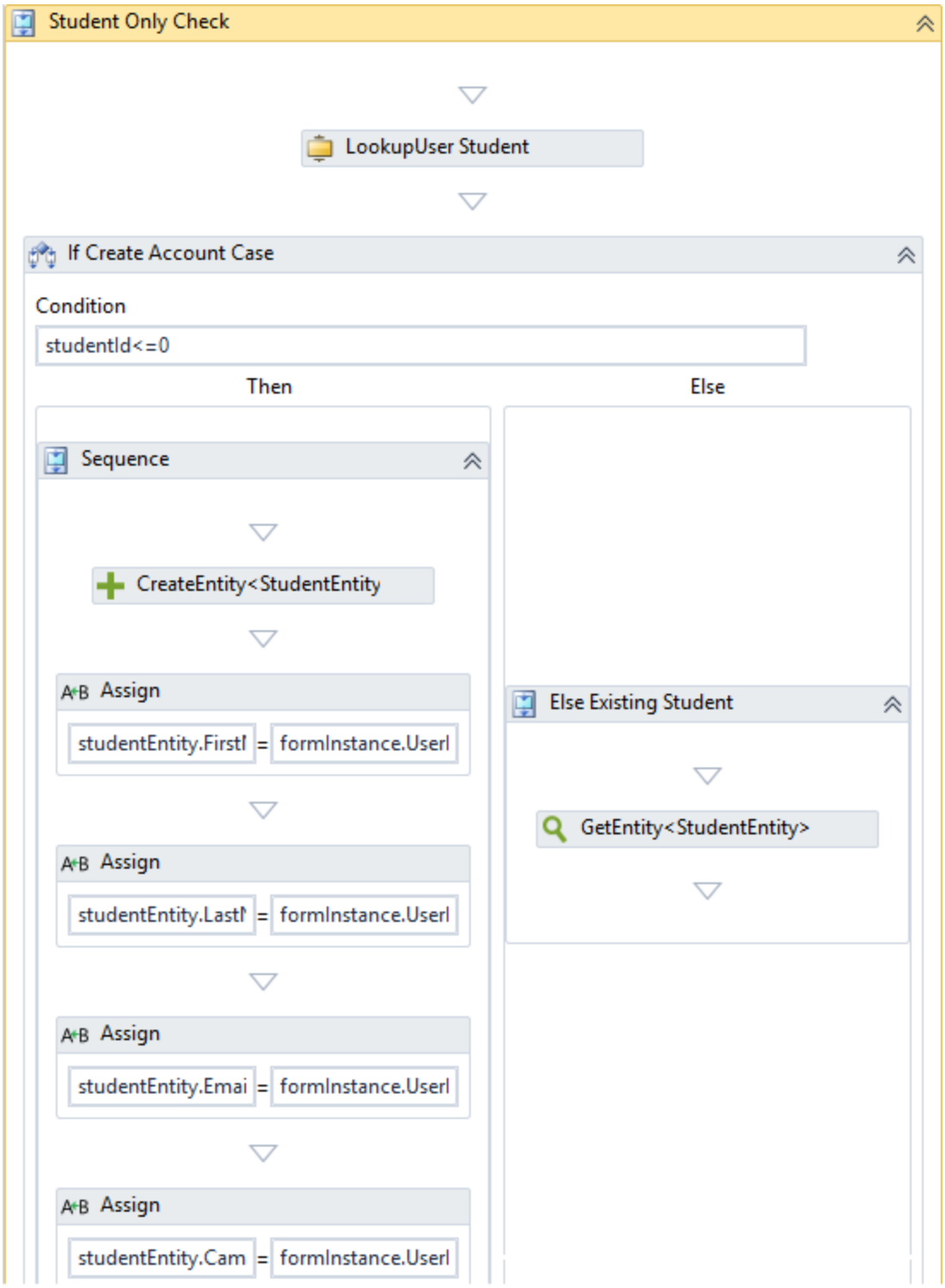
2. An **If** condition checks if **staffid > 0**.

   This condition checks if a staff user is accessing a student sequence. The workflow prevents the staff user from running student sequence and ensures that if a new student attempts to create an account, a student record would not be created for a staff user.

a. The **Then** branch contains a **CreateValidationItem** activity with the following properties:

Message	"Staff User Does Not Have Access to Run Student Sequence"
Messages	formInstance.ValidationMessages
MessageType	Error
Result	<blank>

b. The **Else** branch contains a sequence with LookupUser activity for UserType=Student, followed by a condition that checks for a studentId, retrieves an existing StudentEntity or creates a Student Entity, and assigns required properties to the StudentEntity.



The Else branch contains a **LookupUser** activity with the following properties:

UserId	studentId
UserName	formInstance.UserName

UserType	Cmc.Nexus.FormsBuilder.Entities.UserType.Student
ValidationMessages	formInstance.ValidationMessages

An **If** condition checks if **studentId<=0**.

i. The **Then** branch contains a **CreateEntity<StudentEntity>** activity with the following property:

Result	studentEntity

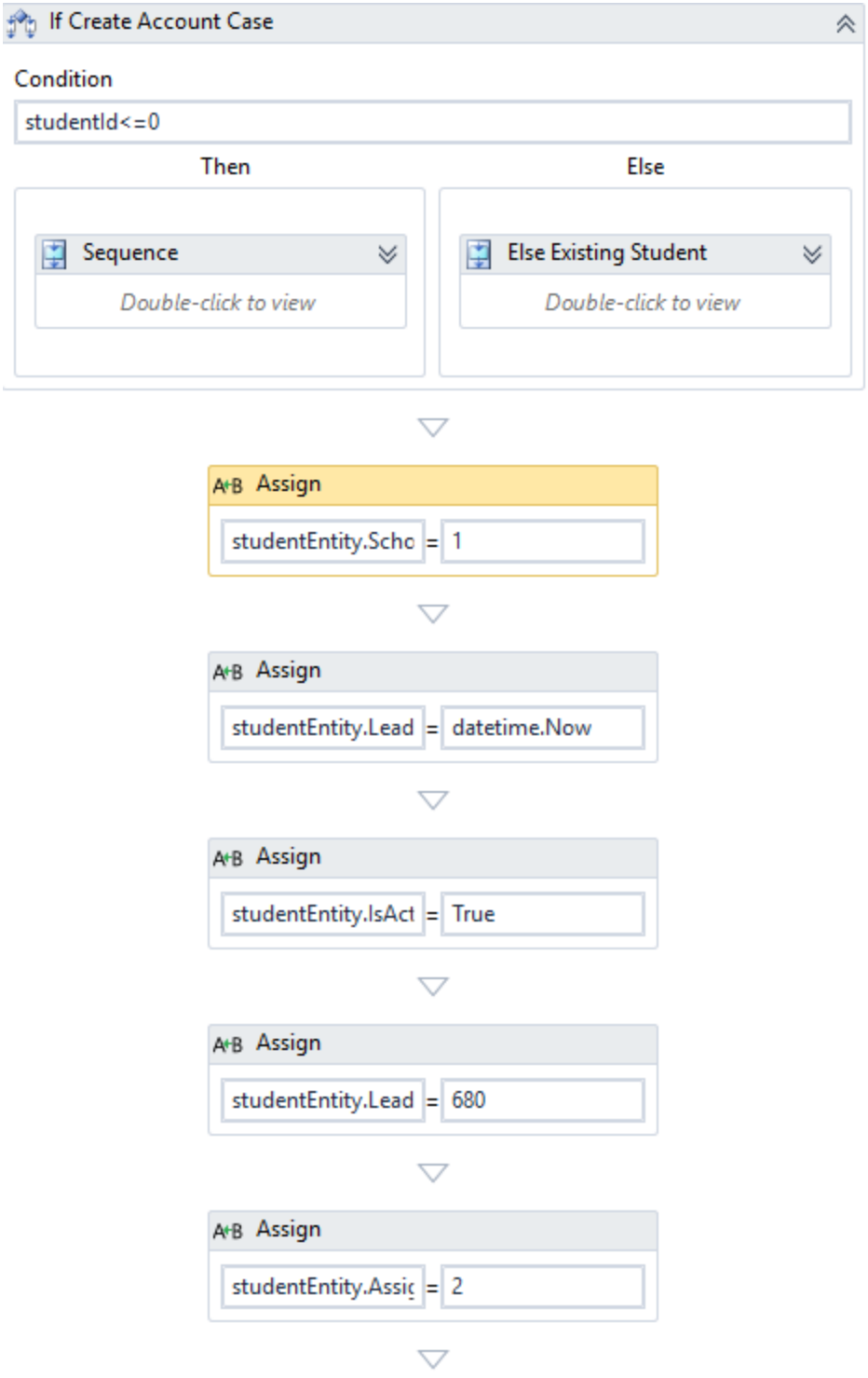
The following **Assign** activities are placed below the CreateEntity activity:

To	**Value**
studentEntity.FirstName	formInstance.UserInfo.FirstName
studentEntity.LastName	formInstance.UserInfo.LastName
studentEntity.EmailAddress	formInstance.UserInfo.EmailAddress
studentEntity.CampusId	formInstance.UserInfo.CampusId

ii. The **Else** branch contains a **Get<StudentEntity>** activity with the following properties:

EntityId	studentId
Result	studentEntity

The following additional properties may need to be assigned to the StudentEntity below the Then and Else branches with the CreateEntity<StudentEntity> and GetEntity<StudentEntity>.activities.

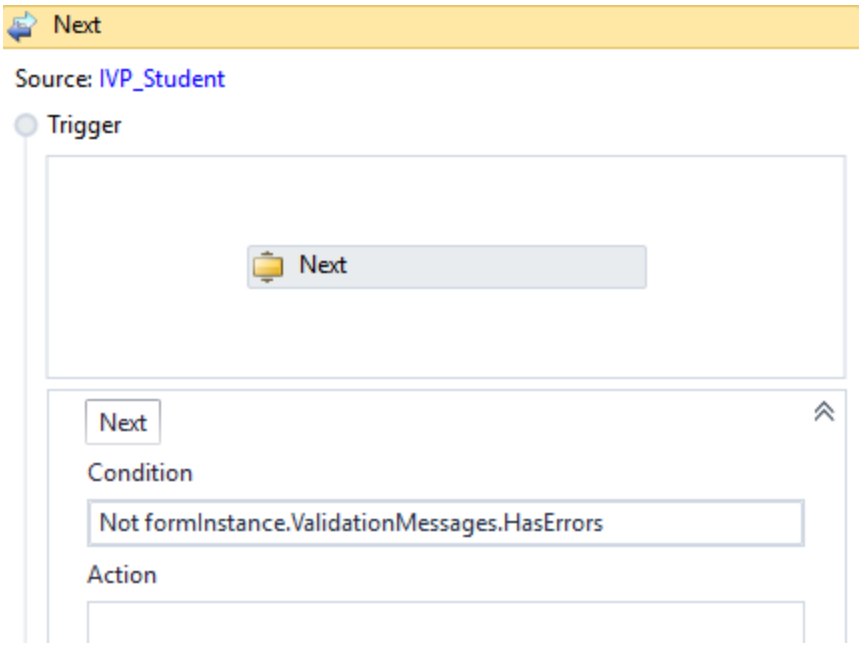


* Assign values as appropriate for your environment.

To	Value*
studentEntity.SchoolStatusId	1

To	Value*
studentEntity.LeadDate	datetime.Now
studentEntity.IsActive	true
studentEntity.LeadSourceId	680
studentEntity.AssignedAdmissionsRepId	2

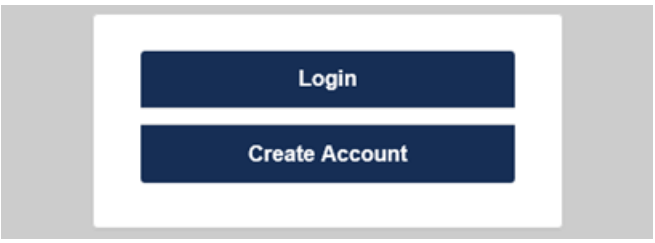
3. In the **Next** transition, check for a validation error to prevent the user from proceeding if the user's role does not match the role associated with the sequence. In the Condition field, specify **Not formInstance.ValidationMessages.HasErrors**.



For details about pre-populating **formInstance.UserInfo** variables when sequences are deployed in a cloud environment with Azure AD, refer to Azure AD Claims.

Login and Account Creation via Portal

For authenticated form sequences, Portal is used for both login and account creation and then control returns to Renderer. A page is displayed providing the option to log in using an existing Azure AD account or, if the user does not yet have an account, to create a new account. If "Create Account" is selected, the user will be redirected to the Portal.



In Portal, the user will follow the steps to create an Azure AD account. The following fields are displayed when creating a new account:

- Campus
- First Name
- Last Name
- Email address
- Username
- Password
- Security questions - if configured (e.g., Place of Birth, etc.)

Upon completion of the "Create new account" page, a new Azure AD account and Portal account (WpUser) will be created, and the user will be redirected back to the Portal login page.

The Login Locale setting can be used to add a drop-down list for locales on the Azure AD login page. For more information, refer to Login Locales.

If the "Create Account" option is selected on the Azure AD login page, the user is directed to the "New Account Creation" page in Portal. The header bar in Portal has a "Choose language" drop-down that is not linked to the Login Locales setting.

Renderer Web.Config Changes

When Forms Builder is installed in an Azure AD environment, the `<issuers>` section in the Forms Renderer web.config file will contain an "accountCreation" attribute that holds the Portal account creation URL.

```
    <!-- STS or Azure AD redirect URLs -->
    <issuers>
      <!-- <url key="A mapping issuerKey"
                                        value="An STS or Azure AD Login URL"
                                        accountCreation="If Azure AD, the portal account creation URL, c
      <url key="Student STS" accountCreation="<Portal account creation URL>" value-
e="https://<server>.<domain>:<port>" />
      <url key="CRM STS" accountCreation="" value="https://<server>.<domain>:<port>" />
    </issuers>
```

Additional Renderer web.config changes may be necessary if the workflows for your authenticated forms (e.g., RFIs) use **formInstance.UserInfo** variables. For more information, refer to Azure AD Claims.

Azure AD Claims

For authenticated forms, user information is captured in claims that are passed from the authentication service to Forms Renderer. You can extract user information from the claims and pass it to workflows using the **formInstance.UserInfo** variable. You can then use the values contained in the variable to pre-populate form fields such as first name, email, and user name.

The claims that are passed to Renderer are different depending on whether authentication is performed by Student/Contact STS or by Azure Active Directory (AD). Prior to the release of Forms Builder 3.4, the STS was the only authentication service available. The code for the formInstance.UserInfo variable was designed to work with the STS claims. The code does not automatically extract user information from claims passed by Azure AD. This means that any previously created authenticated forms with workflows that use formInstance.UserInfo variables will not provide the capability to pre-populate form fields with user information when deployed in a cloud environment with Azure AD.

However, the code for Forms Builder 3.4 allows overriding the claims in the web.config file of Renderer. These overrides map STS claims to Azure AD claims. The overrides prevent any existing workflows that rely on the formInstance.UserInfo variables from failing with null pointer reference errors in AzureAD environments.

The following default claim types from the Student STS are available:

```
CampusIdClaimType:          "http://s-
chemas.xmlsoap.org/ws/2010/08/identity/claims/campusid"
EmailAddressClaimType:      "http://schemas.xmlsoap.org/ws/2010/08/identity/claims/email"
FirstNameClaimType:         "http://schemas.xmlsoap.org/ws/2010/08/identity/claims/fname"
FullNameClaimType:          "http://schemas.microsoft.com/identity/claims/displayname"
LastNameClaimType:          "http://schemas.xmlsoap.org/ws/2010/08/identity/claims/lname"
MiddleNameClaimType:        "http://schemas.xmlsoap.org/ws/2010/08/identity/claims/mname"
RoleClaimType:              "http://s-
chemas.microsoft.com/ws/2008/06/identity/claims/role"
UserIdClaimType:
"http://schemas.xmlsoap.org/ws/2010/08/identity/claims/userid"
UserNameClaimType:          "http://schemas.xmlsoap.org/ws/2010/08/identity/claims/uname"
```

To override claims, you can add keys to `<appSettings>` section of Renderer web.config file. Use the claim type as the `key` and add your `value`. For Azure, the values are fixed, i.e., you must use the values as they are. For other environments, you can specify your own values to override any claim with anything you desire.

```
<add key="FirstNameClaimType" value-
e="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname" />
<add key="EmailAddressClaimType" value-
e="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name" />
<add key="UserNameClaimType" value-
e="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name" />
```

With the overrides above added to the web.config of the Renderer in the Azure AD environments, the FirstName, UserName, and EmailAddress fields will be populated and accessible within the workflow using the formInstance.UserInfo variable.

The following claims are currently not available in Azure AD. If any existing workflows contain formInstance.UserInfo variables with these properties, the properties will be blank (i.e., empty) when the forms are deployed in an Azure AD environment:

- CampusId
- LastName
- FullName (Technically, the Full Name claim is available in Azure AD, however, the Forms Builder code ignores it even the override is there.)

The above limitation continues to occur when an Azure authenticated form sequence is integrated with Student Portal as an external link and is configured to be viewed as a pop-up or as a tab. The form sequence will be redirected to Azure for authentication and the above Azure AD claims will be used to populate the formInstance.UserInfo variable.

However, this limitation is overcome when the form sequence is launched within an iframe in Student Portal as an external link. The following information will be retrieved from the Student Portal web service (IntegrationServices/CMCIntegrationService.asmx) using the AuthGuid which is passed as a query string when the user selects the external link, thus enabling successful SSO login.

The following table lists mappings of the FormInstance.UserInfo variable to the Student Portal webservice response (ExternalAuthorization).

FormInstance.UserInfo Field	ExternalAuthorization Field Value
UserId	CampusPortalID
UserName	UserCode
FirstName	FirstName
LastName	LastName
MiddleName	Dummy
EmailAddress	Email
CampusId	CampusList[0]
Roles	RoleType
ExternalUserId	UserID

This method of launching the form sequence in an iframe can be implemented only on the Student Portal, and not on the Applicant Portal.

For details about designing sequences for authentication with Azure AD, refer to Azure AD Authentication.

Embed a Form on a Website

You can easily integrate a form sequence into a website using the HTML <iframe> tag. This will display the form within a frame on a webpage. Any redirects and submit functions on the form will be displayed within the frame. The user does not need to navigate away from the webpage to complete the form.

Embedding a form using the <iframe> tag allows you to maintain the form separately from the website. You can update the form in Forms Builder and the updates will appear on the website as long as the URL of the sequence remains the same.

You can use CSS to style the <iframe>. You can apply borders, scroll bars, set up responsive behavior, and so on. The default CSS settings for the <iframe> element in most browsers are:

```
iframe:focus {
outline: none;
}
iframe[seamless] {
display: block;
}
```

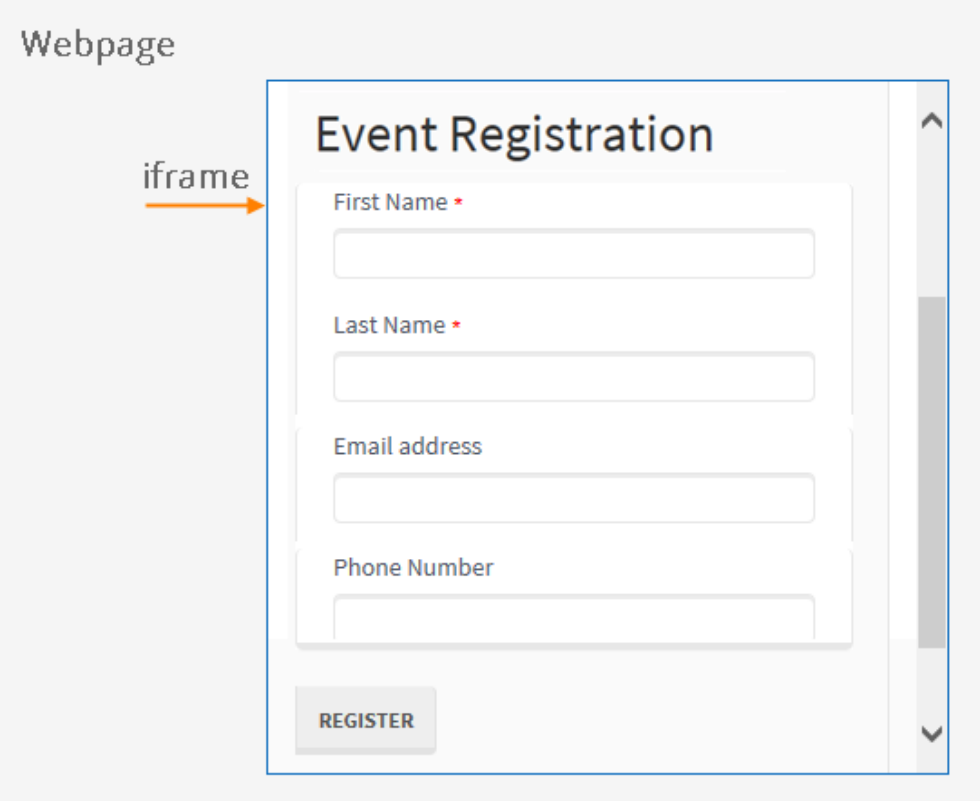
Procedure

1. Build your form sequence and access it from the **Sequence List**.
2. Create or edit a page on your website. Add an **<iframe>** tag in the body of the webpage.
3. In the <iframe> tag, specify the **width** and **height** of the frame and the **URL** of the form in the Sequence List.

   Example

   <iframe width="80%" height="700px" style="border: 1px solid green" src-c="https://<server>:<port>/#/renderer/511">
4. **Save** the webpage and **publish** it to your website.
5. When users go to the webpage, they'll access the form referenced from the Sequence List in a frame on

the webpage.


Webpage
iframe
Event Registration
First Name *
Last Name *
Email address
Phone Number
REGISTER

Forms Builder Form as External Link in Portal

You can create links to Forms Builder forms as external links in Portal. Forms Builder forms can be used to enable students to fill out forms for various purposes, e.g., event sign-up forms, parking passes, immunization forms, etc. To access these forms, Portal users will log in as student or staff using single sign-on (SSO), select a Portal menu option (e.g., the External Links), and select a link that will bring up a form.

This is available for both on-prem and cloud environment (1.0, multi-tenant).

Depending on the options selected in the Portal Configuration tool, a linked form can be displayed as popup, new browser tab, or as an IFrame within the Portal window.

For anonymous or authenticated form sequences, the Portal link can be configured with auto-login (SSO) enabled.

For authenticated sequences:

a. The AuthGuid value provided by Portal enables the user to be impersonated, which also helps to establish an SSO-login into the form sequence that's configured as an external link in Portal.

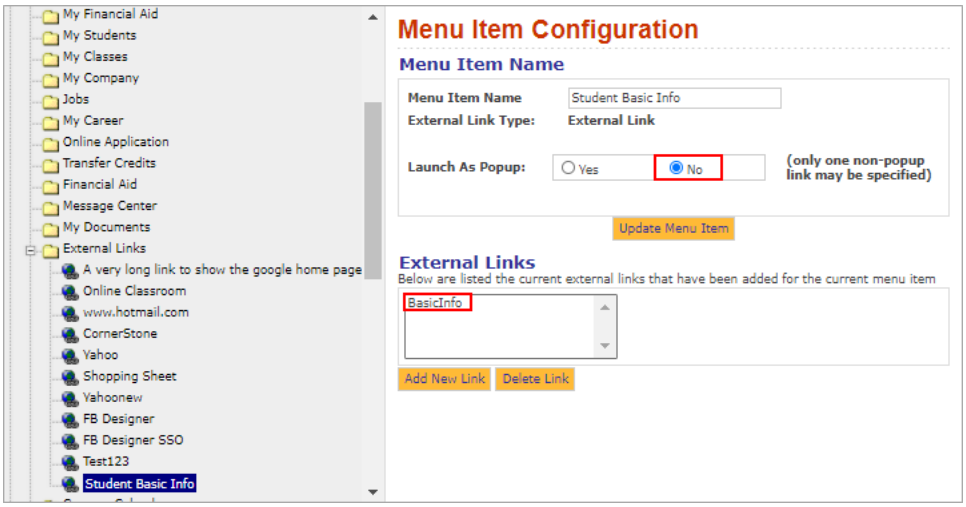
b. User related information can be obtained through the FormInstance.UserInfo variable which can be consumed to perform workflow related activities using Workflow Composer.

End users can use the default Logout button in Portal to log off from the Portal and the form configured in the iframe. No additional log out button will be displayed for the form.

Configure an IFrame Link in the Portal Configuration Tool

1. Log in to the **Portal Configuration tool**.
2. Expand the Server node and select your **Campus**.
3. Select **Menu Groups** > **External Links**.
4. Select the **Menu Item Configuration** for your form. In our example, the Menu Item is named "Student Basic Info".
5. In the **Launch As Popup** option, select **No**. This option displayed the linked form in an IFrame within the Forms Builder window. Select **Update Menu Item** to save your changes.

   Select the menu item ("BasicInfo" in our example) in the External Links list to access additional configuration options.



6. In the External Link Information section, configure the following items:

    a. **Target URL**: Specify the URL for the rendered Forms Builder sequence.

    b. **Requires Auto-Login (SSO)**: Select **Yes**.

    When auto-login is enabled, Portal generates a unique GUID for the session. Portal will insert the session data with the Student ID (or Staff ID) along with the AuthGuid value into a data table. The AuthGuid will apply to any URL that is specified in the Target URL and Auto-Login URL fields.

    In our example, the AuthGuid will be appended to the Forms Builder sequence URL when a user selects the external link. The values passed in the iframe element are displayed in the browser development tools as shown below:

    ```
    <iframe src="https://<server:domain>:9003/#/renderer/5619?AuthGuid=e28733a4-cfe8-40d6-bf71-3bd9535b645e&amp;TargetURL=https://<server:domain>:9003/#/renderer/5619" id="_ctl0_PlaceHolderMain_iframeContent" title="Links" align="left" frameborder="no" style="WIDTH: 100%; HEIGHT: 680px">

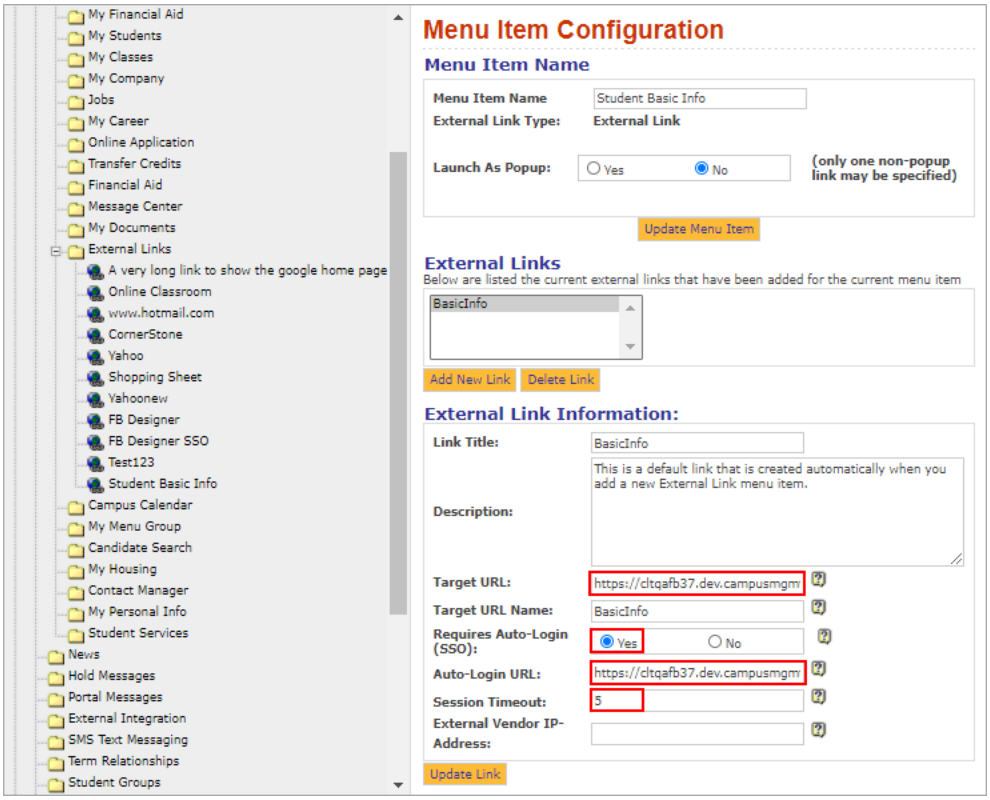
    </iframe>
    ```

    c. **Auto-Login URL**: Specify the link to the rendered Forms Builder sequence.

    d. **Session Timeout**: Specify a session timeout value in minutes to ensure that the session is closed when the user is inactive.

    e. **External Vendor IP-Address**: Specify outbound IP address of Forms Builder Renderer.

    To configure the external vendor IP address for Forms Builder Renderer in the Portal configuration, raise a DevOps ticket to obtain the Azure 2.0 and 1.x environment outbound IP addresses from the SCM team. After receiving the outbound IP addresses, you can configure them in the Portal configuration tool. For on-premise customers, reach out to the team responsible for owning the servers (Occasionally, customers manage their own servers).



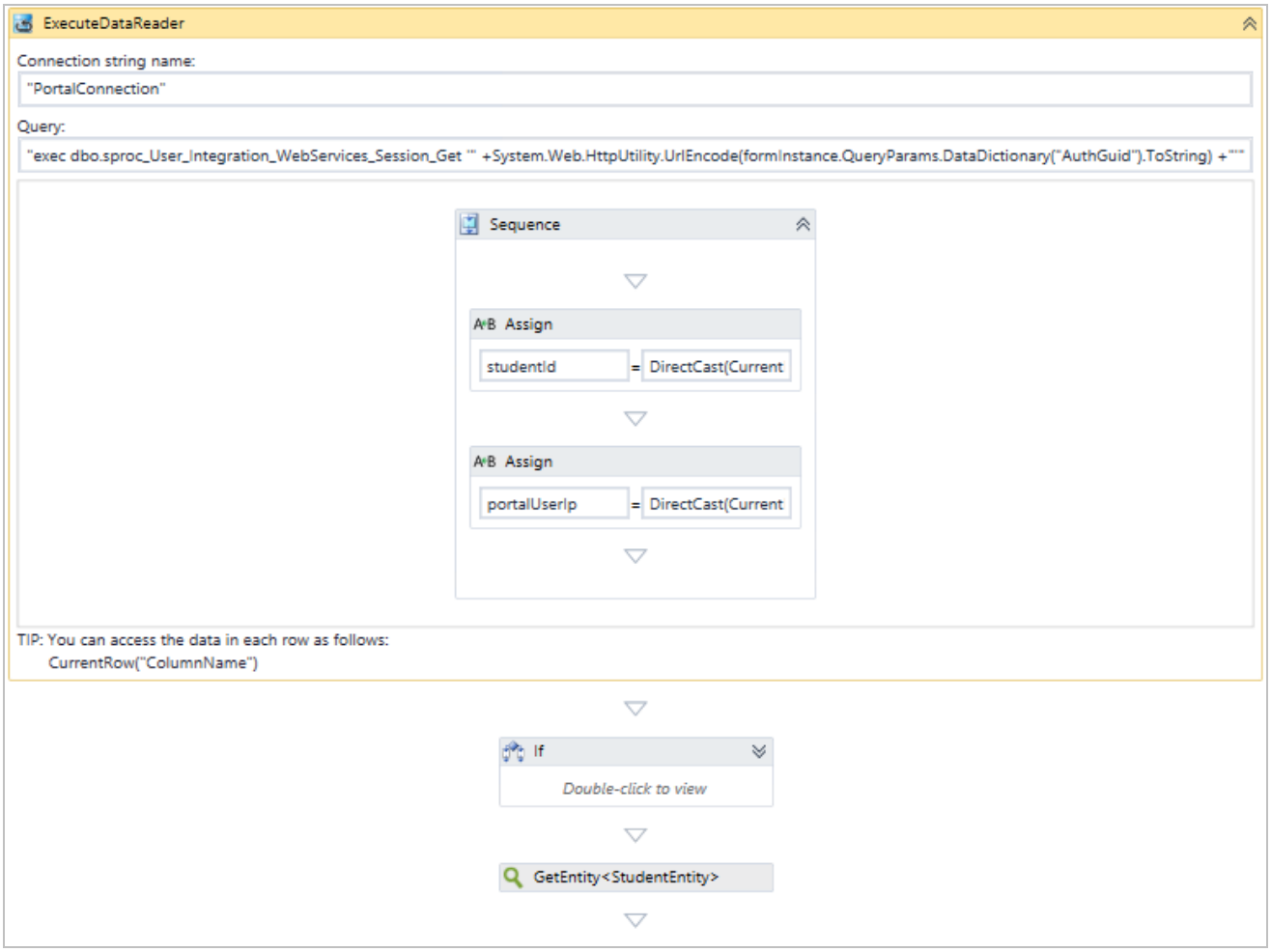
7. Select **Update Link** to save your changes.

Modify the Workflow to Resolve the AuthGuid (for anonymous sequences)

The following steps were used as a workaround in previous versions to authenticate users in Azure environments. These steps were required due to security issues described in version 4.0 of this topic.

From release 4.1, while it's recommended to use the steps described above to configure a new form sequence, the following steps performed in previous versions remain valid for enabling auto-login (SSO) of previously configured form sequences.

1. Launch Workflow Composer and open the workflow for the Forms Builder sequence that is linked in Portal.
2. Insert an ExecuteDataReader activity at the top of the workflow.



a. Specify a **Connection string name**.

b. Specify the following value in the **Query** field to call a stored procedure:

   "exec dbo.sproc_User_Integration_WebServices_Session_Get '" +System.Web.HttpUtility.UrlEncode(formInstance.QueryParams.DataDictionary("AuthGuid").ToString) +"'"

   The sproc uses the AuthGuid value provided by Portal in the IFrame URL and retrieves the associated student (or staff) ID from the wpExternalAppSessions table in the Portal database.

   The student (or staff) ID value can then be used in a GetEntity<> activity to obtain the student (or staff) entity for further workflow steps.

3. (Optional) If desired for additional security, you can include **Assign** activities to tie the AuthGuid to a Portal user IP address. The IP address is the URL of the Forms Builder Renderer. The IP address is part of the formInstance entity.

Assign To	Value
studentId	DirectCast(CurrentRow("UserID"),Int32)
portalUserIp	DirectCast(CurrentRow("BrowserIPAddress"),String)

However, don't rely too much on the IP address because student users can be behind a firewall or VPN and the true IP address may not be available. Refer to Collect IP Address.

Link Sequences to Portal Document Center

When form sequences are linked to the Document Center in the Student Portal, users can launch the sequences directly from the Document Center without having to access the Sequence List.

The URLs for sequences have different formats depending on whether Azure AD authentication is used.

Sequence URL **without** Azure AD:

https://<server:port>/#/renderer/<sequence number>

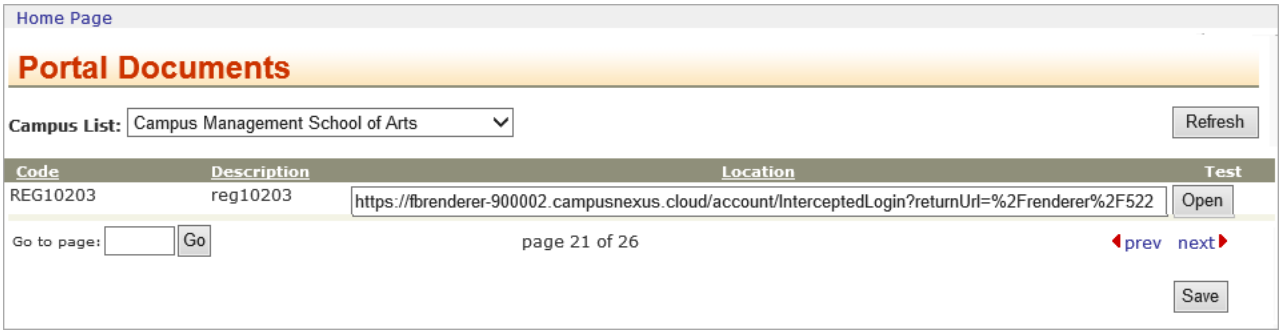
Sequence URL **with** Azure AD:

https://<server>.campusnexus.cloud/**account/InterceptedLogin?returnUrl=**%2Frenderer%2F<sequence number>

When the sequence URL with the "intercepted login" part is made available in the Student Portal, logged in users can access the sequence without having to log in again.

Update Documentation Links in Portal

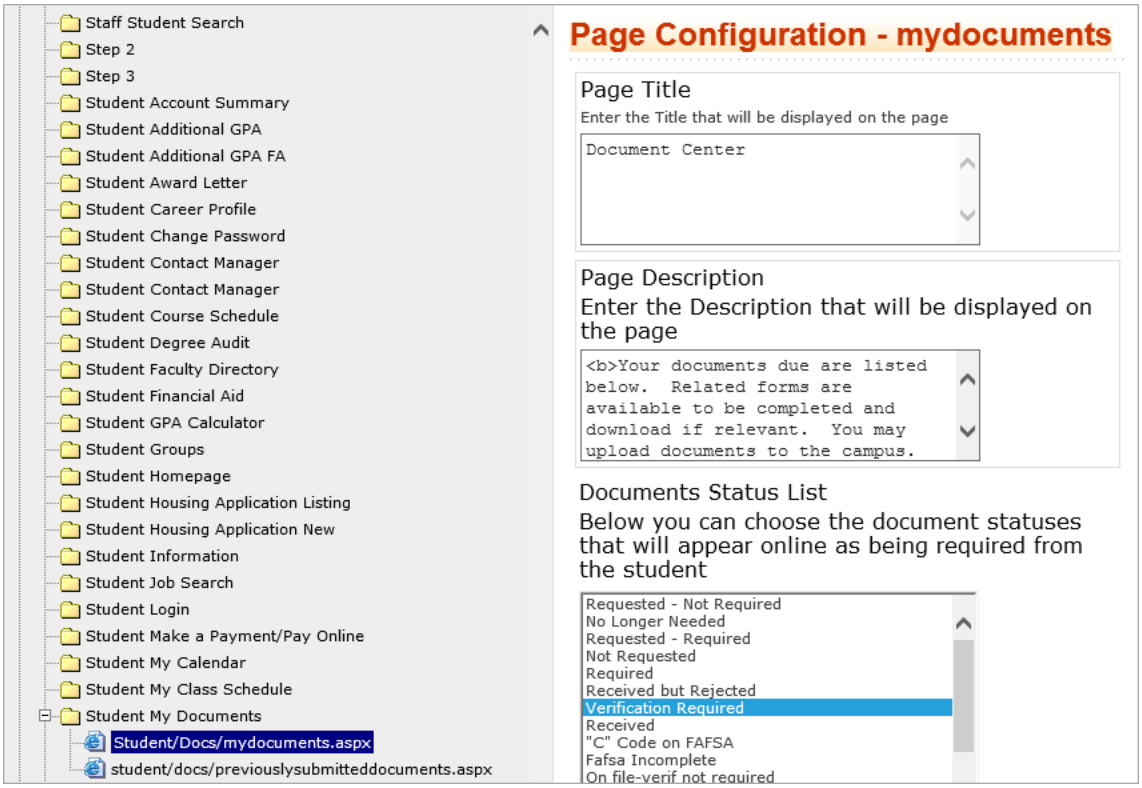
1. Access the **Portal Admin Console**.
2. Under Settings and Environment, select **Portal Documents**.
3. Select the **Campus**.
4. Update the URLs in the **Location** fields.
5. Select **Save**.
6. Select the **Open** button to verify the updated link.



Associate Document Statuses with Documents

Optionally, you can associate document statuses with the documents listed in the Documentation Center of Student Portal.

1. Access the **Portal Configuration** tool.
2. Select the **Campus**.
3. Expand **Page Transactions** and navigate to **Student My Documents**.
4. In the **Documents Status List**, select the applicable document statuses.



5. Select the **Update** button at the bottom of the page.
6. Access the Student Portal, navigate to Documentation Center, and test the updated links.

Multiple Renderer URLs

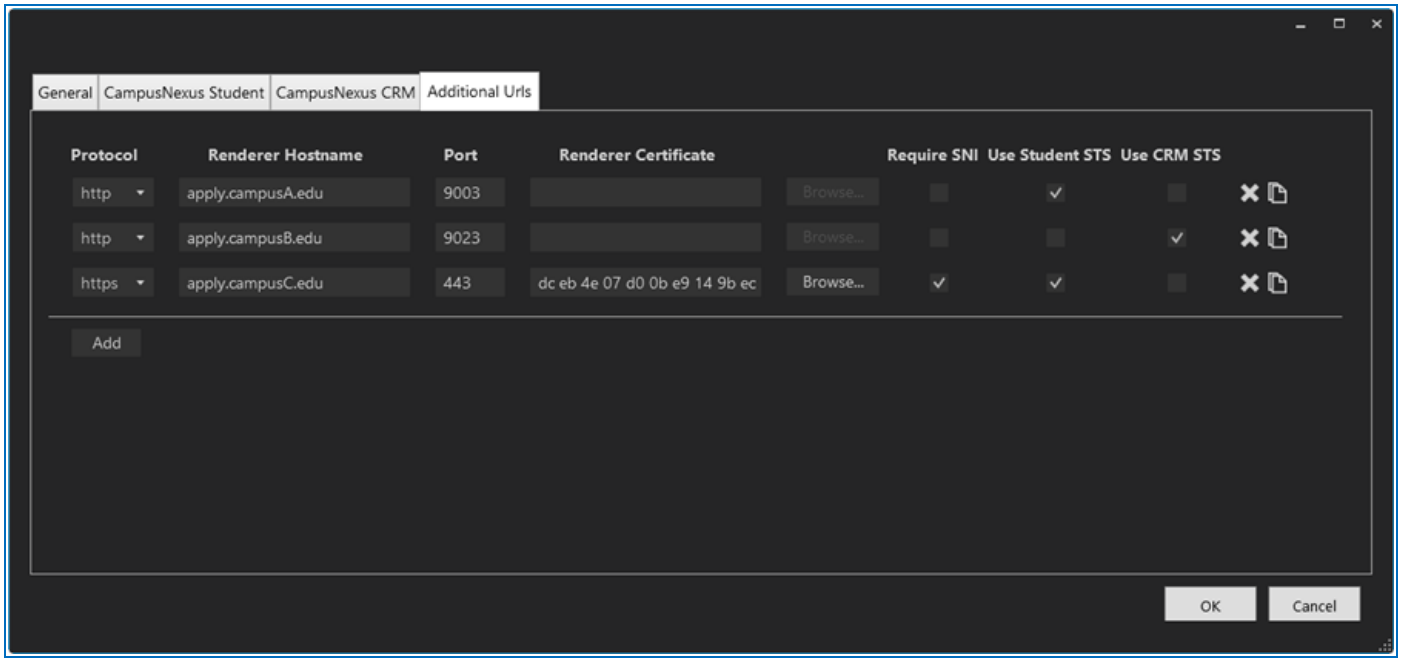
Using a single instance of Forms Builder (Designer and Renderer), you can design form sequences to be displayed at different URLs with different branding.

The example below dynamically modifies styles/logos by using the **Apply Themes Dynamically** sequence property and modifies sequences at run-time based on workflows on the issuer site. This method is applicable only when Forms Builder is installed on premises or for non-Azure customers.

Campus Example

An institution with multiple campuses (Campus A, B, and C) has a Portal website with unique URLs and distinct branding such as logos for each campus. The campuses share an Anthology Student database and/or a CampusNexus CRM database and use the same forms for applications, requests for information, etc.

The Forms Builder Renderer website of the institution is configured with bindings for the URLs of Campuses A, B, and C as shown below in Installation Manager.



The settings specified on the Additional Urls tab are written to the `<authenticationConfigSection>` section in the Renderer web.config file.

- The `<realms>` section contains a key and value for each additional incoming Renderer URL.
- The `<issuers>` section contains a key and value for the authentication services, i.e., Student STS and CRM STS.
- The `<mappings>` section contains the mapping between realm keys and STS keys.

```
<authenticationConfigSection>
  <!-- incoming urls -->

  <realms>
    <!-- <url key="" value="" /> -->
    <url key="CampusA" value="http://apply.CampusA.edu/" />
    <url key="CampusB" value="http://apply.CampusB.edu/" />
    <url key="CampusC" value="http://apply.CampusC.edu/" />
  </realms>

  <!-- STS redirect urls -->

  <issuers>
    <!-- <url key="" value="" /> -->
    <url key="CampusASTS" value="https://studentsts.CampusA.edu:81"/>
    <url key="CampusBSTS" value="https://crmsts.CampusB.edu:81"/>
    <url key="CampusCSTS" value="https://studentsts.CampusC.edu:81"/>
    <url key="Student STS" value="https://<server>.anthology.com:811"/>
    <url key="CRM STS" value="https://<server>.anthology.com/cmc.crm.sts/"/>
  </issuers>

  <mappings>
    <!-- <mapping realmKeys=" comma separated realm keys or * for wildcard match "
                product=" name of the product or * for wildcard match "
               issuerKey=" url key of the issuer " /> -->

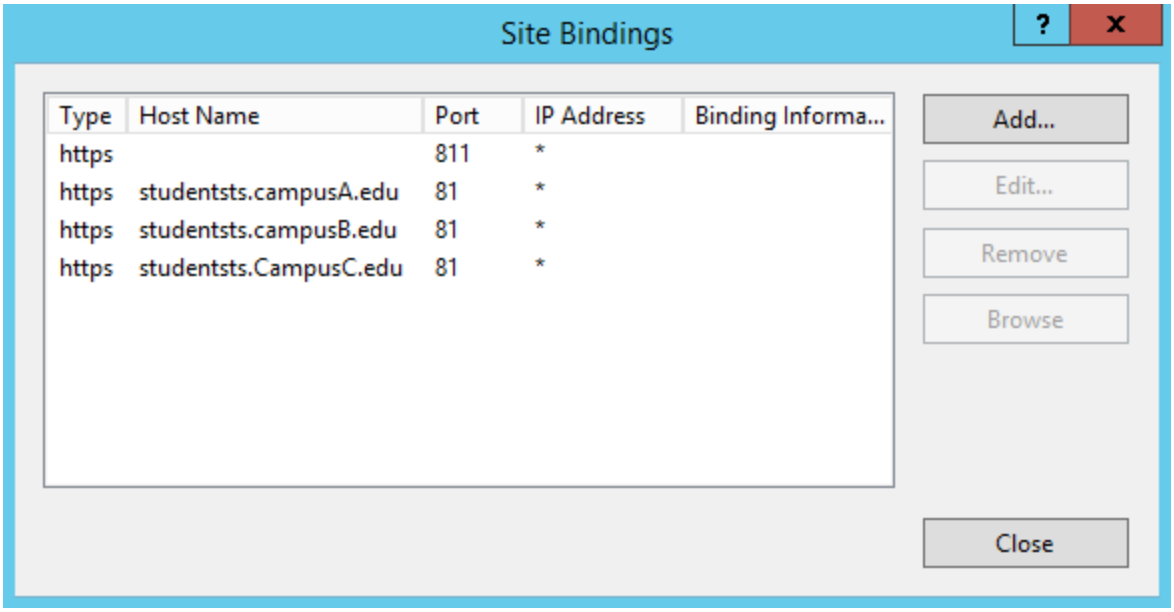
    <mapping realmKeys="CampusA" product="Student" issuerKey="CampusASTS"/>
    <mapping realmKeys="CampusB" product="CRM" issuerKey="CampusBSTS"/>
    <mapping realmKeys="CampusC" product="Student" issuerKey="CampusCSTS"/>
    <mapping realmKeys="*" product="Student" issuerKey="Student STS"/>
    <mapping realmKeys="*" product="CRM" issuerKey="CRM STS"/>
  </mappings>

</authenticationConfigSection>
```

Multiple Renderer URLs for Multiple Student STS Instances

Installation Manager currently supports a single Student STS with multiple Renderer URLs. If an institution has set up additional binding URLs for Student STS instances (refer to example below), the additional STS URLs need to be manually updated in the `<issuer>` and `<mapping>` sections of the Renderer web.config file.

The additional Student STS URLs need to be added to the StudentSTSURL column of the `wpURL` table in the Portal database.



Add Custom Style Sheets and Images

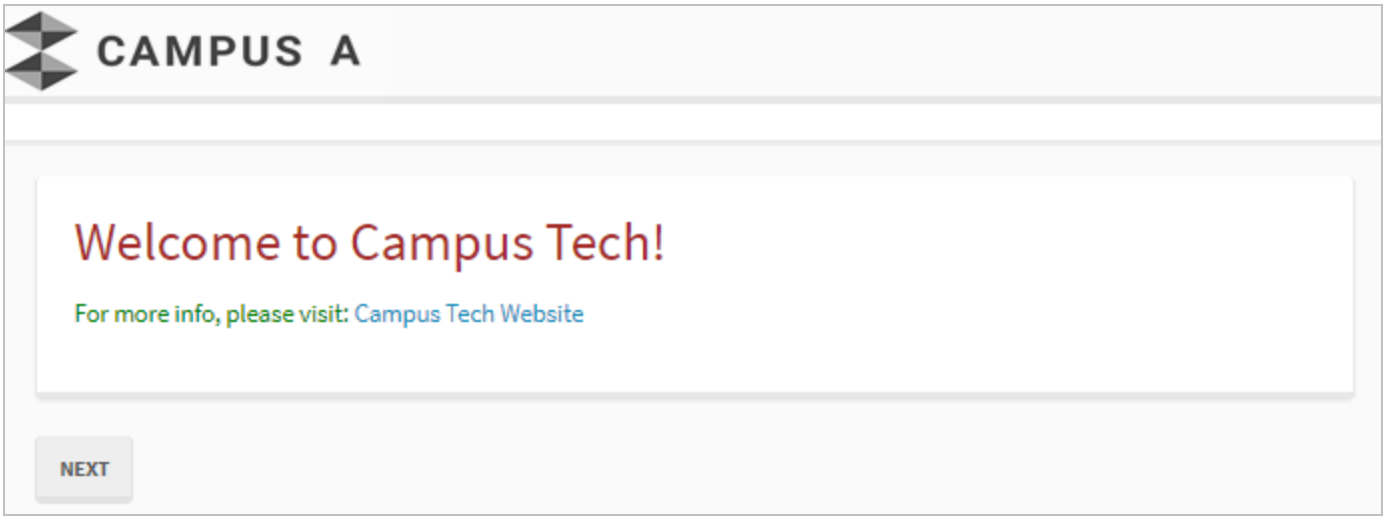
Custom style sheets that will be applied to form sequences accessed from each campus need to be added to the Custom Content workspace of Form Designer. In our example, the institution named "MyUniversity" has 3 campuses with unique logos. So, we add custom .css files for CampusA, CampusB, and CampusC, and custom logos for each campus to custom content.

In our example, each .css file modifies the image that is displayed in the top left corner of the form sequences. We customized the following style definition:

```
.navbar-brand {
    padding: 0;
    background: url('images/CampusNexus-SVG.svg') center / contain no-repeat;
    width: 200px;
    margin-left: 5px;
    }
```

We changed the image and the way it's referenced, added the **!important** rule to override the default logo, and removed width and margin.

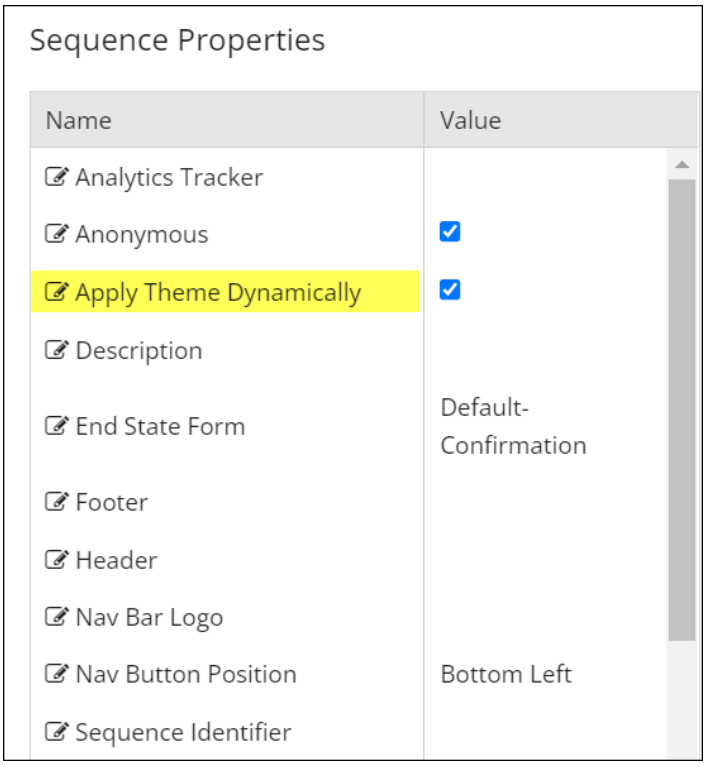
```
.navbar-brand {
    padding: 0;
    background: url(getCustomContentDataAsFile?src=CampusA_logo.png) center / contain
no-repeat !important;
    }
```



Associate Sequences With Uploaded CSS Files

The form sequences that will be accessed from different campuses need to be associated with the uploaded CSS files.

1. In Sequence Designer, open your sequence.
2. In the Sequence Properties, select **Apply Themes Dynamically**.

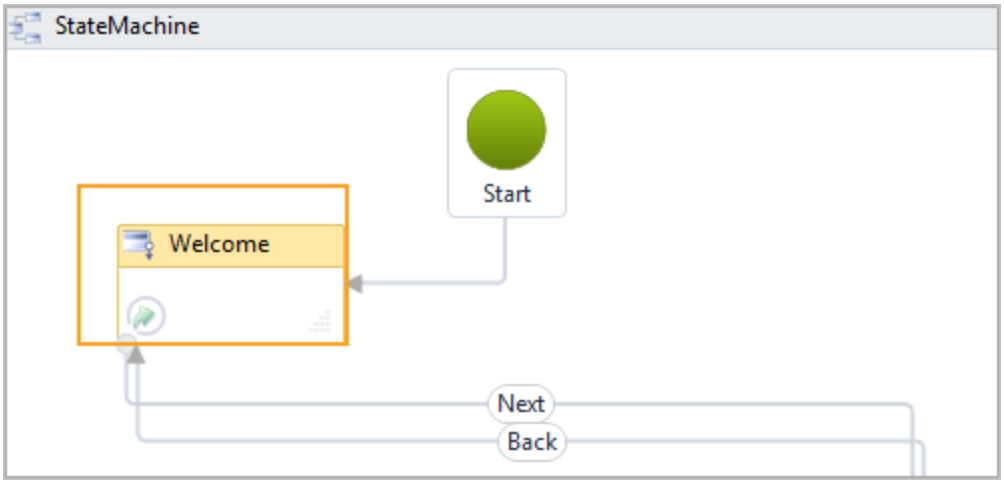


3. Save the sequence.

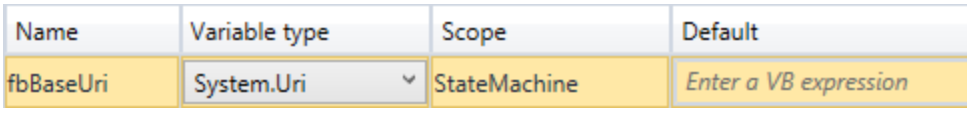
Select Style Sheets Using Workflow Activities

The first state in the form sequence workflows needs to be modified to create the association between URLs and custom style sheets.

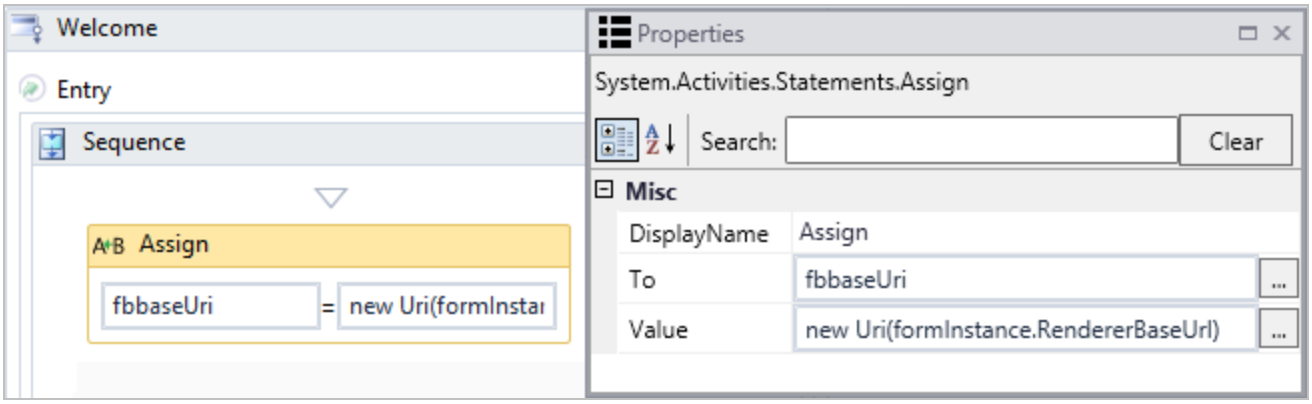
1. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
2. Double-click the **first state** in the workflow. In our example it is the Welcome form.



3. Drag a **Sequence** activity into the Entry section of the Welcome form.
4. Create a variable of type **System Uri**.

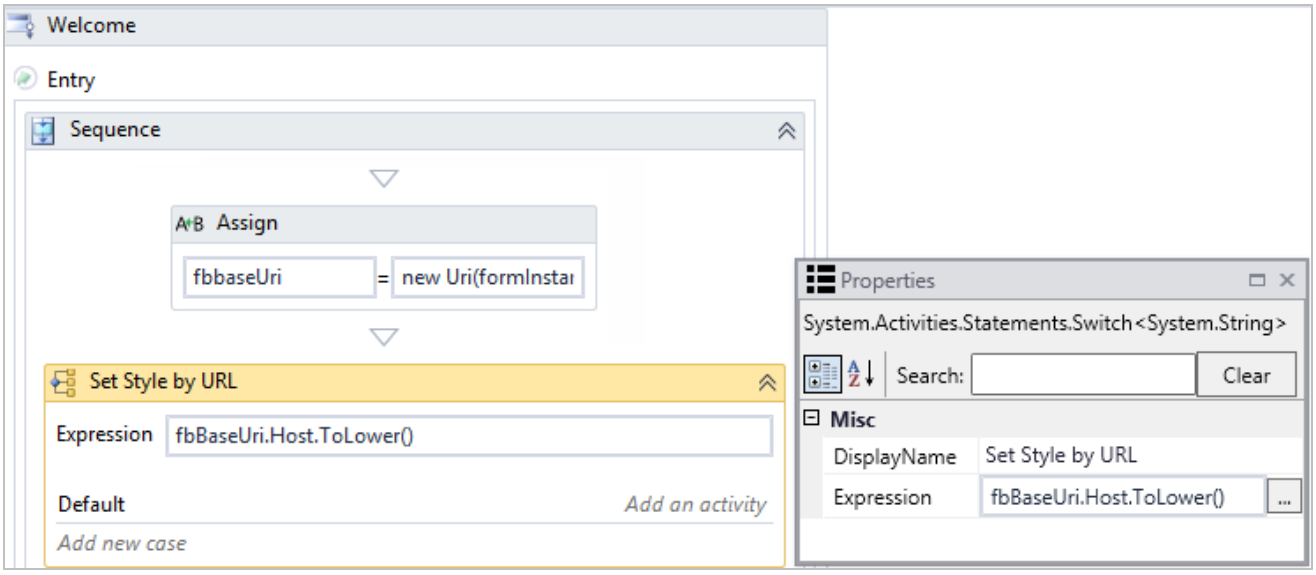


5. Drag an **Assign** activity into the new Sequence and specify the following properties:
   - To: **fbbaseUri** (the name of the variable created in the previous step)
   - Value: **new Uri(formInstance.RendererBaseUrl)**



6. Drag a **Switch** activity below the Assign activity and specify the following properties:

- Type: **System.String** (Use the "Browse for Type" option to find this type.)
- Display Name: **Set Style by URL**
- Expression: **fbBaseUri.Host.ToLower()**



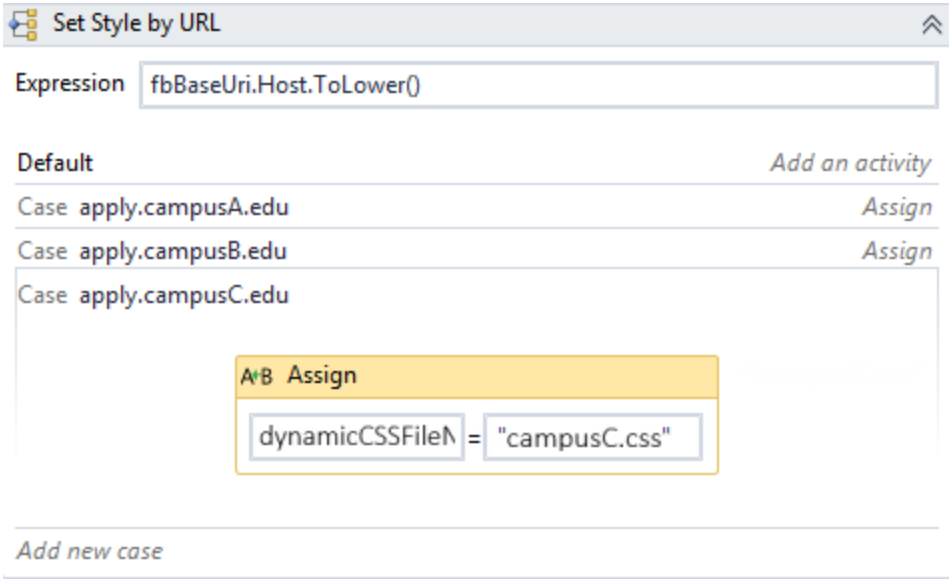
7. Select **Add new case** in the Switch activity.

   a. In the **Case Value** field, specify **apply.campusA.edu** (This is the URL for Campus A. It is defined in the IIS bindings.)

   b. Drag an **Assign** activity into the Case section and specify the following properties:

      - To: **dynamicCSSFileName** (This is the model name that's stored in the database for institutions that use the **Apply Themes Dynamically** option, and this model name will be referenced in workflows for CSS assignment.)
      - Value: **"campusA.css"** (This is a css file that is uploaded to custom content.)

   Repeat this step for any other custom style sheets. In our example, we added the assign statements for the style sheets of Campus B and Campus C.

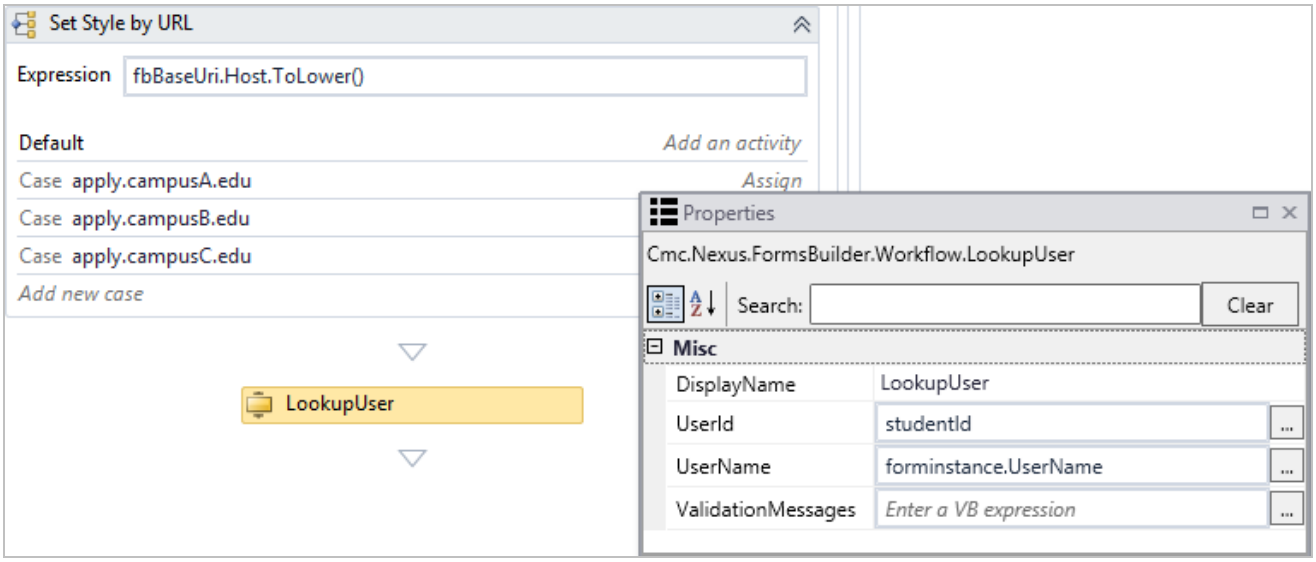


You can copy and paste the sequence with the Assign statement for the BaseUrl and the Switch activity to other form sequences that are rendered using the custom style sheets.

> Additionally, **dynamicCSSFileName** must be declared as a custom argument of String type and In/Out direction.

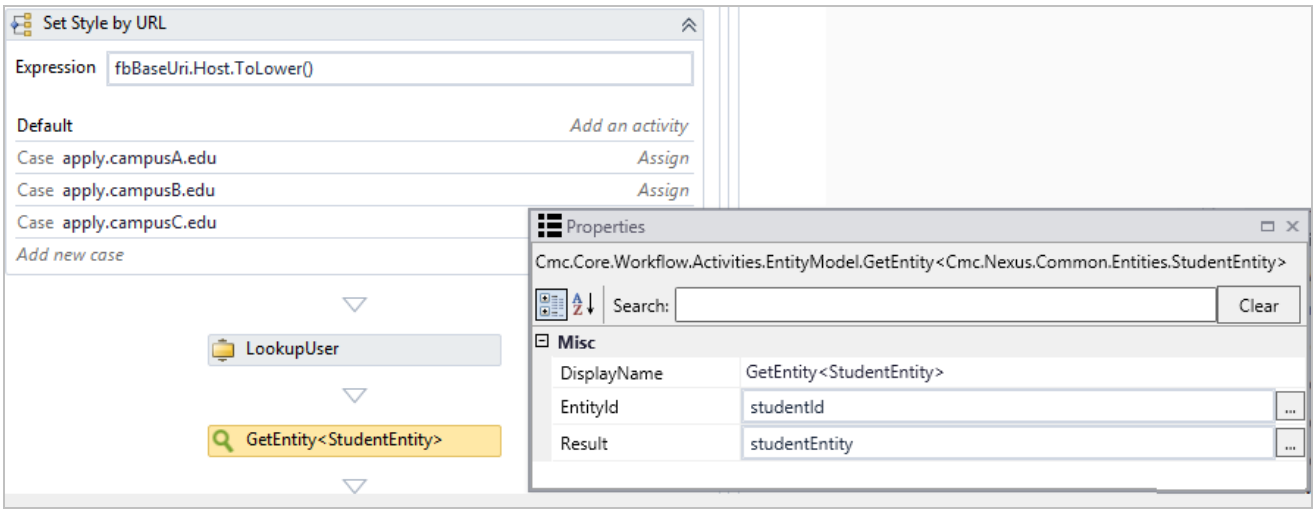
If your form sequences need to retrieve student records from the database, you may want to add the LookupUser and GetEntity activities to the copied sequence.

8. Drag a **LookupUser** activity into the workflow. Specify the following properties:
   - Userid: **studentId** (This variable of type Int32 must be defined in the workflow.)
   - UserName: **formInstance.UserName**



9. Drag a **GetEntity<>** activity into the workflow. Browse for entity type **<StudentEntity>** and specify the following properties:

   - EntityId: **studentId**
   - Result: **studentEntity**



Preview and Update a Form Sequence

When you design or modify a form or sequence you may want to preview the rendered output before making it available to end-users. To do so:

1. In Form Designer, create a form with the required fields/components and properties. **Save** the form.
2. In Sequence Designer, add the form to a sequence and **save** the sequence.
3. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
4. In Workflow Composer, **save**, **publish**, and **enable** the workflow for the sequence.
5. In Forms Renderer, select and review the rendered sequence. Keep the browser open.

   Notes

   - Use different browsers for Form Designer and Forms Renderer. For example, open the Sequence List (i.e., Renderer) in Google Chrome while Form Designer/Sequence Designer is open in Internet Explorer.
   - You can also use the incognito (or private browsing) mode in any browser. For example, open Form Designer in Chrome and then open the Sequence List in another instance of Chrome in incognito mode. You must open another instance of the browser (not just another tab) in incognito mode.
6. To revise the sequence:
   - In Form Designer, edit the layout of fields/components, change property values as needed, and save the form.
   - In Workflow Composer, edit the workflow for the sequence, save, publish, and enable the new version of the workflow.
7. Return to the browser with the rendered sequence, press **F5** or **Ctrl+F5**, and check if the revisions produced the desired results.

   Repeat the last two steps until you are satisfied with the rendered sequence.

When the results are approved, publish the URL for the rendered sequence and ensure that the correct workflow version is enabled.

If you decide not to use a sequence, do not publish the URL, and disable the workflow. When a workflow is disabled or deleted, end users accessing the associated sequence will receive an error message.

If you modify a published workflow and then publish the updated workflow version, end users can complete any active instances of the previous workflow version without receiving an error message.

Redirects for Rendered Sequences

All Forms Builder sequences are available in the Sequence List at the Renderer URL: http or https://<server->.<domain>:<port>**/#/Sequencelist**

The default port number is 9003. The installer configures the port. Your Forms Builder installation may not use the default port.

You can launch all sequences from the Sequence List. A redirect occurs depending on whether the sequence is anonymous or authenticated.

Anonymous Sequences

Anonymous sequences do not have a redirect. These sequences can be accessed by appending the sequence ID to the Renderer URL, for example:

https://<server>.<domain>:<port>/#/renderer/468

The first form of the sequence is displayed when you:

- Navigate to the sequence directly from the Sequence List.
- Copy/paste the URL+ID of the sequence directly into the browser's address bar.
- Point to the URL+ID from another webpage (refer to Embed a Form on a Website).

When navigating to an anonymous sequence from the Sequence List, you can navigate back to the Sequence List using the browser's back button.

Authenticated Sequences

Authenticated sequences are redirected to the Portal login page, for example:

http or https://<server>.<domain>:81/Login.aspx

The redirect code for the sequence is appended to the URL of the Portal login page, for example:

https://<server->.<-domain>:81/Lo-gin.as-px?ReturnUrl-=%2f%3fwa%3-dwsignin1.0%26wtrealm%3d-https%253a%252f%252f<server>.<domain>%253a9003%252f%26wct%3d2017-04-27T17%253a16%253a59Z%26wreply%3d-

https%253a%252f%252f<server-
>.<-
domain>%253a9003%252f%26w-
ctx%3drm%253d0%2526id%253d-
pass-
ive%2526ru%253d%2523%252-
fren-
der-
er%252f502%26whr%3d-
https%253a%252f%252f<server-
>.<-
domain>%253a9003%252f%26Ap-
pType%3dRenderer&wa=wsignin1.0&wtrealm=https%3a%2f%2f<server>.<domain>%3a9003%2f&wct=2017-
04-
27T17%3a16%3a59Z&wreply-
=https%3a%2f%2f<serve-
r>.<-
domain>%3a9003%2f&w-
ctx-
=rm%3d0%26id%3-
dpass-
ive%26ru%3d%23%2-
frenderer%2f502&whr=https%3a%2f%2f<server>.<domain>%3a9003%2f&AppType=Renderer

The Portal login page is displayed when you:

- Navigate to the sequence directly from the Sequence List
- Ccopy/paste the redirect code directly into the browser's address bar
- Point to the redirect from another webpage (refer to Embed a Form on a Website).

The first form of the sequence is displayed after logging in to Portal.

When navigating to an authenticated sequence from the Sequence List, you cannot navigate back to the Sequence List using the browser's back button.

Default Navigation Paths within Sequences

Scenario	Anonymous Sequence	Authenticated Sequence
The user completes a sequence.	The sequence ends on the confirmation page. The submitted data is saved if the workflow contains appropriate activities.	

Scenario	Anonymous Sequence	Authenticated Sequence
The user launches a previously completed sequence.	The first form of the sequence is displayed. This is a clean sequence, i.e., the previously entered data is not displayed.	
The user exits a sequence before completing it and returns to the sequence.	If a user has closed the browser, the first form of the sequence is displayed. This is a clean sequence, i.e., the previously entered data is not displayed. The durable instance becomes an orphan and will never be used again, because a new one will be created. However, in Forms Builder 3.5, if the user has never closed the browser and just gone to another website, user information cached in the browser may still be available to look up a durable instance, and coming back to the sequence in this case, the sequence will start up where it left off, up to a limit where anonymous pseudo-authentication expires or is replaced by other authentication information.	If the user stays logged in, the sequence continues at the form where the user previously left the sequence. If the user logs out, the Portal login page is displayed. The sequence starts from the form where user logged out.
The user logs out before completing the sequence and logs in again.	– NA –	The Portal login page is displayed. The sequence starts from the form where user logged out.

- The default paths described above can be changed by the workflow for a sequence.
- The placement of SaveEntity<> activities in the workflow determines when data will be saved within a sequence. Data can be saved at every Next transition or at the end of the sequence. For more information, refer to Create, Get, and Save Entity Activities.

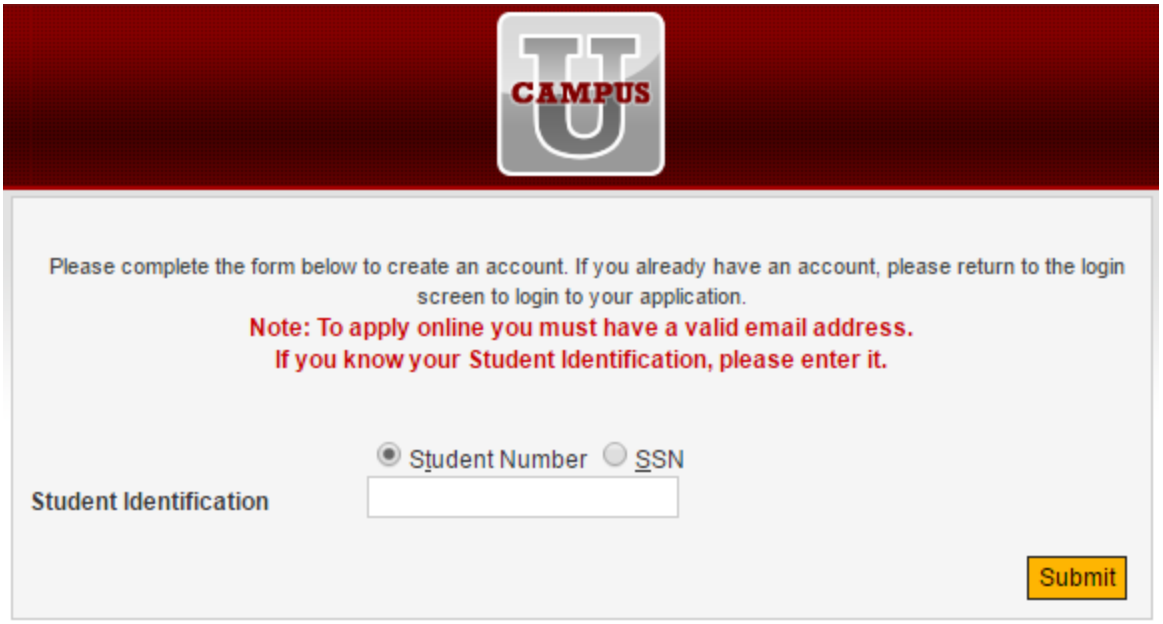
Renderer Authentication

When Forms Builder is integrated with Anthology Student, an end user logging in to an institution's Portal is authenticated through the Portal Security Token Service (STS). Accounts are created via the STS and can be customized using the Portal Configuration Tool.

The Portal security service also authenticates users accessing a form sequence when the sequence is non-anonymous (refer to Sequence List). End users do not need to log in to a form; however, if they are logged in via another sequence or via Portal, they will have an option to log out.

To create a Portal user account:

1. Type the **Portal URL** in the address bar of your browser. The login page is displayed.
2. Select the **Student Portal** link. The My Campus Login page is displayed.
3. Select **Create Account**.
4. Enter the **Student Identification** (if known) and select **Submit**.



5. Enter your details in the displayed page to create a new account. All fields marked with an asterisk are required.

When applicants or students create new accounts via the CMCPortalSTS, a `wpUserId` (web portal user Id) is generated for them alongside the User Name they choose. The `wpUserId` is linked to the `SyStudent` record in the Anthology Student database. The wpUserRelation needs to be created through a workflow. For more information, refer to Link a Portal Account to a Student Record.

If a staff sequence is accessed via cloud services (Azure), you must include a LookupUser activity with User-Type=Staff in the workflow to ensure proper authentication and authorization for the Staff role. For more information, refer to Azure AD Authentication.

Renderer Connection Strings

Forms Builder is installed with default connection strings for the database(s) where the Forms Builder and Workflow tables are created. The location of the Forms Builder and Workflow tables depends on the deployment configuration.

When Forms Builder is deployed with:

- Anthology Student and CampusNexus CRM, the Forms Builder and Workflow tables are in the Student database.
- Anthology Student only, the Forms Builder and Workflow tables are in Student database.
- CampusNexus CRM only, the Forms Builder and Workflow tables are in CRM database (`tlmain`).

In Forms Builder deployments with CRM and Student, where the default connection strings are always set to connect to the Student database, there was a need for a CRM connection string to:

- Query the CRM database via workflows
- Perform any additional logic and duplicate check operation via SQL scripts.

To accommodate these types of CRM database queries, the "CrmConnection" string was added to the Renderer web.config file in Forms Builder 3.6.

Excerpt of Renderer web.config in a deployment with Anthology Student and CampusNexus CRM

```
<connectionStrings>
  <add name="WorkflowDurableInstancingConnection" connectionString="Data Source=<DB Server>;Initial Catalog=<Student DB>;Integrated Security=True;Pooling=True;MultipleActiveResultSets=True;Application Name=FormsBuilder;" />
  <add name="FormsBuilderModel" providerName="System.Data.SqlClient" connectionString="Data Source=<DB Server>;initial catalog=<Student DB>;Integrated Security=SSPI;Persist Security Info=False;MultipleActiveResultSets=True" />
  <add name="dbConnection" providerName="System.Data.SqlClient" connectionString="Data Source=<DB Server>;initial catalog=<Student DB>;Integrated Security=SSPI;Persist Security Info=False;MultipleActiveResultSets=True" />
  <add name="PortalConnection" providerName="System.Data.SqlClient" connectionString="Data Source=<DB Server>;initial catalog=<Student Portal DB>;Integrated Security=SSPI;Persist Security Info=False;MultipleActiveResultSets=True" />
  <add name="CrmConnection" providerName="System.Data.SqlClient" connectionString="Data Source=<DB Server>\inst1;initial catalog=CRM DB;Integrated Security=SSPI;Persist Security Info=False;MultipleActiveResultSets=True" />
 </connectionStrings>
```

The ExecuteDataReader, ExecuteNonQuery, and ExecuteQuery activities retrieve connection strings from the web.config of the product that triggers workflow execution. When Forms Builder is upgraded, the "CrmCon-

nection" string will be added to the web.config of Forms Renderer. If you have created workflows with any of the "Execute" activities, ensure that connection strings in the activities match the updated web.config.

Renderer Media Variables

The following variables are available to a workflow (as arguments) or to external JavaScript by vmModelsRef. These variables may be useful to JavaScript or workflows that have to make decisions about the HTML to render for different size screens.

All the following media variables are true when the condition is satisfied on the current device's screen.

Renderer Media Variables

Variable	Description
`xsMedia`	Maximum width is 599px
`gtxsMedia`	Minimum width is 600px
`smMedia`	Minimum width is 600px and maximum width is 959px
`gtsmMedia`	Minimum width is 960px
`mdMedia`	Minimum width is 960px and maximum width is 1279px
`gtmdMedia`	Minimum width is 1280px
`lgMedia`	Minimum width is 1280px and maximum width is 1919px
`gtlgMedia`	Minimum width is 1920px
`xlMedia`	Minimum width is 1920px
`landscapeMedia`	Media is in landscape mode
`portraitMedia`	Media is in portrait mode
`printMedia`	Media is print media
`isMobile`	Media display is a mobile device *Example:* If a condition in JavaScript sets different classes or different HTML, the attribute `vmModelsRef.isMobile` would be true on a phone.

When these variables have a suffix of `Neg`, they are the opposite, that is to say, `xsMedia` is true, means `xsMediaNeg` is false.

Renderer URL Query Parameter

You can pass a query parameter when the Renderer URL is invoked. You can pass the query parameter into the workflow for a sequence and use the query parameter in various workflow activities. For example, the query parameter can be:

- Written to a log
- Assigned to a query
- Assigned to an entity

Syntax

The syntax for the Renderer URL query parameter is as follows:

https://<server>.<domain>:<port>/#/renderer/<sequencenumber>**?<parameter1>=<value1>&<parameter2>=<value2>**

- A query parameter is a name-value pair.
- A question mark (?) precedes the first query parameter.
- Multiple query parameters are separated by the ampersand (&).
- URL encoding applies to the query parameter string:
  - URL encoding is required for any characters outside of the ASCII character set.
  - URL encoding replaces unsafe ASCII characters with a "%" followed by two hexadecimal digits.
  - URLs cannot contain spaces. URL encoding normally replaces a space with a plus (+) sign or with %20.

For more information, refer to HTML URL Encoding Reference and URL Decode and Encode tool.

Pass a URL Query Parameter to a Workflow

You can pass a Renderer URL query parameter to the workflow of any form sequence.

⚠ You must capture the Renderer URL query parameter in the Entry state of the first form in a sequence.

Example

The Renderer URL with query parameters is as follows:

https://<server>.<domain name>:<port>/#/Renderer/<workflow definition Id or sequence Id>/?query parameters

Our example:

https://<server>.<domain>:<port>/#/Renderer/540?**Firstname=John&Lastname=Doe**

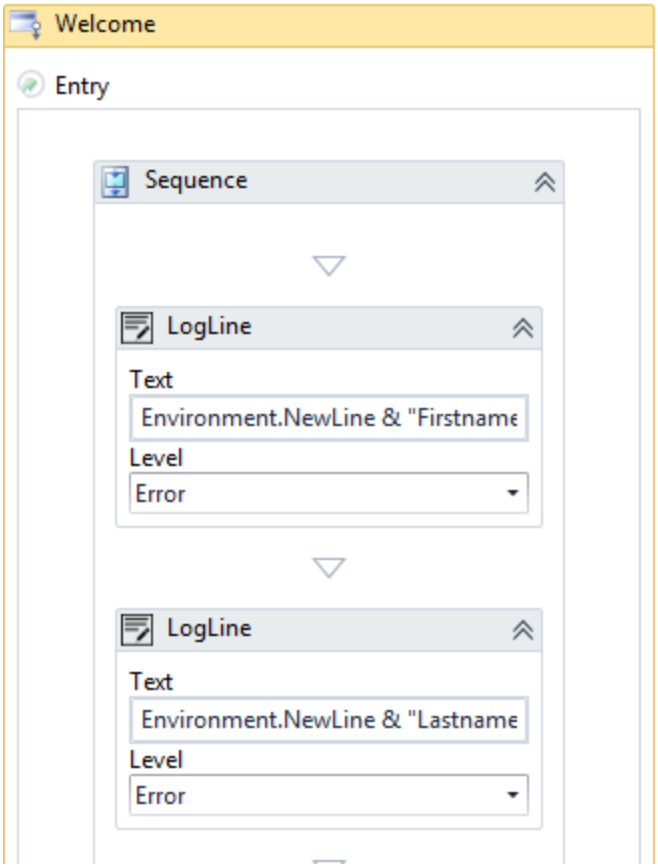
The Welcome form in the workflow for sequence 540 contains LogLine activities that write the Firstname and Lastname values to a log.

```
Environment.NewLine & "Firstname=" & formInstance.QueryParams.DataDictionary("First-
name").ToString
```

```
Environment.NewLine & "Lastname=" & formInstance.QueryParams.DataDictionary("Last-
name").ToString
```

We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

Note that "ToString" is required convert the parameter value to its string representation so that it is suitable for display.



These values are also available client-side.

vm.queryParams.Firstname

vm.queryParams.Lastname

They are also available in external JavaScript in an HTML component:

vmQueryParamsRef.Firstname

vmQueryParamsRef.Lastname

If you changed the NLog configuration as recommended in Log Files, the Level will be "Information" ("Info" in NLog).

Pass "addonQueryParams" via the URL

Another option to pass values from the URL is through the "addonQueryParams" key which is a built-in key name (unlike the previous key value pairs which define both the key and the value). You can use this method client-side for operations such as:

- Setting a title or image in HTML
- Providing a value for JavaScript to use
- Providing a value for a style sheet

Example

https://<server>.<domain name>:<port number>/#/Renderer/<workflowdefinitionId or sequence identifier goes here>/**This+is+arbitrary+information**?firstname=john

You can access the text between the slash (/) and the question mark (?) in the workflow with the following argument:

formInstance.QueryParams.DataDictionary("addonQueryParams").ToString()

Forms Builder 3.3 and later adds client-side access for both the URL Query Parameter key value pairs and the built-in addonQueryParams key.

Query parameters could be used as a binding value with the following:

vm.queryParams.addonQueryParams

vm.queryParams.firstName

Both are also available with the `vmQueryParamsRef` JavaScript global variable. So, the following would be a client-side way to access the examples above:

vmQueryParamsRef.firstName (or vmQueryParamsRef["firstName"])

vmQueryParamsRef.addonQueryParams (or vmQueryParmsRef["addonQueryParams"])

In other words, using the key as a dot suffix gives access to the value.

This could be useful for client-side JavaScript in an HTML component (refer to HTML component).

It also could be used to decode a value before the workflow receives it. If the value of a key, `myEncodedValue` (or the built-in `addonQueryParams` key) happened to be the following URL encoded text:

"This+is+>&2044"

Then you could decode this with some JavaScript so that the workflow would receive this decoded value:

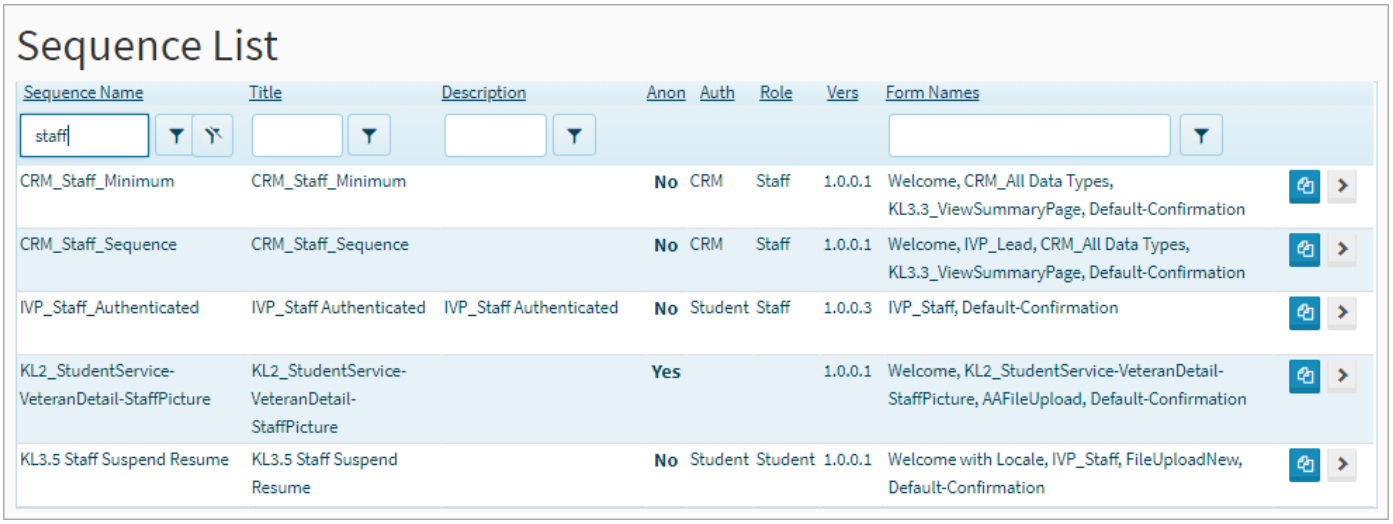
“This is > 44”

The following JavaScript in an HTML component would decode and assign this to a new argument, `vm.models.myDecodedValue`:

```
<script type="text/javascript">
vmModelsRef.myDecodedValue = decodeURI(vmQueryParamsRef.myEncodedValue);
</script>
```

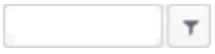
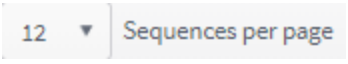
Sequence List

The Sequence List displays a grid listing the sequences by name, title, description, version, and forms contained in each sequence. The items in the Sequence List are generated by the Form Designer.



A server error occurs when the Sequence List is opened in the same browser as Form Designer or Sequence Designer. To avoid this error, use different browsers, e.g., open the Sequence List in Chrome while Form Designer is open in IE.

You can perform the following actions in the Sequence List:

- Select a column header to sort a column in ascending or descending order.
- Adjust the column widths by dragging the column separators.
- Use the search and filter controls  to find items within a column.
- Use the page navigation controls  to view different pages of the grid.
- Use the drop-down list  to select the number of sequences displayed per page.
- Select  to copy the URL of the sequence to clipboard or select  to view a sequence in the current browser.

Sequence List Grid

Column	Description
Sequence Name	Name of the sequence assigned in Sequence Designer.
Title	Title of the sequence assigned in Sequence Designer.
Description	Description of the sequence assigned in Sequence Designer.
Anon	Anonymous property setting for the sequence. The Anonymous property setting determines if a user will be authenticated before accessing the sequence in Renderer (refer to Renderer Authentication) • If Anonymous is 'true', the user will not be authenticated. • If Anonymous is 'false' (default), the user will be authenticated.
Auth	Indicates whether a sequence uses CampusNexus CRM or Anthology Student authentication.
Role	Indicates whether a sequence is designed for a Staff or Student role. The Student role is also used for CampusNexus CRM Contacts. For more information, refer to Role.
Vers	Version of the sequence. For the initial version of a sequence, the version is set to 1.0.0.1. For any subsequent updates of the sequence, the version is incremented, for example from 1.0.0.1 to 1.0.0.2, and so on. The version is updated each time the associated workflow is published. The Version column will show "?.?.?.?" if an invalid workflow version with NULL values for the Major, Minor, Build, and Revision fields is found in the WorkflowDefinitionversion table.
Form Names	Names of all forms in a sequence. The form names are separated by commas.

Vanity URL

Forms Renderer can be deployed with a Vanity URL, a descriptive or branded URL that redirects you from one location to another. An example of a vanity URL is `https://<institution>.campusnexus.cloud/#/sequencelist`, where `<institution>` is a placeholder for the customized URL selected by the institution.

Forms Renderer Standard URL and Vanity URL(s) operate on all environments such as Cloud 1.X, 2.X and On-premise independently or in combination with Standard URL. Standard URL and Vanity URLs are added in Admin Console tool for multiple tenants, single tenants. For on-premise environments, these URLs are added in web.config file.

Vanity URLs are not applicable to Form Designer.

Important: All persisted instances for authenticated sequences must be cleared before changing the Renderer URL to a vanity URL.

If you are a Cloud 1.2 customer and you want to implement a vanity URL for your instance of Renderer, please contact Anthology Inc. at https://support.campusmgmt.com/ and submit a **Service Request (CSR)**.

If you are using Azure AD, Anthology Inc.will need to update or create new App Registrations with vanity URLs for you.

Use Cases

This section provides examples of forms, workflows, and sequences built with Forms Builder.

Credit Card Payment Form

This topic shows how a form sequence for credit card payments could be designed. It also details the associated workflow and shows the rendered sequence.

> ⚠ The screen captures in this topic show an earlier Forms Builder version. While the UI has been updated, the basic functionality is unchanged.

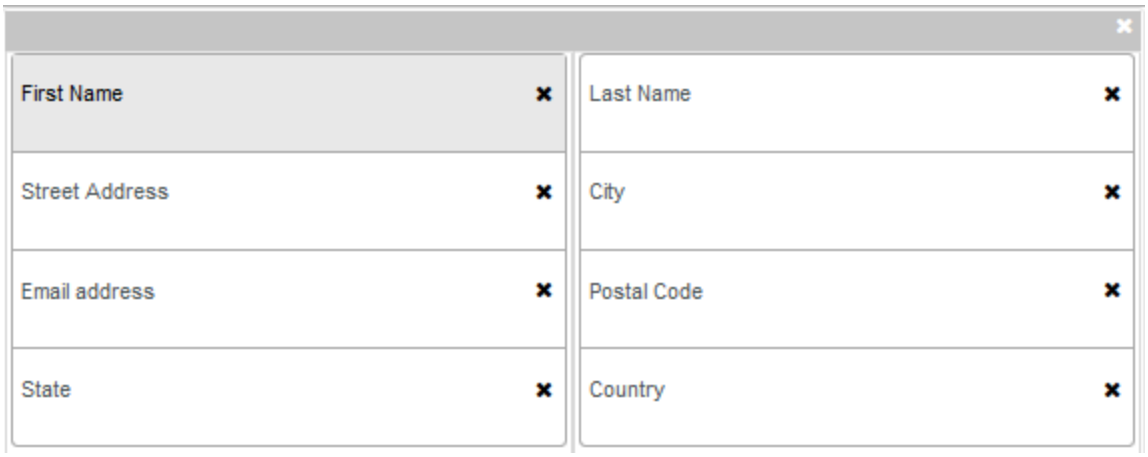
Prerequisites

Refer to Credit Card Payment component.

Create the Form Sequence

For a credit card payment sequence, we recommend creating one form to gather the payment data (e.g., name, address, state, country, etc.), another form for the Credit Card Payment component, and a confirmation form that is displayed when the user returns from the PayPal site.

1. In Form Designer, create the layout of the form to gather the payment data (form 1).



2. Define the properties for the form fields. In this example, the form fields are bound to the **vm.-models.studentEntity**.

   First Name

Name	Value
Control Type	Text Box
Class	
Disabled	false
Id	StudentEntityFirstName
Label	First Name
Max Length	25
Maximum Value	
Min Length	1
Minimum Value	
Model	vm.models.studentEntity.FirstName
Placeholder	
Read-only	false
Required	true
Required Message	
Tooltip	First Name
Type	text
Visible	true

Last Name

Name	Value
Control Type	Text Box
Class	
Disabled	false
Id	StudentEntityLastName
Label	Last Name
Max Length	25
Maximum Value	
Min Length	1
Minimum Value	
Model	vm.models.studentEntity.LastName
Placeholder	
Read-only	false
Required	true
Required Message	
Tooltip	Last Name
Type	text
Visible	true

Street Address

Name	Value
Control Type	Text Box
Class	
Disabled	false
Id	StudentEntityStreetAddress
Label	Street Address
Max Length	255
Maximum Value	
Min Length	1
Minimum Value	
Model	vm.models.studentEntity.StreetAddress
Placeholder	
Read-only	false
Required	false
Required Message	
Tooltip	Street Address
Type	text
Visible	true

City

Name	Value
Control Type	Text Box
Class	
Disabled	false
Id	StudentEntityCity
Label	City
Max Length	25
Maximum Value	
Min Length	1
Minimum Value	
Model	vm.models.studentEntity.City
Placeholder	
Read-only	false
Required	false
Required Message	
Tooltip	City
Type	text
Visible	true

Email address

Name	Value
Control Type	Text Box
Class	
Disabled	false
Id	StudentEntityEmailAddress
Label	Email address
Max Length	50
Maximum Value	
Min Length	1
Minimum Value	
Model	vm.models.studentEntity.EmailAddress
Placeholder	
Read-only	false
Required	false
Required Message	
Tooltip	Email address
Type	text
Visible	true

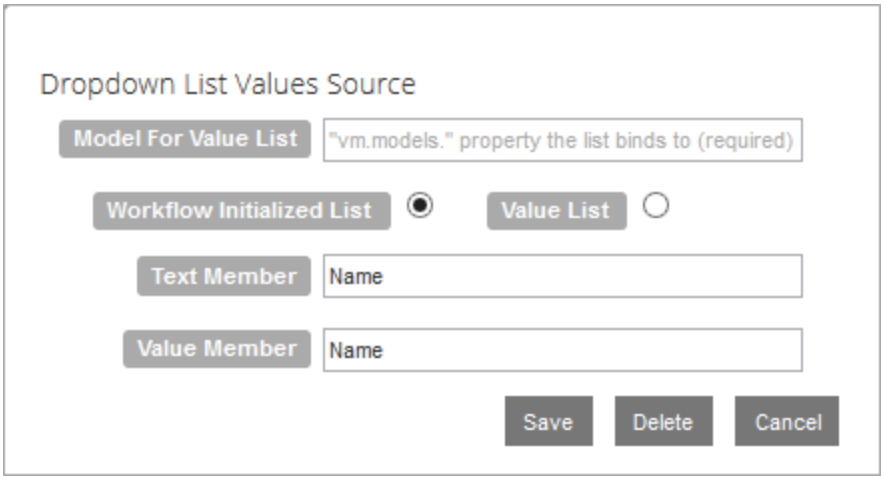
Postal Code

Name	Value
Control Type	Text Box
Class	
Disabled	false
Id	StudentEntityPostalCode
Label	Postal Code
Max Length	10
Maximum Value	
Min Length	1
Minimum Value	
Model	vm.models.studentEntity.PostalCode
Placeholder	
Read-only	false
Required	false
Required Message	
Tooltip	Postal Code
Type	text
Visible	true

State

Name	Value
Control Type	Drop-down List
Class	
Disabled	false
Id	StudentEntityState
Label	State
Lookup Display Member	Name
Lookup Query	UsaStates?$select=Code,Name,Id&$filter=IsActive eq true
Lookup Sort Member	Code
Lookup Value Member	Name
Model	vm.models.studentEntity.State
Option Label	<Select>
Page Size	100
Product	Student
Required	false
Required Message	
Server Filtering	false
Tooltip	State
Value List	Edit...
Visible	true

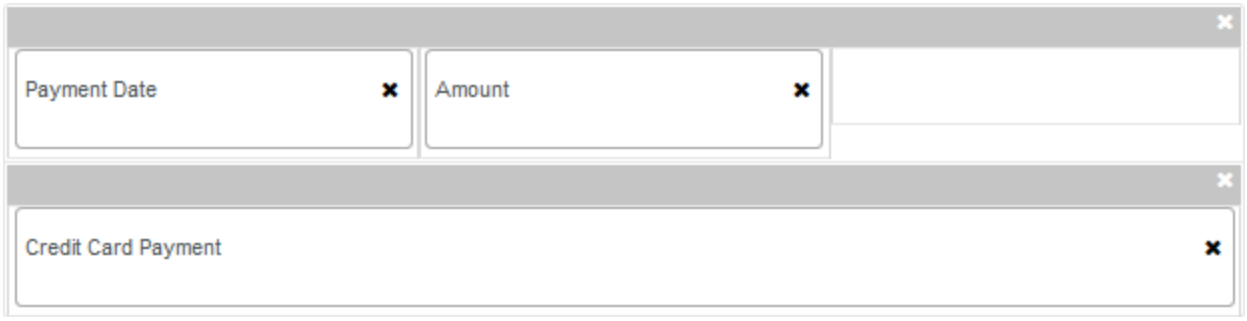
Note that the Value List in the properties for the State field is defined as a "Workflow Initialized List".



Country

Name	Value
Control Type	Single-select Search
Class	
Disabled	false
Grid Columns	[{"field": "Name", "title": "Name"}]
Id	StudentEntityCountryId
Label	Country
Lookup Display Member	Name
Lookup Query	Countries?$select=Code,Name,Id&$filter=IsActive eq true
Lookup Value Member	Id
Model	vm.models.studentEntity.CountryId
Product	Student
Required	false
Required Message	
Tooltip	Country
Visible	true

3. Save the form.
4. Create the layout of the form with the Credit Card Payment component (form 2). This form contains the "Make Payment" button that links to the payment form on the PayPal site.



5. Define the properties and bindings for the form fields. In this example, the Payment Date and Amount fields are bound to the **vm.models.depositEntity**. The property settings for Credit Card Payment component contain bindings for the **vm.models.studentEntity**, the **vm.models.depositEntity**, and **vm.models.countryName**.

Payment Date

Name	Value
Control Type	Date Picker
Class	
Disable Input Text	true
Disabled	false
Format	d
Id	DepositEntityDepositReceivedDate
Label	Payment Date
Maximum Value	
Minimum Value	
Model	vm.models.depositEntity.DepositReceivedDate
Read-only	false
Required	true
Required Message	
Tooltip	Date deposit was received
Visible	true

Amount

Name	Value
Control Type	Numeric Text Box
Class	
Decimals	
Disabled	false
Format	c2
Id	DepositEntityAmount
Label	Amount
Max Length	50
Maximum Value	
Minimum Value	
Model	vm.models.depositEntity.Amount
Placeholder	
Read-only	false
Required	true
Required Message	
Restrict Decimals	false
Round	false
Step	1
Tooltip	Deposit amount
Visible	true

Credit Card Payment

Name	Value
Control Type	Credit Card Payment
Class	
Id	82918206-f7a2-26ff-8130-eae40b5fc11a
Payment Address	vm.models.studentEntity.StreetAddress
Payment Amount	vm.models.depositEntity.Amount
Payment City	vm.models.studentEntity.City
Payment Country	vm.models.countryName
Payment Email	vm.models.studentEntity.EmailAddress
Payment Error Message	Must make a payment to continue.
Payment Firstname	vm.models.studentEntity.FirstName
Payment Instructions	Please make a payment by clicking on the link below and proceeding to the payment site.
Payment Lastname	vm.models.studentEntity.LastName
Payment Link Class	btn btn-primary
Payment Link Text	Make Payment
Payment State	vm.models.studentEntity.State
Payment Warning Message	WARNING: To ensure correct processing, you must click on the Return To Merchant Website link after making the payment on the external Payment Processor page.
Payment Zip	vm.models.studentEntity.PostalCode

The example above shows values for all properties (except Class), however, it is not necessary to specify values for all properties. The required properties for any credit card payment are typically:

- Payment Amount
- Payment Firstname
- Payment Lastname
- Payment Zip

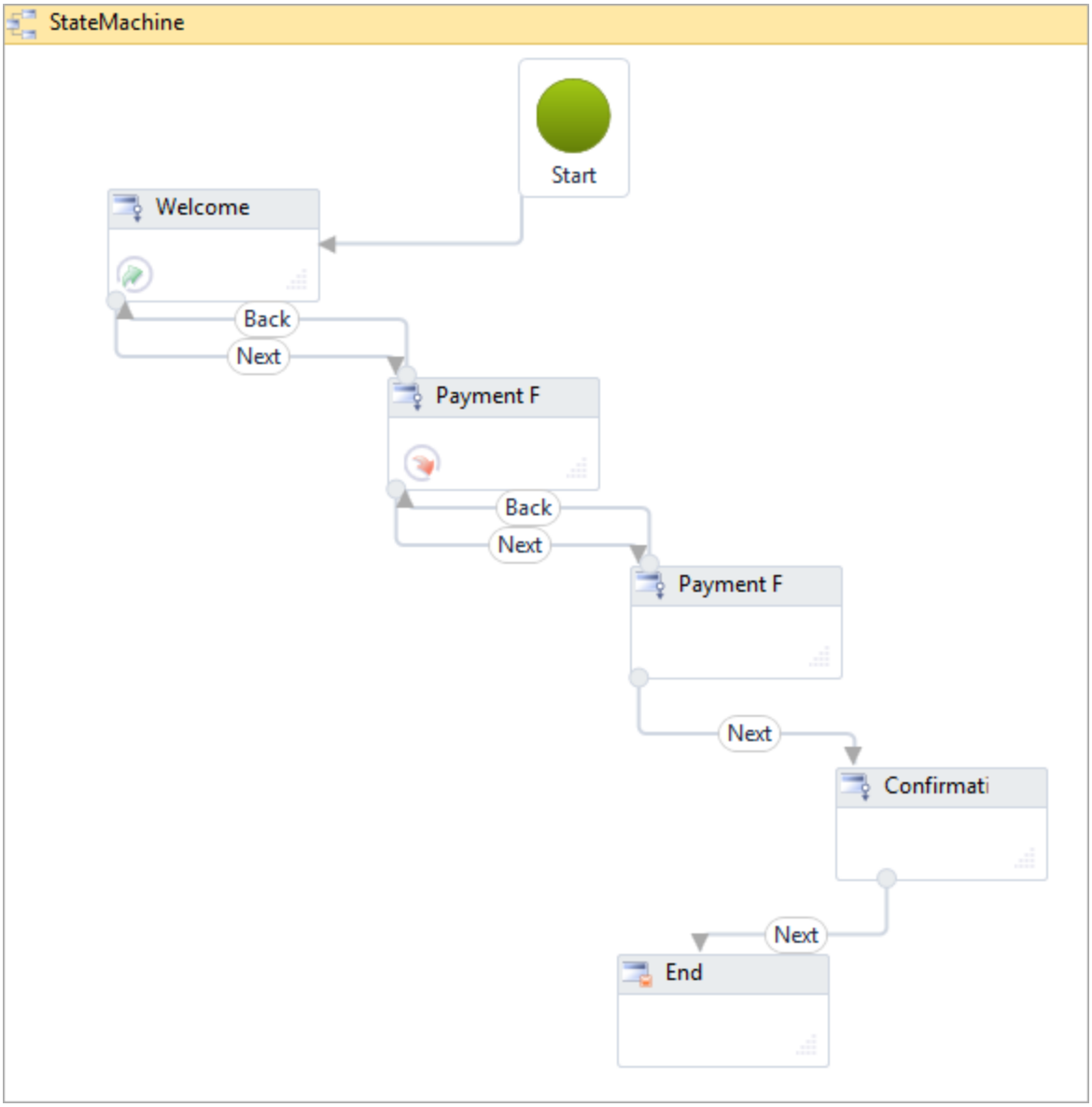
For more detailed information about the properties, refer to the Credit Card Payment component.

6. Save the form.
7. Create a confirmation form that contains an HTML component with property settings similar to the example below (form 3). Note that the syntax **{{vm.models.depositEntity.Amount}}** allows the form to display the value of the Payment Amount property specified in the Credit Card Payment component

above.

Name	Value
Control Type	HTML
Class	
HTML	Thanks for your payment of $ {{vm.models.depositEntity.Amount}}

8. Save the form.
9. In Sequence Designer, create and save an authenticated sequence that contains a welcome form and the 3 forms built above.
10. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.



Modify the Workflow

1. Create **arguments** to match the vm.models bindings defined in Credit Card Payment component above.

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*
studentEntity	In/Out	StudentEntity	*Default value not supported*
depositEntity	In/Out	DepositEntity	*Default value not supported*
countryName	In/Out	String	*Default value not supported*

To prepopulate fields in the payment form, create additional arguments as needed and add corresponding Assign activities in the entry state of the first form in the workflow.

2. Create **variables** as needed. Our example creates a variable named "CountryLookup" that will be used to populate the Single-select Search component used for the Country field.

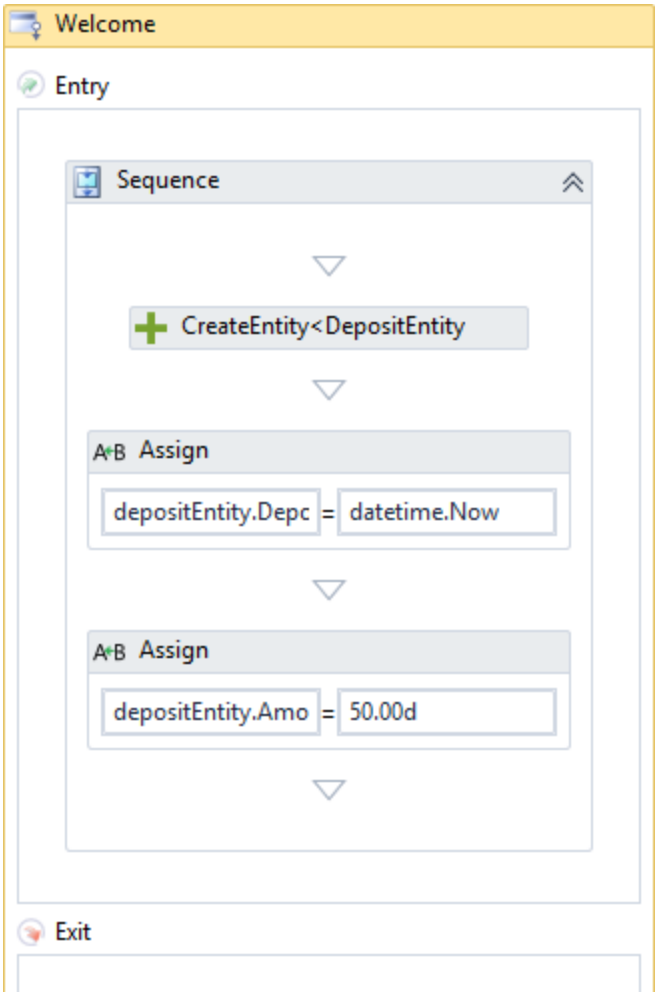
Name	Variable type	Scope	Default
renderedFormImage	String	StateMachine	*Enter a VB expression*
CountryLookup	ReferenceItem	StateMachine	*Enter a VB expression*

3. In the Entry section of the Welcome form/state, add the activities needed to prepopulate the payment form.

   Our example creates the **DepositEntity** and uses Assign activities for the following values:

   depositEntity.DepositReceivedDate = datetime.Now

   depositEntity.Amount = 50.00d



To prepopulate fields in the payment form based on existing database records, place a LookupUser activity in the entry section of the Welcome form.

4. Optionally, add a LogLine activity below the Assign activities to capture the Transaction Id in the log. Specify the following properties:

   Text: **"FormInstance.paymentInfo.transactionId: "&formInstance.PaymentInfo.TransactionId**

   Level: **Information**

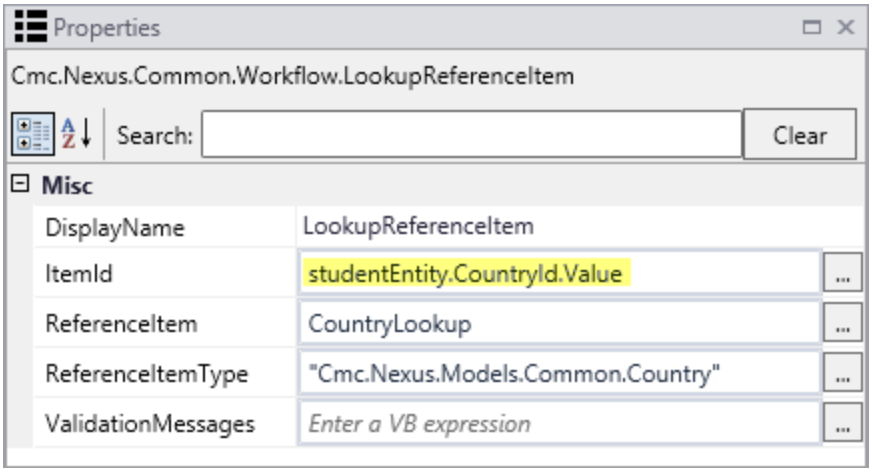
5. Add a **Sequence** activity to the Exit section of the form that contains the Country field (form 1). The Sequence will hold the workflow activities needed to populate the Single-select Search component for the Country field.

6. Drag and **If** activity into the Sequence and specify the following Condition: **studentEntity.CountryId.HasValue**

7. Drag a **LookupReferenceItem** activity into the Then branch. Specify the following properties:

   Reference Item Type = **Country**

   Reference Item Id = **studentEntity.CountryId.Value**



   The CountryId integer value from the Anthology Student database needs to be converted to a string (text) to pass it to the PayPal site. The highlighted value is populated from the Reference Item Type drop-down list in the LookupReferenceItem activity (or from the Anthology Student database if a getEntity > StudentEntity activity is used to populate the bindings). The conversion from integer to string is accomplished with the Assign activity that follows.

   CountryName in assignment statement is the in/out argument bound to Country property in CreditCard component

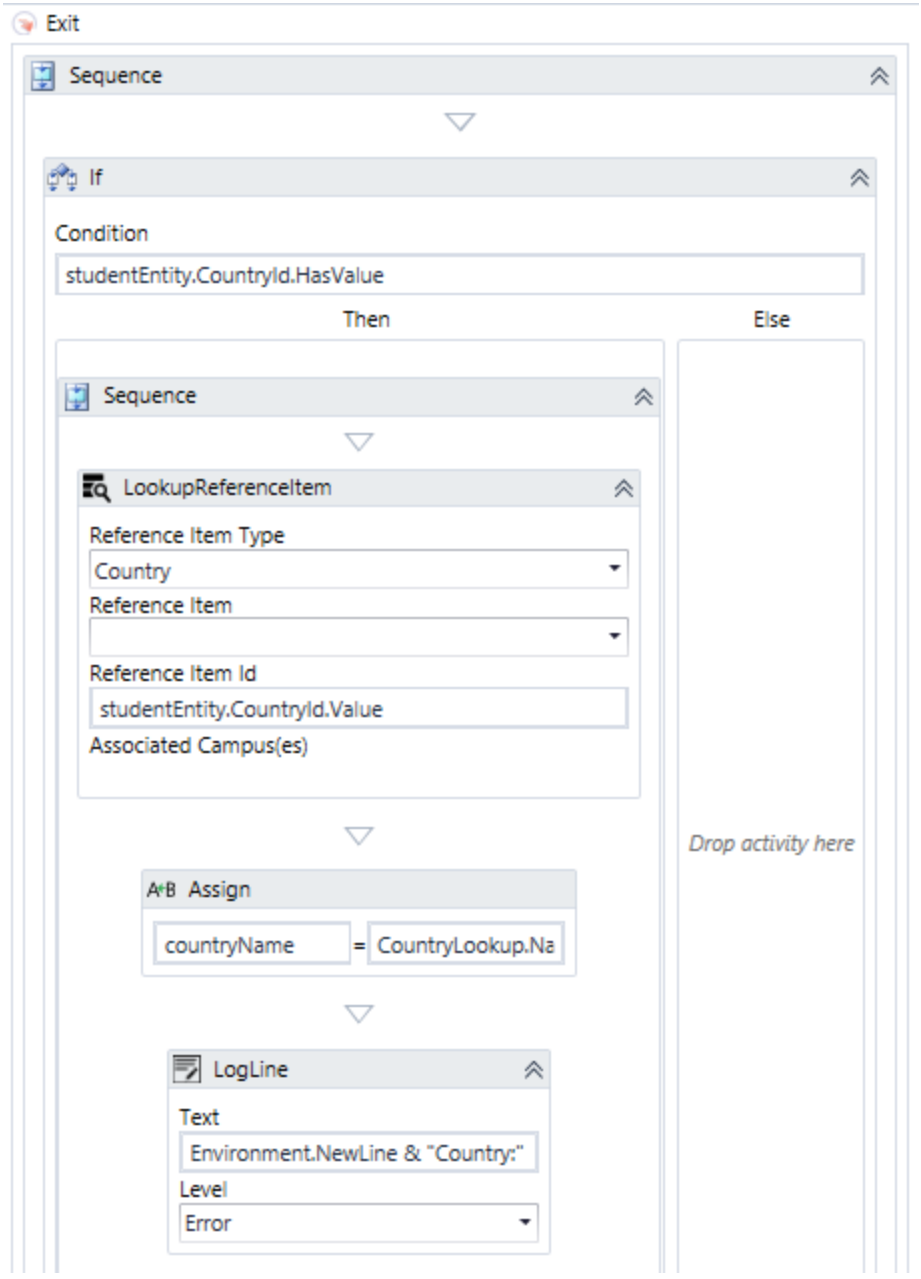
8. Drag an Assign activity into the Then branch. Specify the following properties:

   countryName = **CountryLookup.Name.Trim**

9. Drag a Logline activity into the Then branch. Specify the following properties:

Text: **Environment.NewLine & "Country:" & countryName & " State:" & studentEntity.State**

Level: **Information**



10. Double-click the **Next** transition below the form that contains the Credit Card Payment Component (form 2).
11. In the trigger section of the Next transition, add a **Sequence** activity and insert a WaitForFormBookmark activity.
12. Add a VerifyCardPayment activity with the following properties:

PaymentAmount: **Amount**

PaymentTransactionId: **formInstance.PaymentInfo.TransactionId**

ValidationMessages: **formInstance.ValidationMessages**

13. Drag an **If** activity below the WaitForFormBookmark activity.
    a. In the Condition field, specify: **formInstance.ValidationMessages.HasErrors**
    b. In the Then branch, add a LogLine activity. Specify the following properties:

       Text: **"Verify payment result: " & formInstance.ValidationMessages(0).Message**

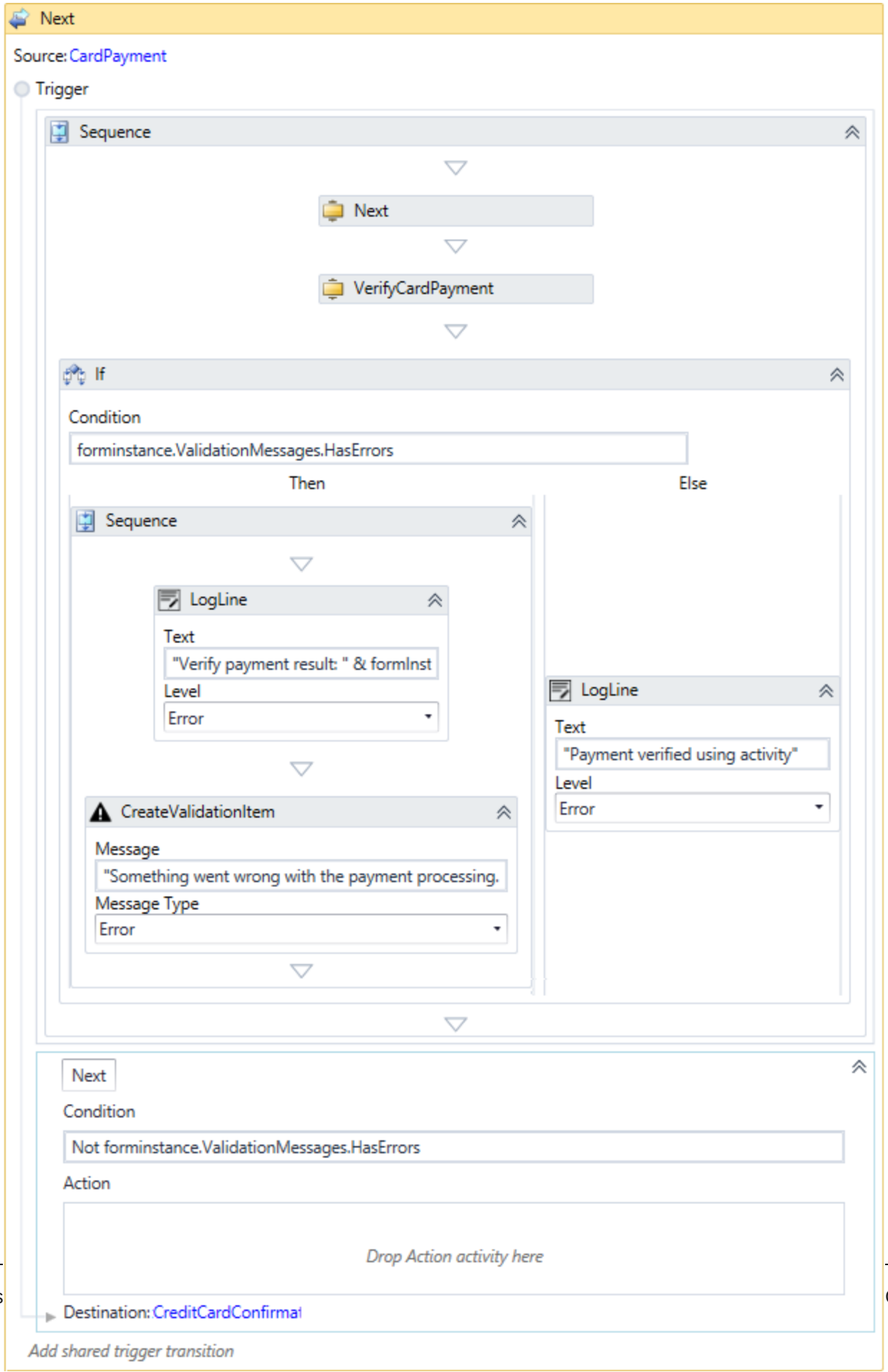
       Level: **Information**
    c. Below the LogLine, add CreateValidationItem activity with the following properties:

       Message: **"Something went wrong with the payment processing. Please contact customer service to confirm that your payment was processed correctly."**

       Message Type: **Error**
    d. In the Else branch, add a LogLine activity. Specify the following properties:

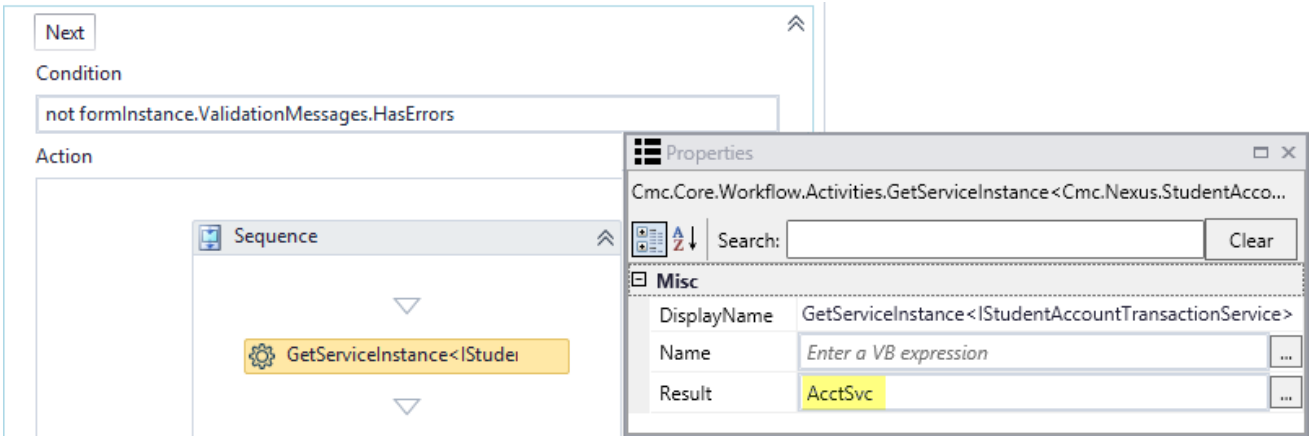
       Text: **"Payment verified using activity"**

       Level: **Information**
    e. In the Condition field of the Next transition, specify: **Not formInstance.ValidationMessages.HasErrors**


Next
Source: CardPayment
Trigger
Sequence
Next
VerifyCardPayment
If
Condition
forminstance.ValidationMessages.HasErrors
Then
Else
Sequence
LogLine
Text
"Verify payment result: " & formInst
Level
Error
CreateValidationItem
Message
"Something went wrong with the payment processing.
Message Type
Error
LogLine
Text
"Payment verified using activity"
Level
Error
Next
Condition
Not forminstance.ValidationMessages.HasErrors
Action
Drop Action activity here
Destination: CreditCardConfirmat
Add shared trigger transition

Use the following activities to post payments.

14. In the Action section of the previous activity, insert a sequence with a **GetServiceInstance<>** activity using the **<IStudentAccountTransactionService>**. Create a variable for the **Result** OutArgument.



The variable must be of type IStudentAccountTransactionService as shown below:

Name	Variable type	Scope	Default
acctSvc	Cmc.Nexus.StudentAccounts.Services.IStudentAccountTransactionService	StateMachine	*Enter a VB expression*

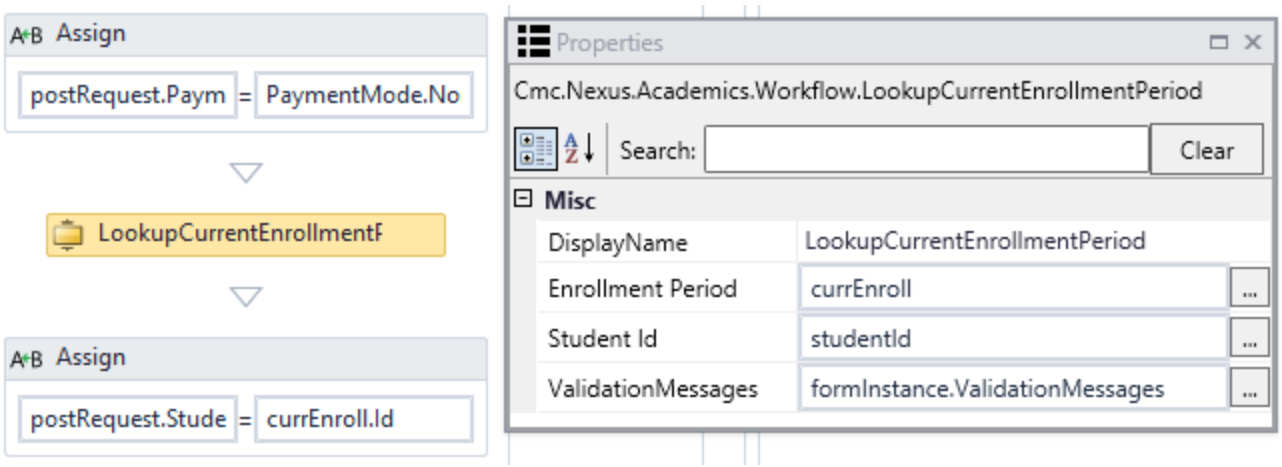
15. Create variables named **postRequest** and **postResponse**as shown below.

Name	Variable type	Scope	Default
postRequest	Cmc.Nexus.StudentAccounts.Services.PostAccountTransactionPaymentRequest	StateMachine	new PostAccountTransactionPaymentRequest
postResponse	Cmc.Nexus.StudentAccounts.Services.PostAccountTransactionPaymentResponse	StateMachine	*Enter a VB expression*

16. Below the GetServiceInstance activity, drop an **Assign** activity for each row in the following table and specify the indicated values:

"To" Field	Value
postRequest.StudentId	studentId
postRequest.TransactionAmount	depositEntity.Amount
postRequest.TransactionDate	depositEntity.DepositReceivedDate
postRequest.PaymentMode	PaymentMode.Normal

17. Insert a **LookupCurrentEnrollmentPeriod** activity below the Assign activities. Use the "studentId" variable as InArgument and the "currEnroll" variable as OutArgument for the LookupEnrollmentPeriod activity.



Define the "currEnroll" variable as shown below:

Name	Variable type	Scope	Default
currEnroll	Cmc.Nexus.Academics.Entities.StudentEnrollmentPeriodEntity	StateMachine	*Enter a VB expression*

18. Insert **Assign** activities below the LookupEnrollmentPeriod activity using the following attributes:

"To" Field	Value
postRequest.StudentEnrollmentPeriodId	currEnroll.Id
postResponse	AcctSvc.PostAccountTransactionPayment(PostRequest)

If cashiering is enabled, additional assignment statements will be needed.

19. (Optional) Insert a **LogLine** activity. Specify the following properties:

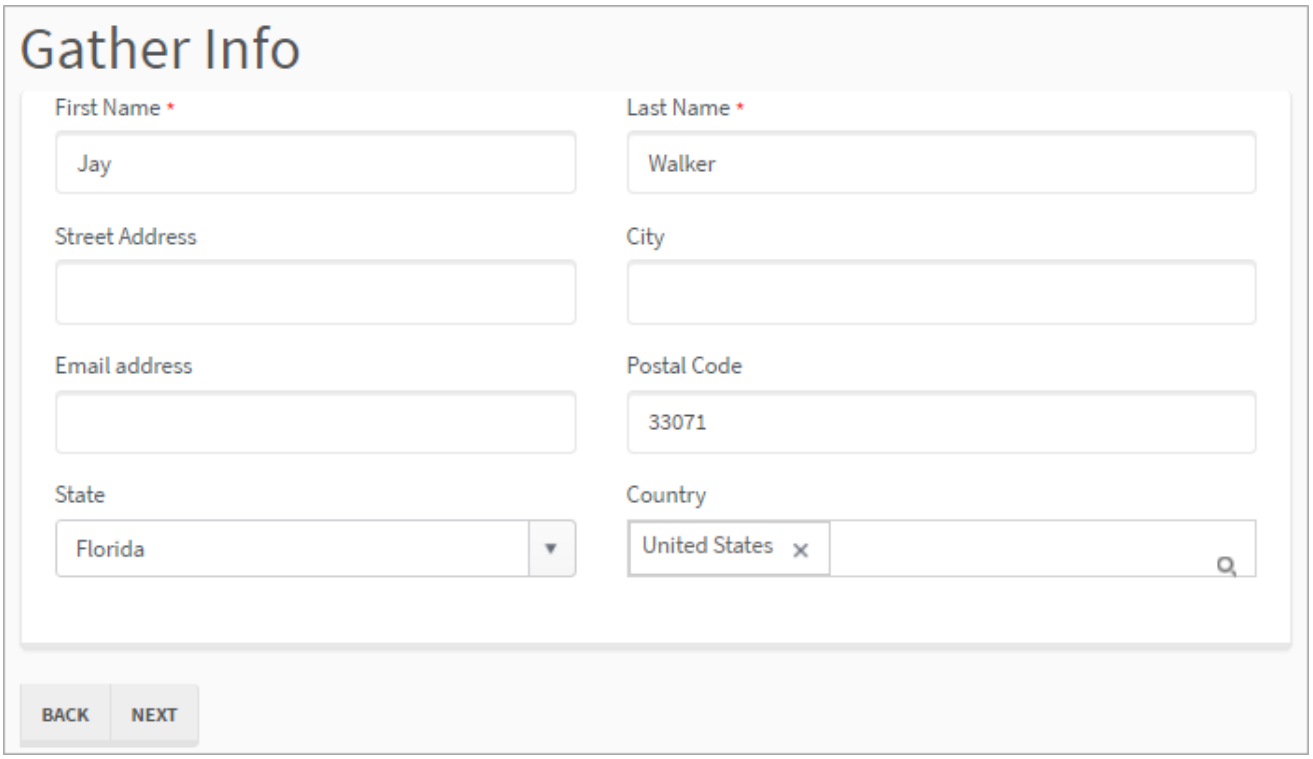
Text: **Newtonsoft.Json.JsonConvert.SerializeObject(postResponse,Newtonsoft.Json.Formatting.Indented)**

Level: **Information**

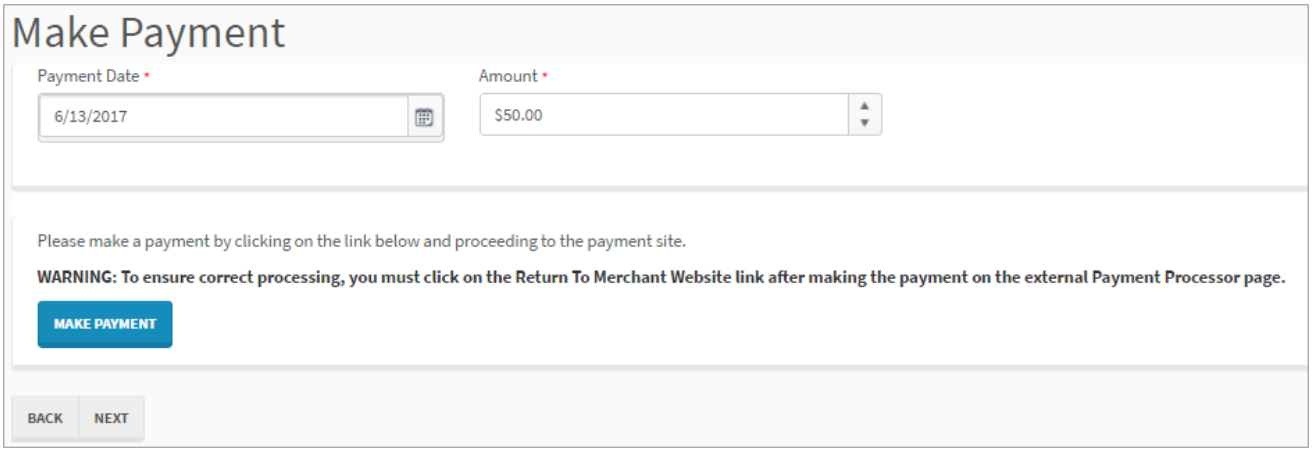
Publish and enable the workflow.

Test the Rendered Sequence

1. Access the sequence in the **Sequence List**.
2. Log in to **Portal**.
3. Complete the form that gathers the payment information (form 1) and select **Next**.

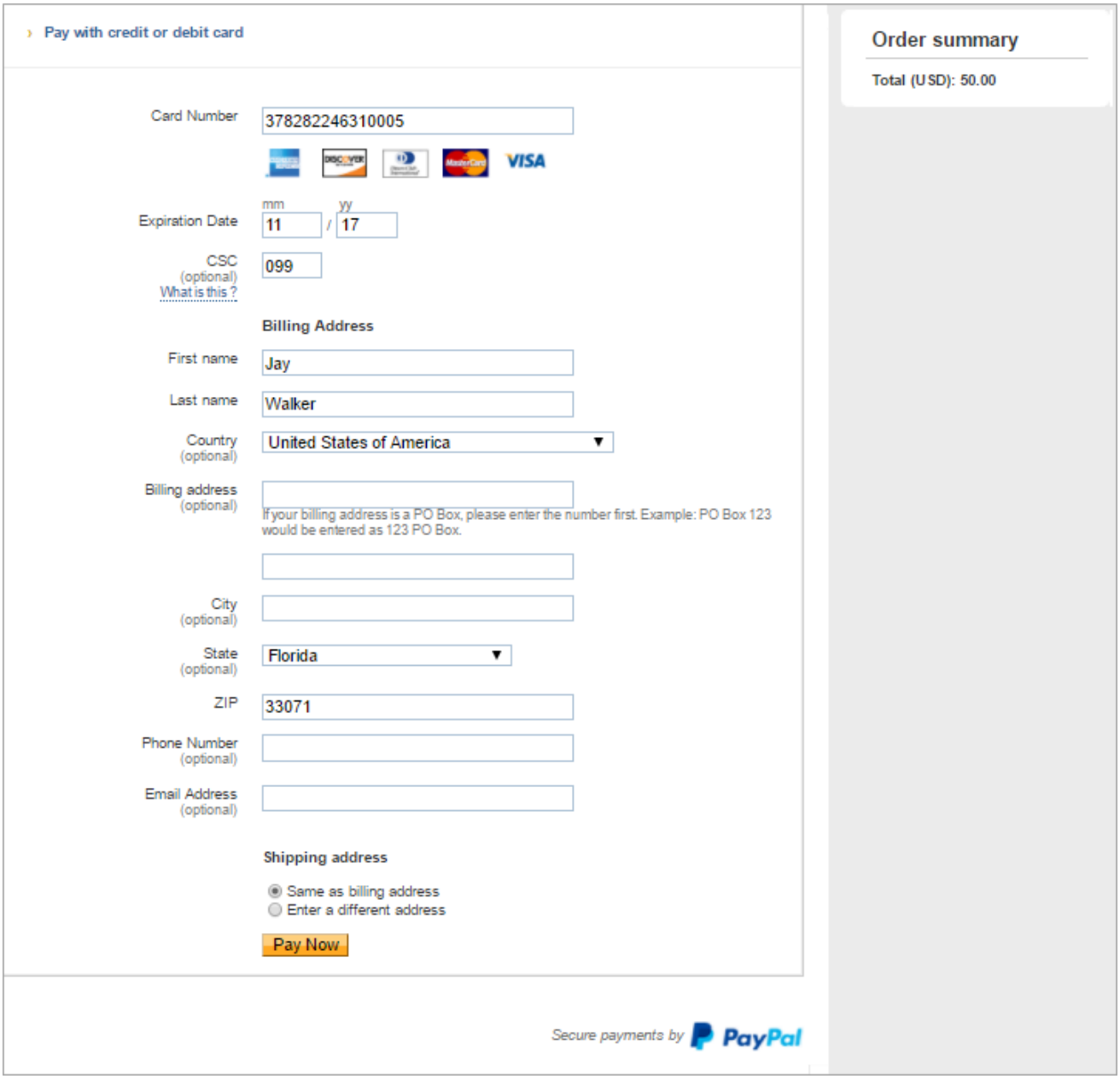


4. On the next form, note that the Payment Date and Amount fields are prepopulated based on the work-flow definition. Select the **Make Payment** button.



5. Complete the payment form on the PayPal site and select **Pay Now**.

   Refer to the following website to obtain credit card numbers for testing: https://developer.paypal.com/docs/classic/payflow/integration-guide/?mark=test%20card%20numbers#credit-card-numbers-for-testing

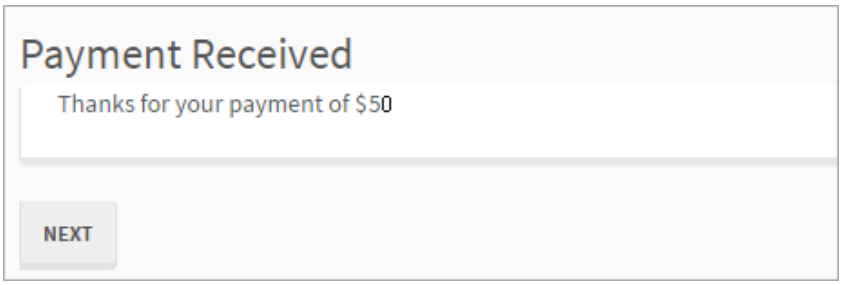


6. A confirmation page from PayPal is displayed. Select the **Return to Anthology Inc.** link (this is the customized return link configured in PayPal).



If a user loses the Internet connection while on the PayPal receipt page and logs back in, the user will be returned to the Make Payment form. The user will be charged again if the Make Payment button is selected again.

7. The confirmation form from your Forms Builder sequence is displayed.



E-Signature Forms

Forms Builder gives you the ability to create forms of any kind and add electronic signatures to them. Forms Builder offers integration with the following e-signature solutions:

- Adobe Sign
- DocuSign
- SignNow

Electronic Signatures vs. Digital Signatures

The terms digital signature and electronic signature are often used interchangeably. However, while digital signatures are a form of electronic signature, not all electronic signatures are digital signatures.

The main difference between the two is that:

- Electronic signatures are often associated with a contract where the signer has the intention to sign the document or the contract.
- Digital signatures are mainly used to secure documents and are authorized by certification authorities.

Electronic Signatures

Electronic signatures (or e-signatures) are an electronic sound, symbol, or process, attached to or logically associated with a contract or other record and executed or adopted by a person with the intent to sign the record. An e-signature is a digitized version of your handwritten signature. It can be used to indicate the intent to approve or accept the contents of a document.

E-signatures' greatest advantage is that they are easy to use. They allow users to sign documents online with the click of a mouse, or trace their handwritten signature using a finger or a stylus.

Digital Signatures

A digital signature—a type of electronic signature—is a mathematical algorithm routinely used to validate the authenticity and integrity of a message (e.g., an email, a credit card transaction, or a digital document). Digital signatures create a virtual fingerprint that is unique to a person or entity and are used to identify users and protect information in digital messages or documents. Digital signatures are significantly more secure than other forms of electronic signatures.

There are typically three algorithms involved with the digital signature process:

- Key generation – This algorithm provides a private key along with its corresponding public key.
- Signing – This algorithm produces a signature upon receiving a private key and the message that is being signed.

- Verification – This algorithm checks for the authenticity of the message by verifying it along with the signature and public key.

Adobe Sign Forms

Forms Builder provides integration with Adobe Sign, a cloud-based electronic signature service that lets you send, sign, track, and manage signature processes. For more information, refer to https://acrobat.adobe.com/us/en/sign/capabilities/electronic-signatures.html.

Adobe Sign Settings

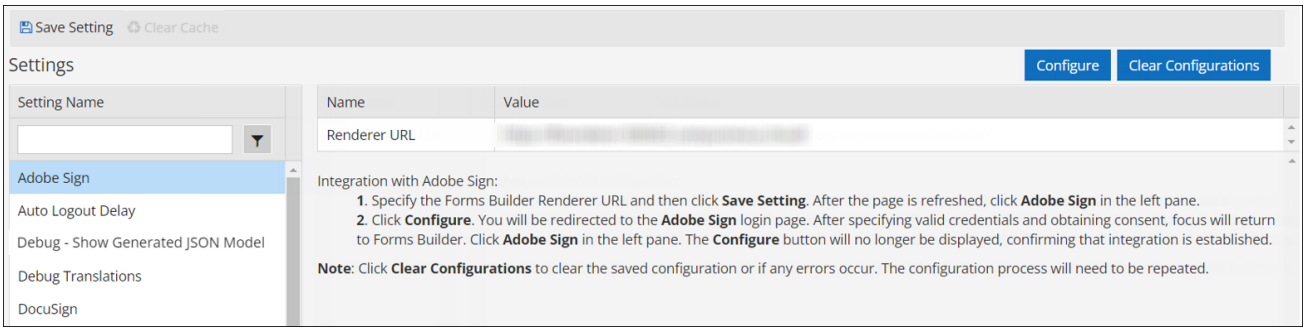
The first step in integrating with Adobe Sign is to create an account at this URL - https://secure.echosign.com/public/login.

Anthology is a certified integration partner with Adobe Sign, which means that the Anthology application that's created is certified by Adobe Sign.

Institutions can integrate with Adobe Sign through a login process by using your Adobe Sign login credentials. Back-end details of Anthology's certified application will then be used in combination with your login credentials to enable end-users to work with Adobe Sign.

To Integrate With Adobe Sign

1. On the Forms Builder home page, select the Settings tile and select **Adobe Sign** in the Settings pane. The Properties pane will be displayed.

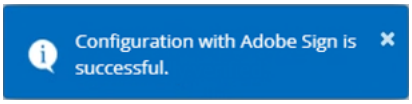


2. In the **Renderer URL** field, specify the Forms Builder Renderer URL and then select **Save Setting**.
3. After the page is refreshed, select Adobe Sign in the left pane and then select Configure.

   You will be redirected to the Adobe Sign login page.

   Specify valid login credentials and also obtain consent. In the background, values specific to the integration process will be fetched from **Adobe Sign** and stored in Forms Builder.

   When consent is obtained, focus will return to Forms Builder. The following message will also be displayed:



4. Select **Adobe Sign** in the left pane. The Configure button will no longer be displayed, confirming that integration is established.
5. Reset IIS or wait an hour for the changes to take effect.

Reminder:

- Select Clear Configurations to clear saved integration settings. For example, administrators can clear configurations in a scenario where errors occur. After configurations are cleared, the integration process must

be repeated.

- If Adobe Sign was not used in the previous 60 days, the configuration process will need to be repeated, which will auto-generate a new refresh token that will then be applied in the background when Adobe Sign is used.

Adobe Sign Workflow Sample - Single Signer

The following procedure details a sample workflow and forms for a sequence that contains single signer.

Prerequisites

1. The Adobe Sign properties are configured in Forms Builder. Refer to SignNow Settings.
2. A form sequence with the necessary Adobe Sign components for a single signer is created. In our example, the **Anonymous** property of the sequence is selected.

   If you create a non-authenticated sequence, make sure that the user's email address is included in the sequence. The email address is required as the Adobe Sign recipient address.
3. A form that contains an IFrame component is created. In our example, this form is named *Default-Frame* with the following properties (case sensitive):
   - Name = **AdobeSignFrame**
   - Url = **{{vm.models.frameUrl}}**

   The IFrame form can be part of the initial sequence; however, in the procedure below, we are adding the form to the workflow.

   > ⚠ We recommend that you copy the original *Default-Frame* form, edit the copy, and use it in your sequences. Save a backup copy of your form.

To optimize the display of your e-signature forms on mobile devices, you may want to add the CSS described in this Note.

Create the Workflow

Open the Workflow and Review Arguments, Variables, and Logging Requirements

1. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
2. In Workflow Composer, drag the State icons and Transition lines to so that you can easily locate each item in the StateMachine workflow.

The workflow requires the following arguments and variables. You can create these arguments and variables before working on the activities, or you can add them when they are needed for a specific activity (as described below).

> ⚠ Keep in mind that arguments are passed in JSON format and that JSON elements are case sensitive. **Be sure to match the casing of argument names in Workflow Composer and Form Designer.**

Arguments

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*
studentEntity	In/Out	StudentEntity	*Default value not supported*
frameUrl	In/Out	String	*Default value not supported*

Variables

Name	Variable type	Scope	Default
renderedFormImage	String	StateMachine	*Enter a VB expression*
Pdf	Byte[]	StateMachine	*Enter a VB expression*
Doc	DocumentEntity	StateMachine	*Enter a VB expression*
URL	String	StateMachine	*Enter a VB expression*
studentId	Int32	StateMachine	*Enter a VB expression*
AdobeSignConfig	AdobeSignConfig	StateMachine	*Enter a VB expression*
AdobeSignDocument	AdobeSignDocument	StateMachine	New AdobeSignDocument
AdobeSignRecipient	AdobeSignRecipient	StateMachine	New AdobeSignRecipient
AdobeSignRequest	AdobeSignRequest	StateMachine	*Enter a VB expression*

LogLine Activities

While testing and troubleshooting the workflow, we recommend adding LogLine activities at critical stages within the workflow. The following expression will provide logging for specific objects (replace <object> with the object name):

Newtonsoft.Json.JsonConvert.SerializeObject(**<object>**, Newtonsoft.Json.Formatting.Indented)

For example, to capture errors related to the CreateAdobeSignRequest activity, you would place a LogLine activity with the following expression below the CreateAdobeSignRequest activity.

Newtonsoft.Json.JsonConvert.SerializeObject(**AdobeSignRequest**, Newtonsoft.Json.Formatting.Indented)

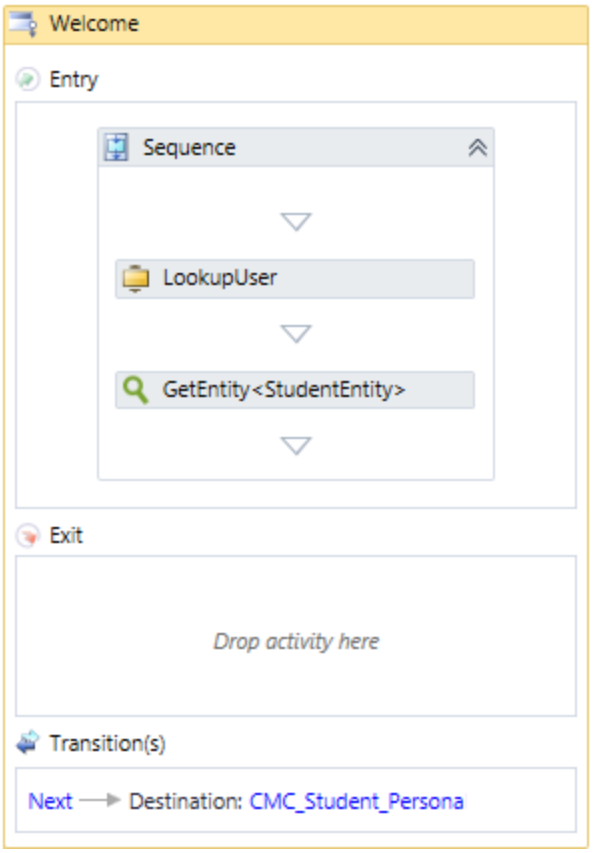
We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

Capture the Login User Information

If your sequence is non-authenticated (Anonymous=true), the LookupUser and GetEntity activities are not applicable, but a CreateEntity activity may be needed instead. Be sure that any anonymous sequence includes the user's email address. It is required as the Adobe Sign recipient address.

The first step is to capture the information of the user who logged in to the form.

1. Create a variable named **studentId** of Type **Int32**.
2. Drop a **LookupUser** activity in the Entry area of the Welcome form.
    - In the UserId property, specify **studentId**. This is the variable created in the previous step.
    - In the UserName property, specify **formInstance.UserName**. The user name is retrieved from the login account information.



3. Drop a **GetEntity** activity in the below the LookupUser activity.
    - In the EntityId property, specify **studentId**.
    - In the Result property, specify **studentEntity**.

    Workflow Composer automatically wraps the activities in a Sequence.
4. **Save** the workflow locally and continue with the next set of steps.

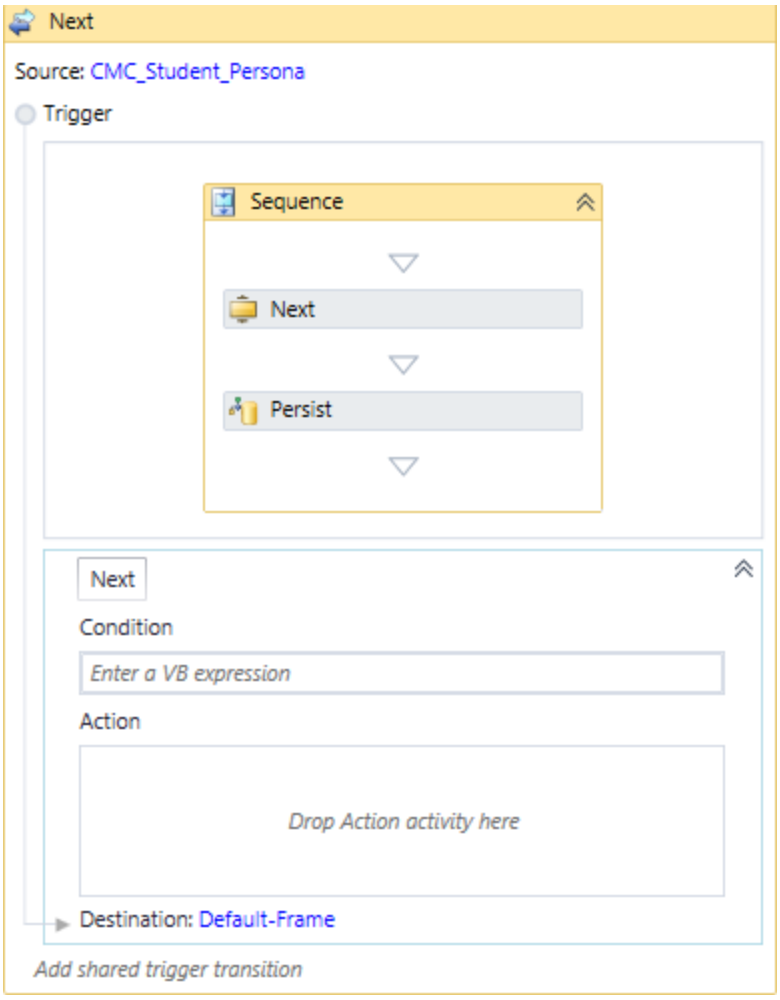
Persist the Workflow Instance After Collecting the Form Data

Since the workflow can be busy for a long time, we recommend that you persist the workflow instance before creating the PDF.

1. Double-click the **Next** transition from the form that is used to collect the data, *CMC_Student_Personal Info*.

2. Drag a **Persist** activity into the Trigger area below the Next activity. The Persist activity does not require any properties to be specified.

   Workflow Composer automatically wraps the Next and Persist activities in a Sequence.



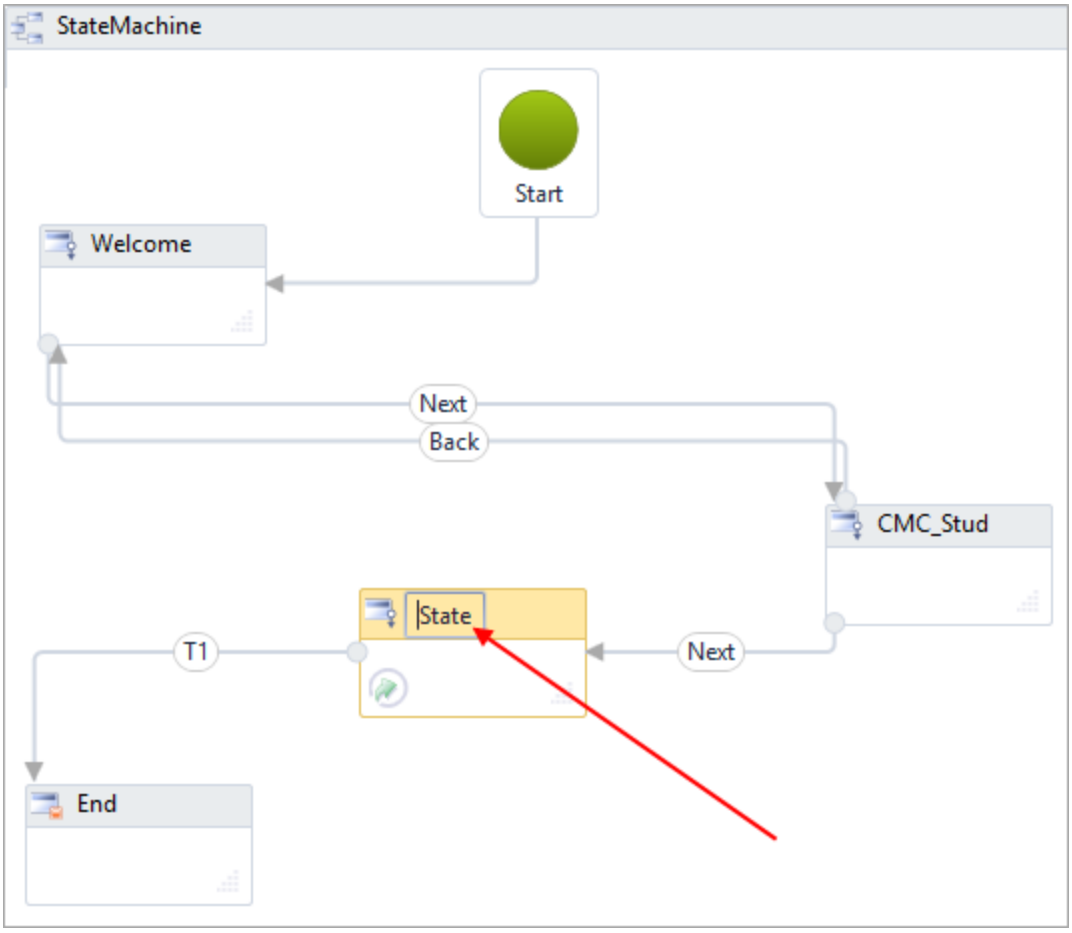
3. **Save** the workflow locally and continue with the next set of steps.

Add an IFrame Form to the Workflow

1. In Workflow Composer, select the **Arguments** tab.
2. Create an argument named **frameUrl**. Set the Direction to **In/Out**. Set the Type to **String**.

   Be sure to use the exact casing shown here.

3. Select the **StateMachine** link in the Designer pane.
4. Drop a **State** activity onto the transition line after the Next transition from the form that contains the Adobe Sign components (*CMC_Student_Personal Info*).



5. Change the default name of the State activity to the name of the form that contains the IFrame. In our case, the name is **Default-Frame**.
6. **Save** the workflow locally and continue with the next set of steps.

Create a PDF of the Adobe Sign Form

In the following steps we will create a set of activities in the *Default-Frame* State, which represents the IFrame form.

1. Double-click the State icon of the form that contains the IFrame component.
2. Create a variable named **URL**. Set the Variable type to **String**. Set the Scope to **StateMachine**. This variable will be assigned to the PDF that will be created from the form.
3. Drop an **Assign** activity into the Entry area of the State.
    a. In the *To* field, specify the variable name **URL**.
    b. In the *Value* field, specify the following:

forminstance.RendererBaseUrl + "#/viewCreator/" + formInstance.WorkflowDefinitionId.ToString + "/forms=CMC_Student_Personal_AdobeSign"

Where "/forms=*CMC_Student_Personal_AdobeSign*" indicates the Forms Builder form that contains the Adobe Sign components.

Notes

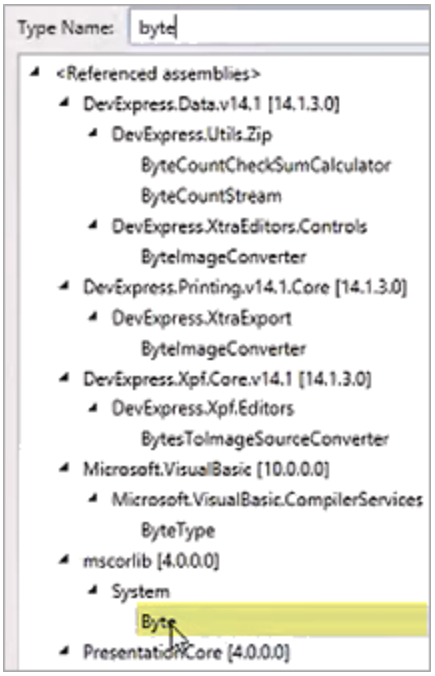
- If the form name has a space, replace the space with a + sign, e.g., *CMC_Student_Personal+Info_AdobeSign*
- If multiple forms are sent to Adobe Sign, specify a comma-separated list of form names in the "/forms=" attribute.

4. Drop a **Persist** activity below the Assign activity.
5. Create a variable with a name like **Pdf**. Set the Scope to **StateMachine**. This variable will hold the document image created from the form.

   Make sure you use the same variable name when you reference this variable later in the workflow. (This applies to any other variable.)

   a. In the Variable type field, select **Array of [T]** and select **Browse for Type**.

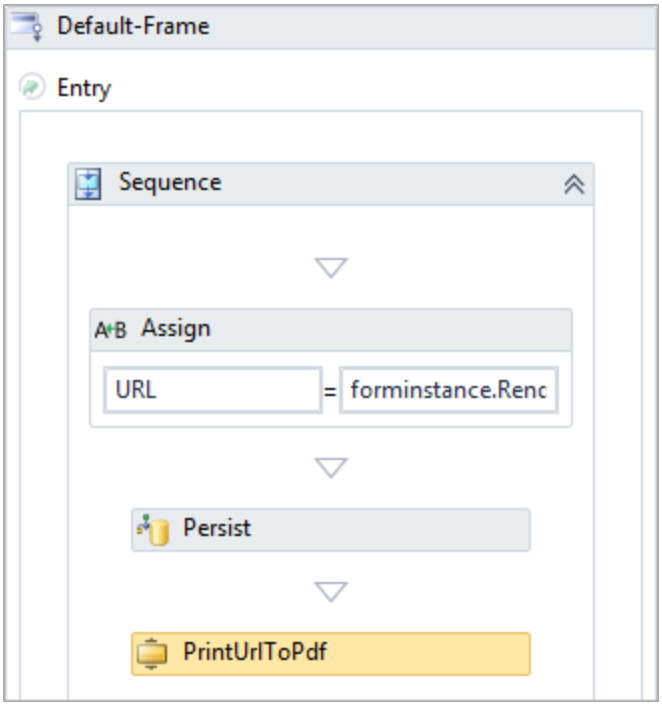
   b. In the "Browse and Select a .Net Type" window, specify "**byte**", select the System variable "**Byte**", and select **OK**.



6. Drop a **PrintUrlToPdf** activity below the Persist activity.

   Specify the properties for the activity as follows:

- PdfDocument = **Pdf** (This is the name of the variable created above.)
- Url = **URL** (This is the name of the variable created above.)
- Validation Messages = **formInstance.ValidationMessages**



7. **Save** the workflow locally and continue with the next set of steps.

Get the Adobe Sign Configuration and Pass the Recipient Information

In the following steps we will continue to add activities to the *Default-Frame* State, which represents the IFrame form.

1. Create a variable named **AdobeSignConfig**. Set the Scope to **StateMachine**.

   In the Variable type field, select **Browse for Type**. In the "Browse and Select a .Net Type" window, scroll down to Cmc.Nexus.FormsBuilder.Contracts > Cmc.Nexus.FormsBuilder.Entities and select **AdobeSignConfig**.

2. In the State of the IFrame form, drop a **GetAdobeSignConfig** activity below the PrintUrlToPdf activity.

   Specify the properties for the activity as follows:

   - AdobeSignConfig = **AdobeSignConfig**
   - Validation Messages = **formInstance.ValidationMessages**

   The GetAdobeSignConfig activity retrieves the User Name, Password, Integrators Key, and REST API Url from the Adobe Sign settings in Forms Builder. These values enable the workflow to log in to Adobe Sign.

To process the Adobe Sign request, the Email Subject needs to be assigned to the AdobeSignConfig variable. This property is required.

3. Create the following variables:

Variable Name	Type	Scope	Default
AdobeSignDocument	Cmc.Nexus.FormsBuilder.Entities.AdobeSignDocument	StateMachine	New AdobeSignDocument
AdobeSignRecipient	Cmc.Nexus.FormsBuilder.Entities.AdobeSignRecipient	StateMachine	New AdobeSignRecipient
AdobeSignRequest	Cmc.Nexus.FormsBuilder.Entities.AdobeSignRequest	StateMachine	N/A

4. Below the GetAdobeSignConfig activity, drop an **Assign** activity for each row in the following table and type the indicated values:

"To" Field	Value	Notes
AdobeSignConfig.EmailSubject	"CMC Campus AdobeSign Testing"	This is the email that the end user will receive after the signing process is done.
AdobeSignDocument.DocumentId	"1"	When multiple AdobeSign documents are included in the same AdobeSign envelope, every document needs a unique Id. In this example, there is only one document.
AdobeSignDocument.Name	"StudentInfoPdf.pdf"	Name of the signed document.
AdobeSignDocument.Content	Pdf	Name of the variable that holds the document image (refer to Create a PDF of the Form). In our example the name of the variable is **Pdf**.
AdobeSignRecipient.FirstName	studentEntity.FirstName	First name of the user who submitted the signed document.
AdobeSignRecipient.LastName	studentEntity.LastName	Last name of the user who submitted the signed document.

"To" Field	Value	Notes
AdobeSignRecipient.Email	AdobeSignConfig.SignerAccountEmail	Email address of the person who will receive the signed document.

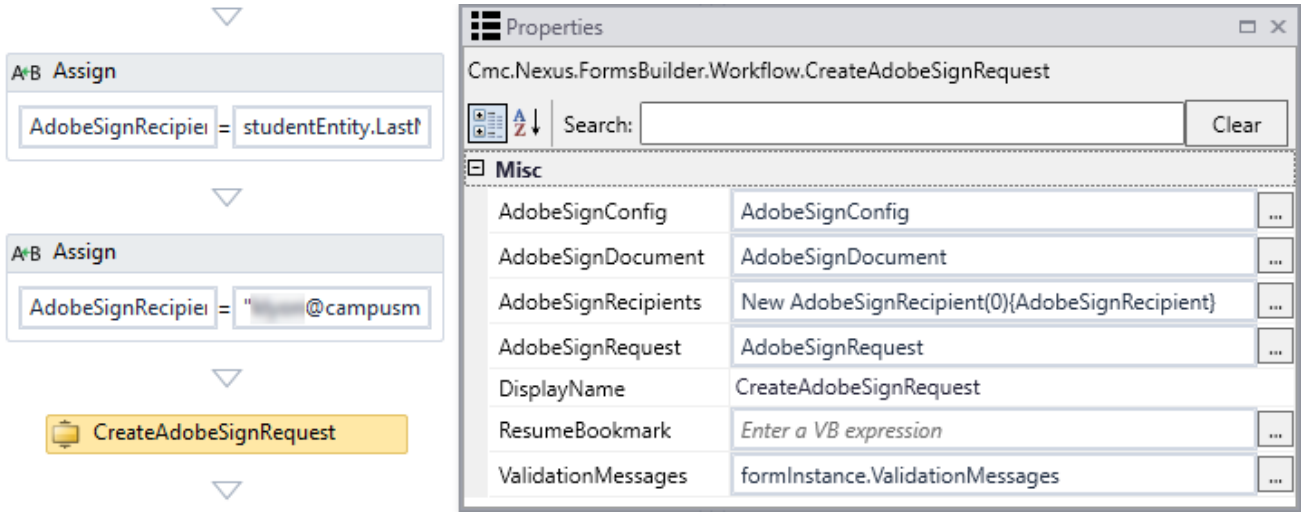
5. **Save** the workflow locally and continue with the next set of steps.

Create the Adobe Sign Request and Specify the IFrame URL

In the following steps we will continue to add activities to the *Default-Frame* State, which represents the IFrame form.

1. Drop the **CreateAdobeSignRequest** activity below the previous activity.

   Specify the properties for the CreateAdobeSignRequest activity using the names of the variables created above as follows:



   Warning: Do not enter anything in ResumeBookmark if you are creating single signer workflows. This could cause InstanceHandleConflictExceptions and aborted workflows. Fill this in only for Multi-Signer workflows.

2. Drop an **If** activity below the CreateAdobeSignRequest activity.
   a. In the *Condition* field, specify **formInstance.ValidationMessages.HasErrors**.
   b. In the *Else* branch, drop an **Assign** activity.

      In the *To* field, specify **frameUrl**. (This is the argument associated with the IFrame form.)

In the *Value* field, specify **AdobeSignRequest.Url**.

 Be sure to use the exact casing shown here.

3. **Save** the workflow locally and continue with the next set of steps.

Receive the Signed Adobe Sign Document

In the following steps we will create activities in the Submit transition which follows the IFrame form.

1. Double-click the **T1** transition after the *Default-Frame* state and rename it as **Submit**.
2. Drop a **WaitForFormBookmark** activity into the Trigger area of the Submit transition.
3. Rename the WaitForFormBookmark activity as **Submit**.
4. Create a variable named **AdobeSignedDocument**. In the Variable type field, select **AdobeSignDocument**. Set the Scope to **StateMachine**.
5. Drop a **GetAdobeSignSignedDocument** activity below the Submit activity.

   Specify the properties for the activity as follows:

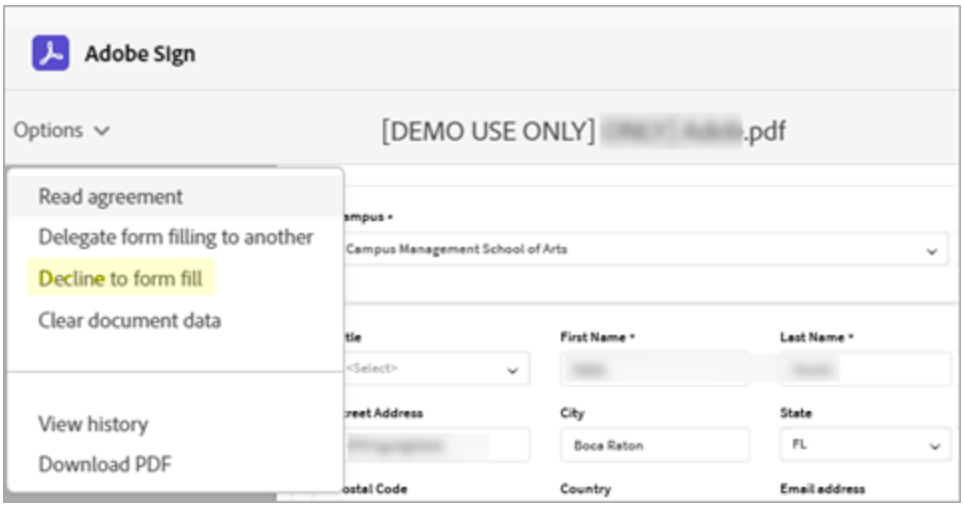
   - AgreementId= **AdobeSignRequest.AgreementId**
   - AdobeSignConfig = **AdobeSignConfig**
   - AdobeSignDocument = **AdobeSignedDocument** (This is the name of the variable created above.)
   - Validation Messages = **formInstance.ValidationMessages**

   Workflow Composer automatically wraps the activities in a Sequence.
6. **Save** the workflow locally and continue with the next set of steps.

You can log in to your Adobe Sign account and view the signed document "StudentInfoPdf".

Note

Adobe Sign provides signers the **Decline to form fill** option.



If the decline option is selected:

- The signer will access a webpage (IFrame) stating "*You declined to sign <PDF name>.*"
- The signer will receive an email stating that Adobe Sign terminated the agreement exchange.
- Adobe Sign will not return control of the sequence to Forms Builder and the workflow will not continue. The workflow will remain as a persisted instance. You will need to terminate persisted workflows periodically using either Sequence Designer (refer to Delete Persisted Workflow Instances) or Workflow Composer (refer to Persisted Workflows).

Create and Save the Document in Anthology Student

To convert an Adobe Sign document to a DocumentEntity that can be attached to a record in Anthology Student, add CreateDocument and SaveDocument activities to the workflow. These activities will be placed in the Submit transition.

1. Create a variable named **Doc**. Set the Scope to **StateMachine**.

   In the Variable type field, select **Browse for Type**. In the "Browse and Select a .Net Type" window, scroll down to Cmc.Nexus.Crm.Contracts > Cmc.Nexus.Crm.Entities and select **DocumentEntity**.

2. Drop a **CreateDocument** activity below the GetSignedDocument activity.

   Specify the required properties for the activity as follows:

   - Module = Select a Module from the drop-down list.
   - Document Type = Select a Type (Template) from the drop-down list.
   - Document Status = Select a Type from the drop-down list.
   - Student = Specify a Student Id or use a variable.
   - Due Date = Specify a date or use a variable, e.g., DateTime.Now
   - Document (OutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
   - Validation Messages = formInstance.ValidationMessages

3. Below the CreateDocument activity, drop an **Assign** activity for each row in the following table and type the indicated values:

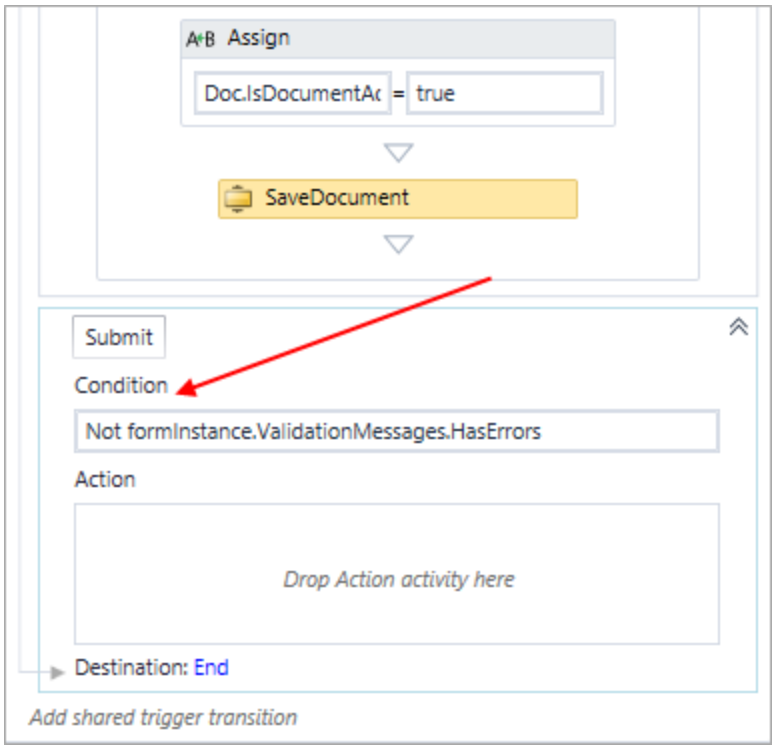
"To" Field	Value
Doc.DocumentImage	SignedDocument.Content
Doc.OriginalFileName	"Signedpdf.Pdf"
Doc.ImageType	"Pdf"
Doc.IsDocumentAddedManually	true

4. Drop a **SaveDocument** activity below the CreateDocument activity.

   Specify the properties for the SaveDocument activity as follows:

   - Document (InOutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
   - Validation Messages = **formInstance.ValidationMessages**

5. In the Condition field of the Submit transition, specify the following (to prevent users from submitting a form that has validation errors): **Not formInstance.ValidationMessages.Has Errors**



6. **Save** the workflow locally and continue with the next set of steps.

Final Steps

1. Select **Publish**. The New Workflow Definition Version window is displayed.
2. If you want the workflow to be run as soon as the event occurs on the entity, select **Enable This Workflow Version?**, otherwise leave the check box cleared.
3. Select **Save**, then **Cancel** to close the publisher window.
4. In the **Sequence List**, select and complete the form that contains the Adobe Sign fields.
5. Verify that the signed Adobe Sign document is available at the return URL.
6. Verify that the signed document is added to the Anthology Student database.

While testing your workflow, make sure that you terminate the previous instance of the workflow before running an updated version of the same workflow. In Workflow Composer, select **Open Persisted Workflow**, select your workflow instance, and select **Terminate**.

Persisted workflow instances can be deleted from the Sequence Designer workspace. For more information, refer to Delete Persisted Workflow Instances.

Adobe Sign Workflow Sample - Multiple Signers

The following procedure details a sample workflow and forms for a sequence that contains multiple signers.

Prerequisites

1. The Adobe Sign properties are configured in Forms Builder. Refer to Adobe Sign Settings.
2. An authenticated form sequence is created. In our example, the sequence includes the following forms:
    a. *Welcome*
    b. *CMC_Student_Personal_Adobe_Multi* — This is an admissions application form that collects the personal data of a student and contains 6 Adobe Sign components:
        - Signer 1: Signature, Date Signed, Date, and Initial (4 components)
        - Signer 2: Signature, Date Signed (2 components)

    The signing process for the primary signer is the same as described in Adobe Sign Workflow Sample - Single Signer. The signing process for the primary signer is an embedded process within the form window. For the secondary signer, Adobe Sign sends an email along with the document to be signed to the secondary signer. The signing process for primary and secondary signer takes place within the same Adobe Sign request. Adobe Sign returns events when each signer signs the document and when the document is complete. The workflow responds to these events.
    c. *Default-Frame* — This form contains an IFrame component.

    > ⚠ We recommend that you copy the original *Default-Frame* form, edit the copy, and use it in your sequences. Save a backup copy of your form.

    In your copy of the form, edit the Name property as shown below (case sensitive). Keep the URL property as {{vm.models.frameUrl}}.

    Name = *AdobeSignFrame*
    d. *Default-DocuSignWait* — This form is provided with the Forms Builder installation. We suggest that you copy and rename it to *AdobeSignWait*. It contains two HTML components.

    The first HTML component displays the following text. Edit the text in your copy of the form.

    "*Current user may close this browser tab. This page will auto submit when all the signatures are collected from DocuSign.*"

    The second HTML component contains JavaScript code that is not applicable to Adobe Sign. Remove that HTML component in your *AdobeSignWait* form.

> ⚠ The *Default-DocuSignWait* form will be overwritten when Forms Builder is upgraded. Copy the original form, rename it to *AdobeSignWait*, edit the copy, and use it in your sequences. Save a backup copy of your form.

This logic can be repeated as needed for additional signers.

e. *Default-Confirmation* — This form is provided with the Forms Builder installation. It has an HTML component that displays the text:

"*This page will be displayed at end of each sequence as confirmation. Please update with custom confirmation message.*"

To optimize the display of your e-signature forms on mobile devices, you may want to add the CSS described in this Note.

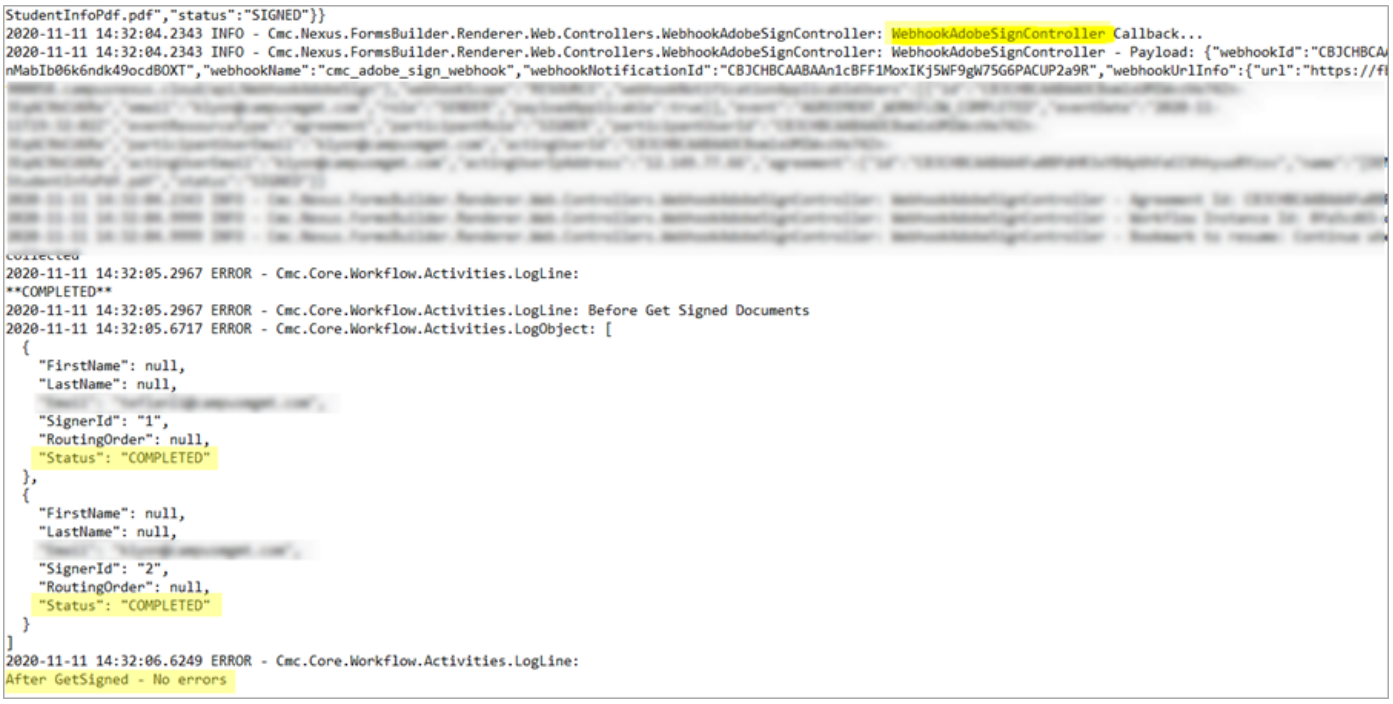
Test the Multiple Signer Feature

To test the multiple signature feature with Adobe Sign, the test environment needs to meet specific requirements. Adobe Sign will try to call APIs on Renderer when secondary signers complete their signing process via email. This feature is referred to as *Webhook*.

- If you are testing with an Adobe Sign account in test mode:
  - Renderer must be hosted on port 80 or 443.
  - Port 80 or 443 must be open on the firewall depending on which port Renderer is using for Adobe Sign to communicate. You can request an exception from your IT team for port 80 or 443 for an IP address range.
- If you are testing with a live Adobe Sign account:
  - Renderer must be hosted on port 443 (https).
  - If hosting Renderer on port 443 is not possible, an Adobe Sign representative for your account will have to make an exception.

To confirm that multi-signature forms are completing the workflow process, please confirm that Webhook responses are received and displayed in the log. For more information, refer to Webhook Responses for Multiple Signer Sequences.

Log Example

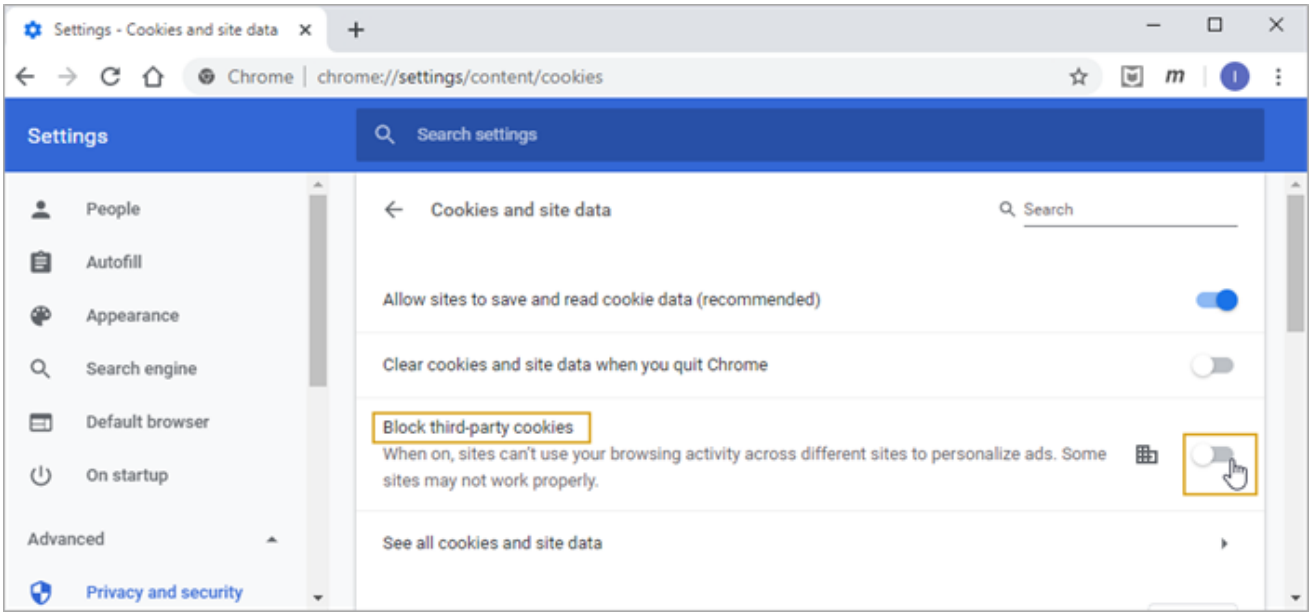


Chrome Browser Settings for Multiple Signers

The multiple signer feature for Adobe Sign requires a specific setting in Google Chrome. If you experience issues with multiple signer forms, ensure that third-party cookies are allowed.

1. In Chrome, select **Settings**.
2. Select **Advanced** at the bottom of the page.
3. Expand **Site Settings**.
4. Expand **Cookies and site data**.

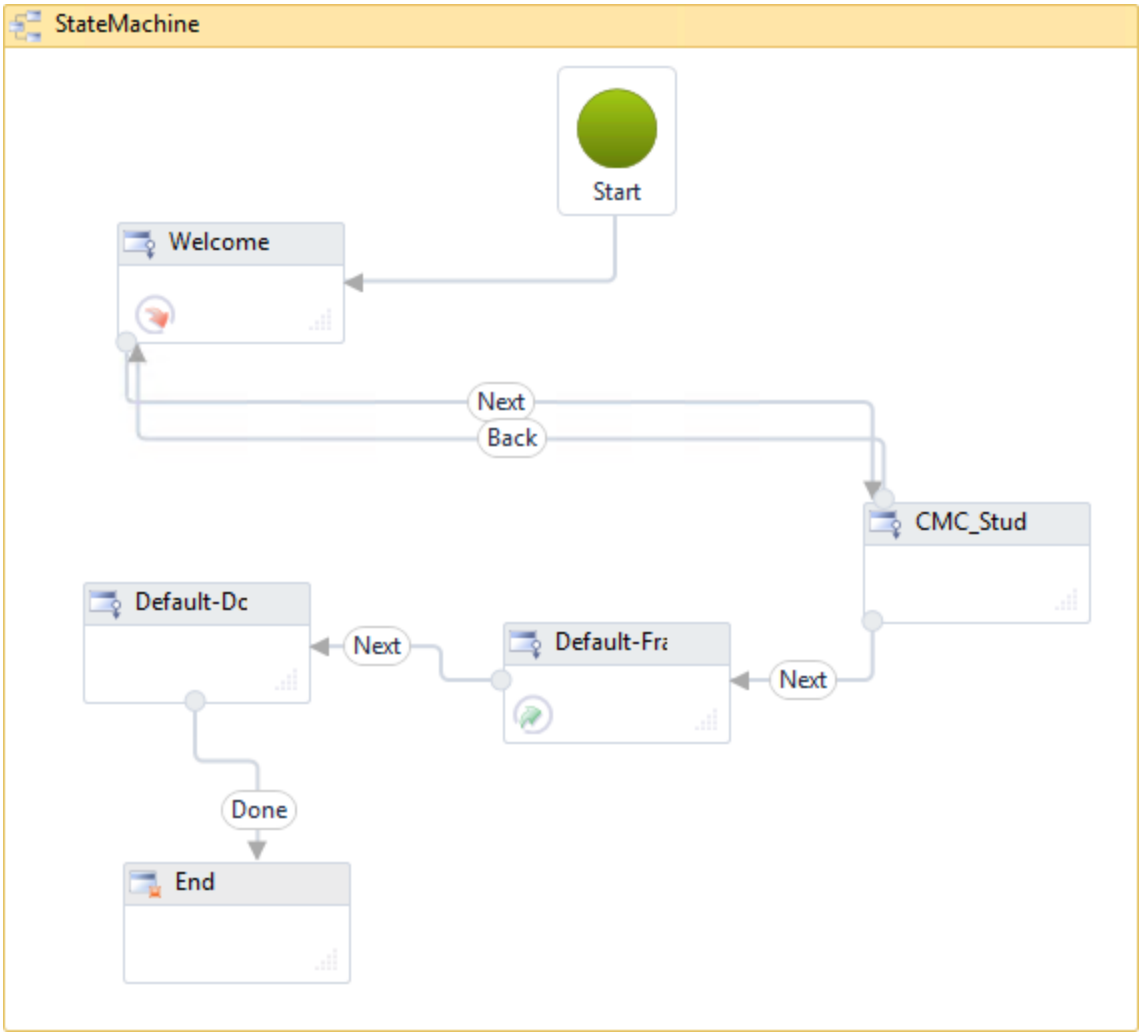
5. Disable **Block third-party cookies**.



Create the Workflow

Open the Workflow, Review Arguments, Variables, and Logging Requirements

1. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
2. In Workflow Composer, drag the state icons and transition lines to so that you can easily locate each item in the StateMachine workflow.



The workflow requires the following arguments and variables. You can create these arguments and variables before working on the activities, or you can add them when they are needed for a specific activity (described below).

> ⚠ Keep in mind that arguments are passed in JSON format and that JSON elements are case sensitive. **Be sure to match the casing of argument names in Workflow Composer and Form Designer.**

Arguments

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*
studentEntity	In/Out	StudentEntity	*Default value not supported*
frameUrl	In/Out	String	*Default value not supported*

Variables

Name	Variable type	Scope	Default
renderedFormImage	String	StateMachine	*Enter a VB expression*
Pdf	Byte[]	StateMachine	*Enter a VB expression*
AdobeSignConfig	AdobeSignConfig	StateMachine	*Enter a VB expression*
Doc	DocumentEntity	StateMachine	*Enter a VB expression*
URL	String	StateMachine	*Enter a VB expression*
studentId	Int32	StateMachine	*Enter a VB expression*
AdobeSignDocument	AdobeSignDocument	StateMachine	New AdobesignDocument
AdobeSignRequest	AdobeSignRequest	StateMachine	*Enter a VB expression*
AdobeSignRecipient	AdobeSignRecipient	StateMachine	New AdobeSignRecipient
AdobeSignRecipient2	AdobeSignRecipient	StateMachine	New AdobeSignRecipient
AdobeSignRecipients	AdobeSignRecipient[]	StateMachine	*Enter a VB expression*

LogLine Activities

While testing and troubleshooting the workflow, we recommend adding LogLine activities at critical stages within the workflow. The following expression will provide logging for specific objects (replace <object> with the object name):

Newtonsoft.Json.JsonConvert.SerializeObject(**<object>**, Newtonsoft.Json.Formatting.Indented)

We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

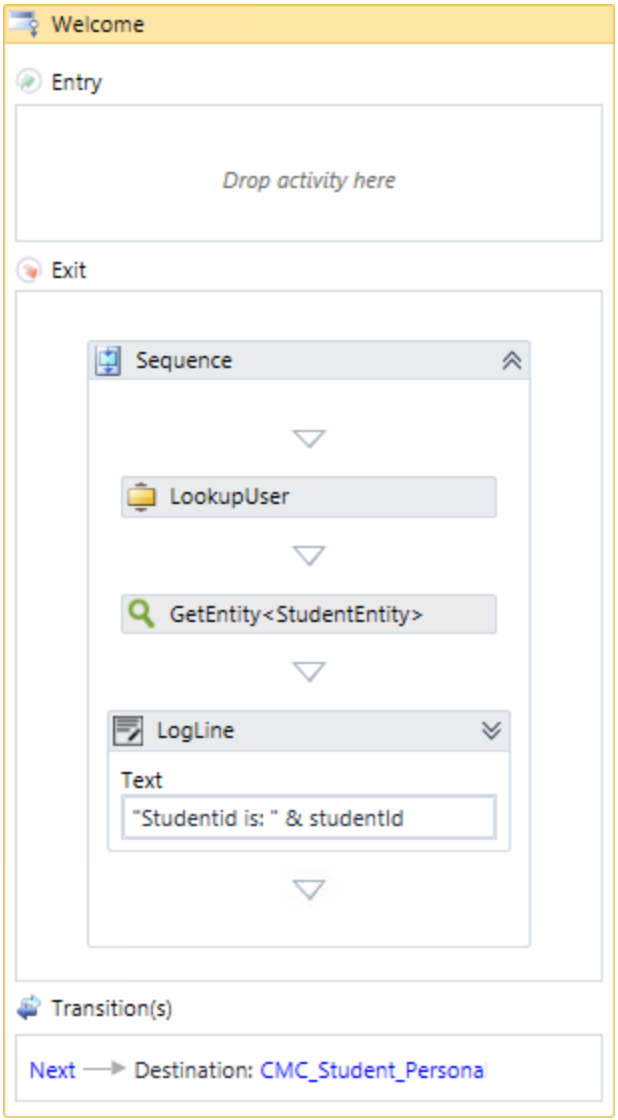
LookupUser and GetEntity

If your sequence is non-authenticated (Anonymous=true), the LookupUser and GetEntity activities are not applicable, but a CreateEntity activity may be needed instead. Be sure that any anonymous sequence includes the user's email address. It is required as the Adobe Sign recipient address.

Use the *Welcome* form to retrieve the Student Id and Student Entity.

1. Double-click the icon of the **Welcome** state.
2. Create a variable named **studentid** of Type **Int32**.
3. Drop a **LookupUser** activity into the Entry area of the Welcome state.
    - In the **UserId** property, specify the **studentid** variable.
    - In the **UserName** property, specify **formInstance.UserName**.
4. Drop a **GetEntity** activity below the LookupUser activity.

- In the **EntityId** property, specify the **studentid** variable.
- In the **Result** property, specify **studentEntity**.

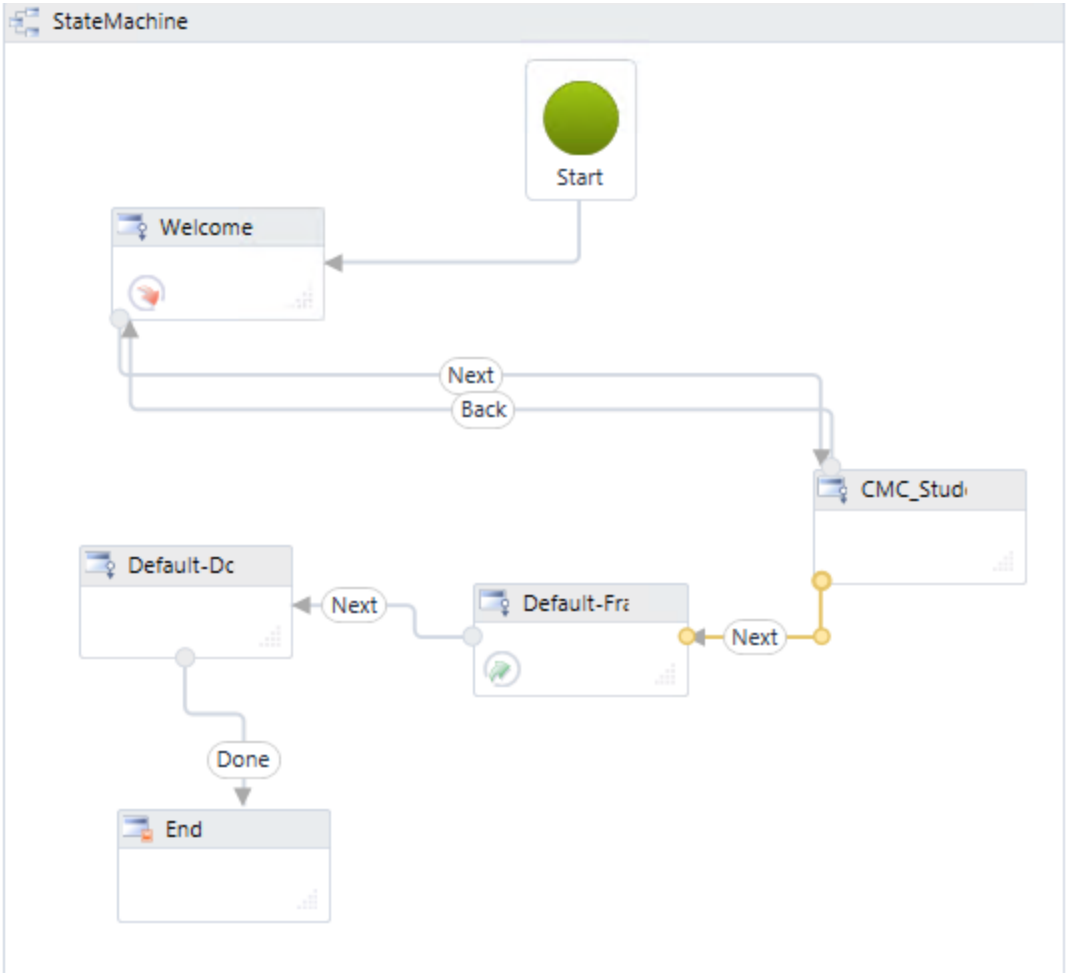


5. **Save** the workflow locally and continue with the next set of steps.

Persist the Workflow Instance After Collecting the Form Data

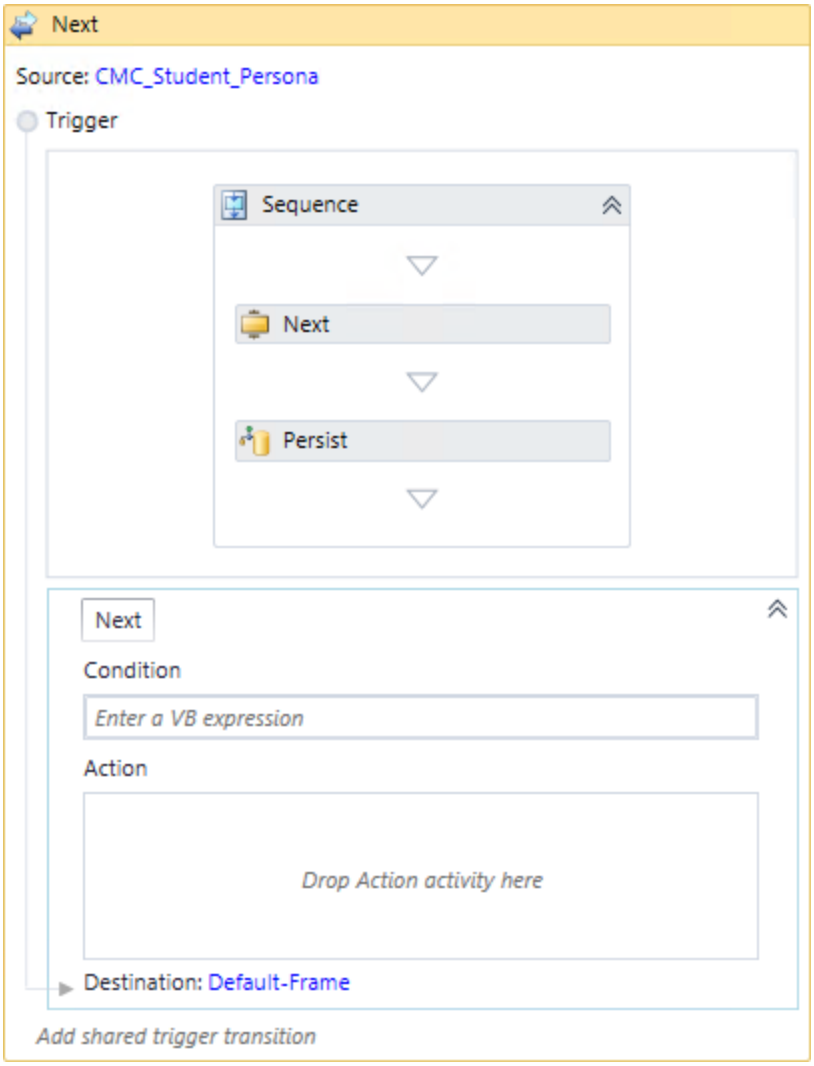
Since the workflow can be busy for a long time, we recommend that you persist the workflow instance before creating the PDF.

1. Double-click the **Next** transition between the *CMC_Student_Personal_Adobe_Multi* and *Default-Frame* states.



2. Drag a **Persist** activity into the Trigger area below the Next activity. The Persist activity does not require any properties to be specified.

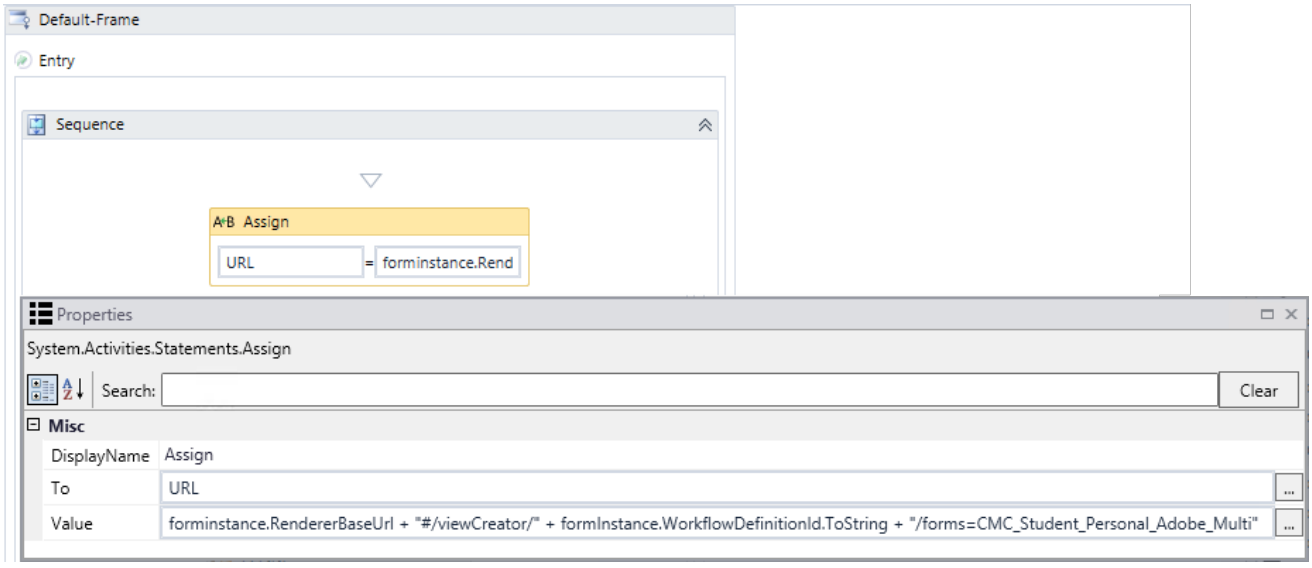
   Workflow Composer automatically wraps the Next and Persist activities in a Sequence.



3. **Save** the workflow locally and continue with the next set of steps.

Create a PDF of the Adobe Sign Form

1. Create a variable named **URL**. Set the Variable type to **String**. Set the Scope to **StateMachine**. This variable will be assigned to the PDF created from the form.
2. Drop an **Assign** activity into the Entry area of the *Default-Frame* state.



a. In the *To* field, specify the variable name **URL**.

b. In the *Value* field, specify the following:

formInstance.RendererBaseUrl + "#/viewCreator/" + formInstance.WorkflowDefinitionId.ToString + "/forms=CMC_Student_Personal_Adobe_Multi"

Where "/forms=*CMC_Student_Personal_Adobe_Multi*" indicates the Forms Builder form that contains the Adobe Sign components.

Notes

- If the form name has a space, replace the space with a + sign.
- If multiple forms are sent to Adobe Sign, specify a comma-separated list of form names in the "/forms=" attribute.

3. Drop a **Persist** activity below the Assign activity.

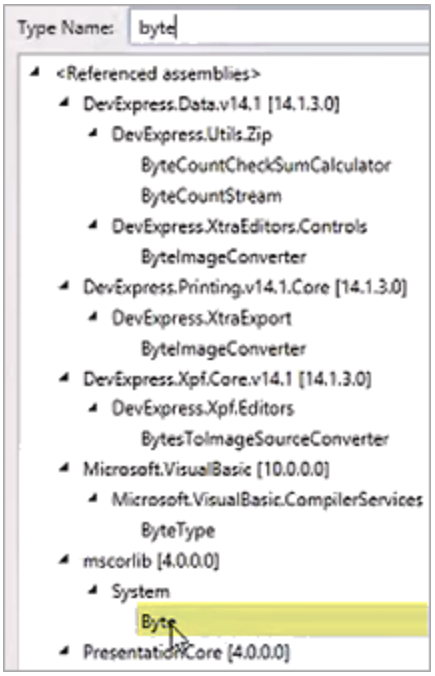
4. Create a variable with a name like **Pdf**. Set the Scope to **StateMachine**. This variable will hold the document image created from the form.

Make sure you use the same variable name when you reference this variable later in the workflow. (This applies to any other variable.)

a. In the Variable type field, select **Array of [T]** and select **Browse for Type**.

b. In the "Browse and Select a .Net Type" window, specify **byte**, select the System variable **Byte**, and

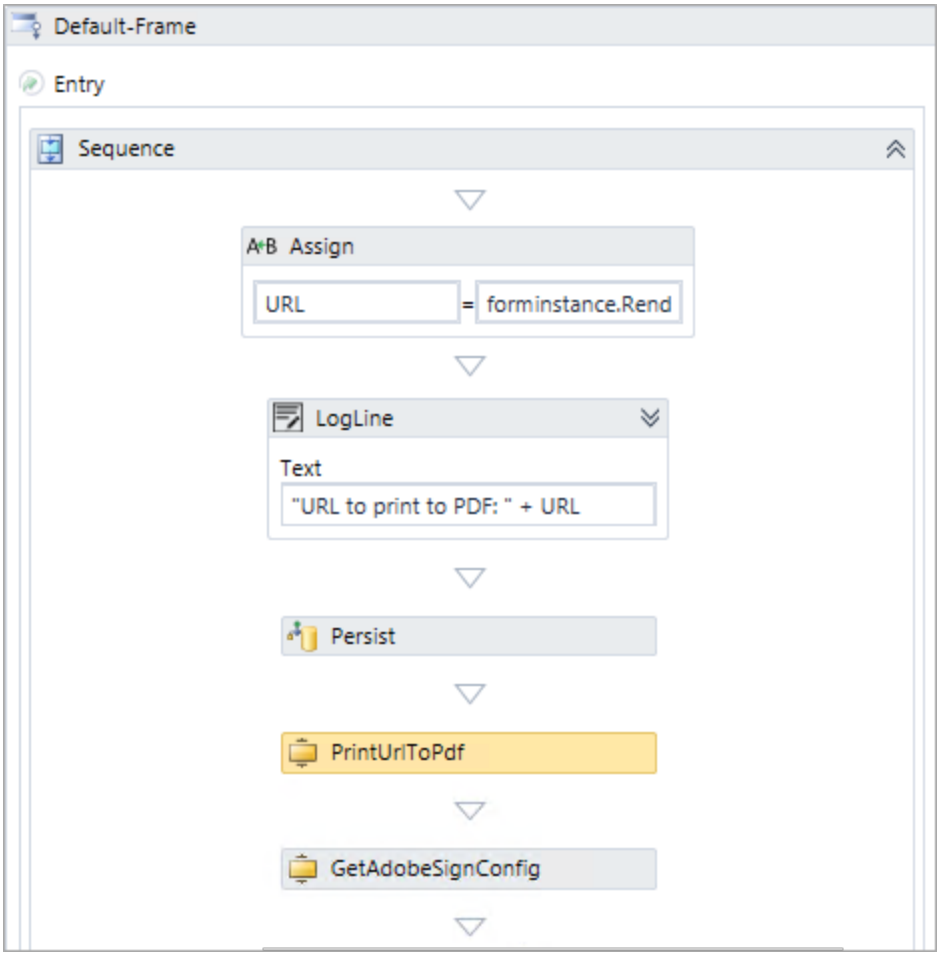
select **OK**.



5. Drop a **PrintUrlToPdf** activity below the Persist activity.

   Specify the properties for the activity as follows:

   - PdfDocument = **Pdf** (This is the name of the variable created above.)
   - Url = **URL** (This is the name of the variable created above.)
   - Validation Messages = **formInstance.ValidationMessages**



6. **Save** the workflow locally and continue with the next set of steps.

Get the Adobe Sign Configuration and Pass the Recipient Information

In the following steps we will continue to add activities to the *Default-Frame* state.

1. Create a variable named **AdobeSignConfig**. Set the Scope to **StateMachine**.

   In the Variable type field, select **Browse for Type**. In the "Browse and Select a .Net Type" window, scroll down to Cmc.Nexus.FormsBuilder.Contracts > Cmc.Nexus.FormsBuilder.Entities and select **AdobeSignConfig**.

2. Drop a **GetAdobeSignConfig** activity below the PrintUrlToPdf activity.

   Specify the properties for the activity as follows:

   - AdobeSignConfig = **AdobeSignConfig**
   - Validation Messages = **formInstance.ValidationMessages**

The GetAdobeSignConfig activity retrieves your user credentials from the Adobe Sign settings in Forms Builder. These values enable the workflow to log in to Adobe Sign.

3. Create the following variables:

Variable Name	Type	Scope	Default
AdobeSignRecipient	Cmc.Nexus.FormsBuilder.Entities.AdobeSignRecipient	StateMachine	New AdobeSignRecipient
AdobeSignRecipient2	Cmc.Nexus.FormsBuilder.Entities.AdobeSignRecipient	StateMachine	New AdobeSignRecipient
AdobeSignDocument	Cmc.Nexus.FormsBuilder.Entities.AdobeSignDocument	StateMachine	New AdobeSignDocument
AdobeSignRequest	Cmc.Nexus.FormsBuilder.Entities.AdobeSignRequest	StateMachine	N/A

4. Below the GetAdobeSignConfig activity, drop an **Assign** activity for each row in the following table and type the indicated values:

"To" Field	Value	Notes
AdobeSignConfig.EmailSubject	"Adobe Sign Test at " + DateTime.Now.ToString()	This is the email that the end user will receive after the signing process is done.
AdobeSignDocument.DocumentId	"1"	The SignerId should match the Signer property in the Adobe Sign component on the form. Allowed values are "1" to "5".
AdobeSignDocument.Name	"StudentInfoPdf.pdf"	Name of the signed document.
AdobeSignDocument.Content	Pdf	Name of the variable that holds the document image (refer to Create a PDF of the Form). In our example the name of the variable is **Pdf**.
AdobeSignRecipient.FirstName	studentEntity.FirstName	First name of the user who submitted the signed document.
AdobeSignRecipient.LastName	studentEntity.LastName	Last name of the user who submitted the signed document.
AdobeSignRecipient.Email	studentEntity.EmailAddress	Email address of the person who will receive the signed Adobe Sign document.

"To" Field	Value	Notes
AdobeSignRecipient2.FirstName		First name of the co-signer who submitted the signed document. This could be a parent name from the StudentRelationshipAddressEntity.
AdobeSignRecipient2.LastName		Last name of the co-signer who submitted the signed document. This could be a parent name from the StudentRelationshipAddressEntity.
AdobeSignRecipient2.Email		Email address of the co-signer who will receive the signed Adobe Sign document. This could be the email address retrieved from the StudentRelationshipAddressEntity.
AdobeSignRecipient2.SignerId	"2"	The SignerId should match the Signer property in the Adobe Sign component on the form. Allowed values are "1" to "5".

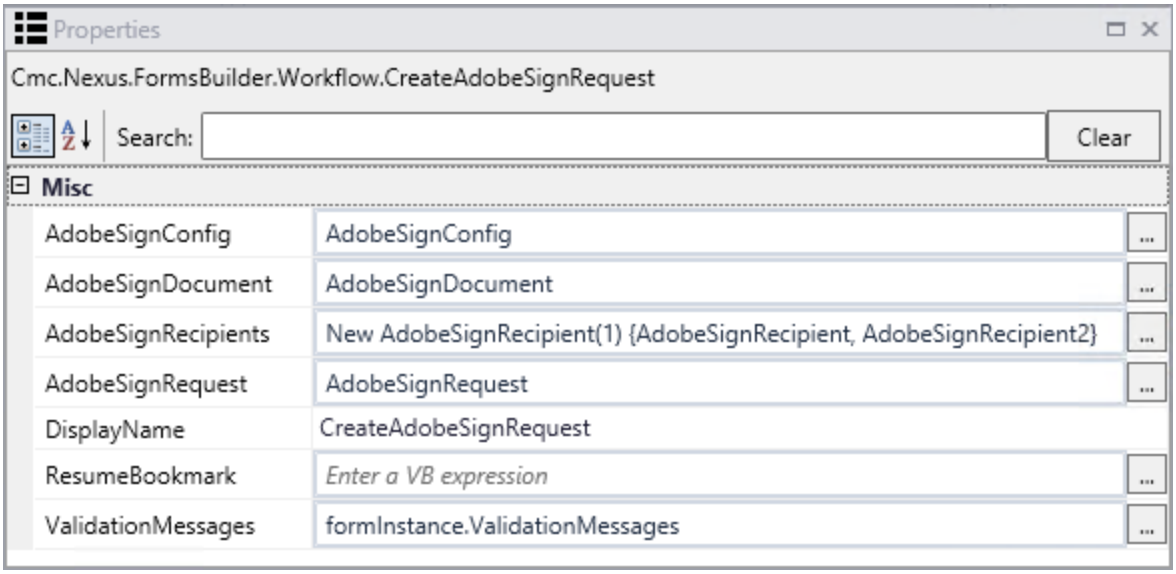
5. **Save** the workflow locally and continue with the next set of steps.

Create the Adobe Sign Request and Specify the IFrame URL

In the following steps we will continue to add activities to the *Default-Frame* state.

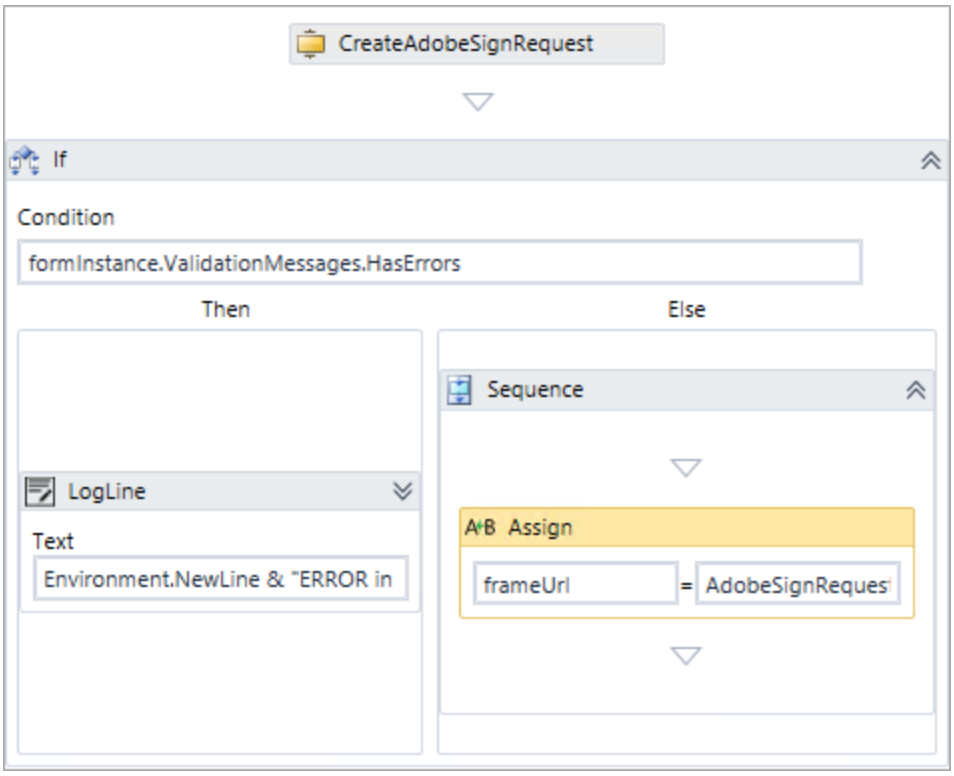
1. Drop the **CreateAdobeSignRequest** activity below the previous activity.

   Specify the properties using the names of the variables created above as follows:



Notes

- The expression for AdobeSignRecipients passes multiple items (the primary and secondary recipient) within an array.
- The out argument AdobeSignRequest returns the URL of the signed document.



2. Drop an **If** activity below the CreateAdobeSignRequest activity and specify the following condition:

   formInstance.ValidationMessages.HasErrors

3. Add a **LogLine** activity to the *Then* branch and specify a message to capture errors, e.g.:

   Environment.NewLine & "ERROR in CreateAdobeSignRequest: " & Environment.NewLine & formInstance.ValidationMessages.Item(0).Message

4. Add an **Assign** activity to the *Else* branch.
   a. In the *To* field, specify **frameUrl**. (This is the argument associated with the IFrame form.)

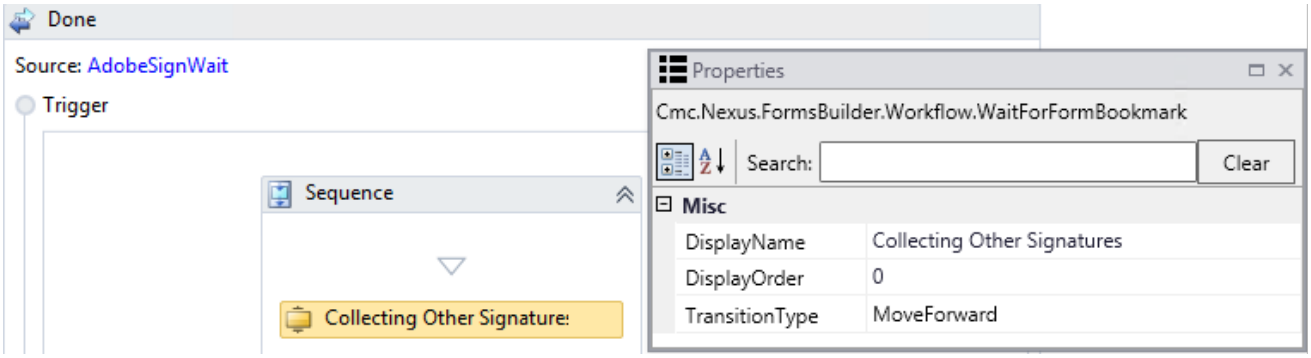
      Be sure to use the exact casing shown here.

   b. In the *Value* field, specify **AdobeSignRequest.Url**.
5. **Save** the workflow locally and continue with the next set of steps.

Check Signature Status and Receive Signed Document

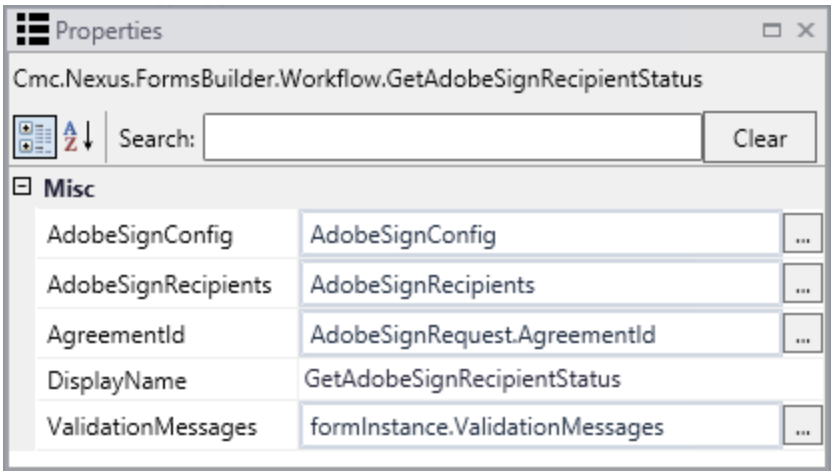
The IFrame form (*Default-Frame*) will receive the Adobe Sign document with the first signature, but the signing process is not yet complete. We want the workflow to continue with the *AdobeSignWait* form upon receipt of the first signature. The workflow will pause on the *AdobeSignWait* form until the second signature is received. If user closes the form or logs out after submitting the first signature, upon return to the form, the workflow will display the *AdobeSignWait* form.

1. Double-click the **Done** transition leading to the *End* state.
2. In the Trigger area, name the WaitForFormBookmark as **Collecting Other Signatures**.
3. Select **Transition Type = MoveForward** on the WaitForFormBookmark activity.



4. Drop a **GetAdobeSignRecipientStatus** activity below the WaitForFormBookmark activity.

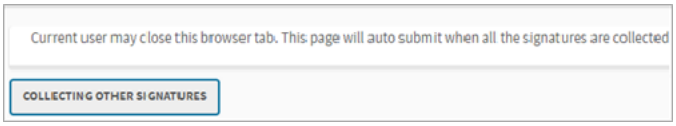
   Specify the properties using the names of the variables created above as follows:

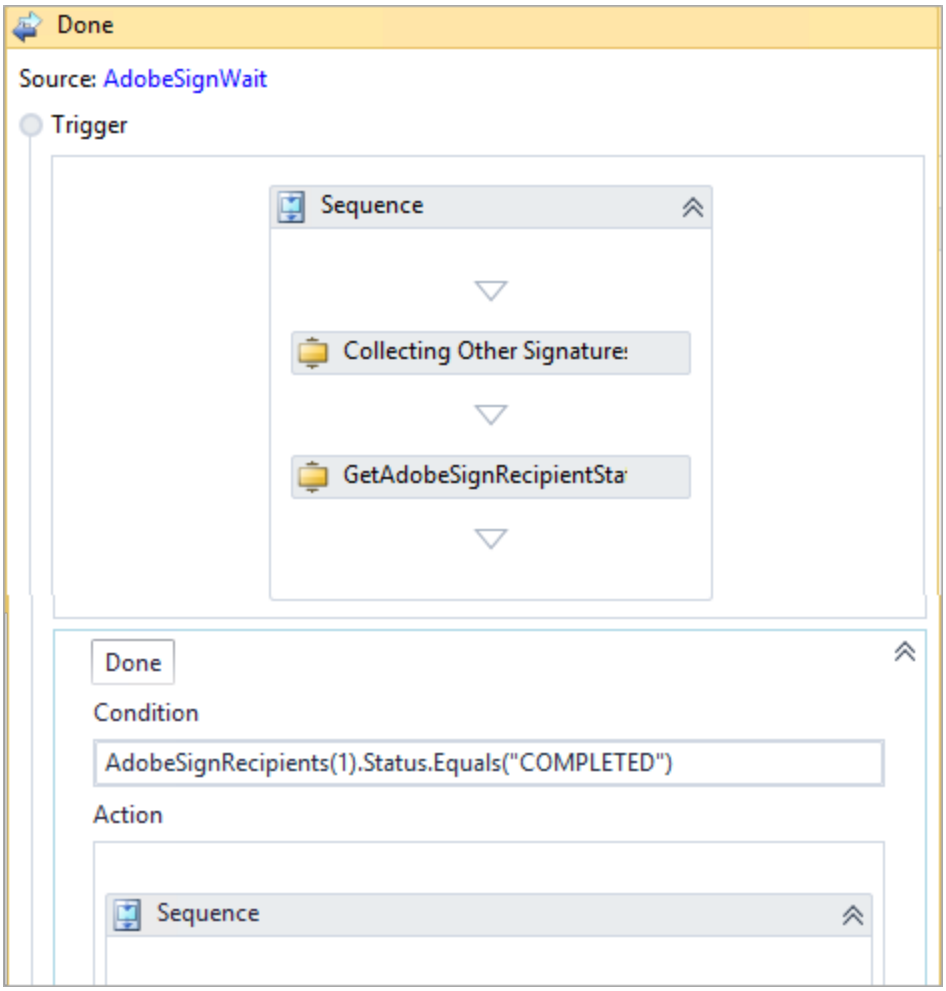


5. Specify the following in the Condition field for the Done state:

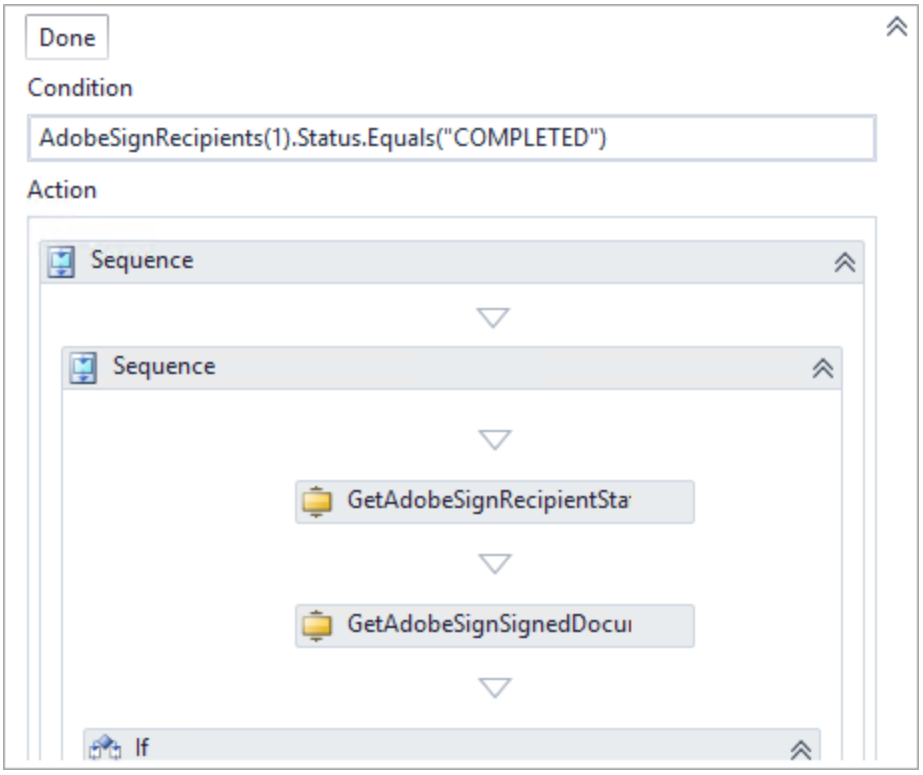
   AdobeSignRecipients(1).Status.Equals("COMPLETED")

   This ensures that even if the user selects the button ”Collecting Other Signatures" instead of closing the browser, the workflow will not proceed until the status of the additional signer is COMPLETED.

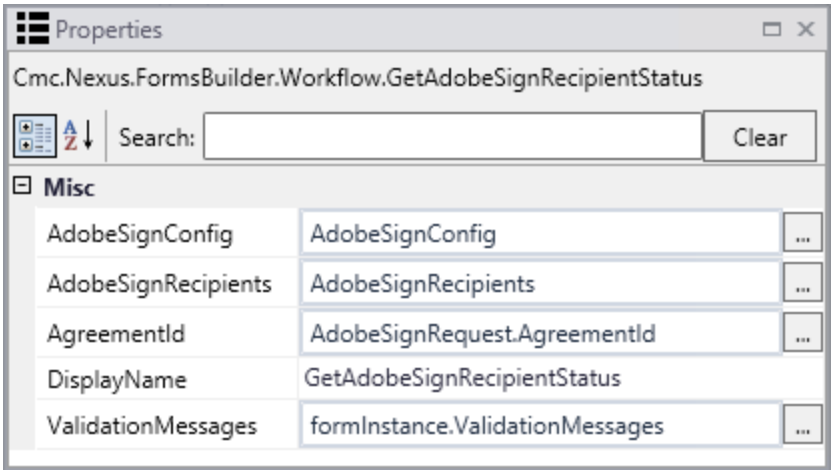




6. Drop a **GetAdobeSignRecipientStatus** activity into the sequence.

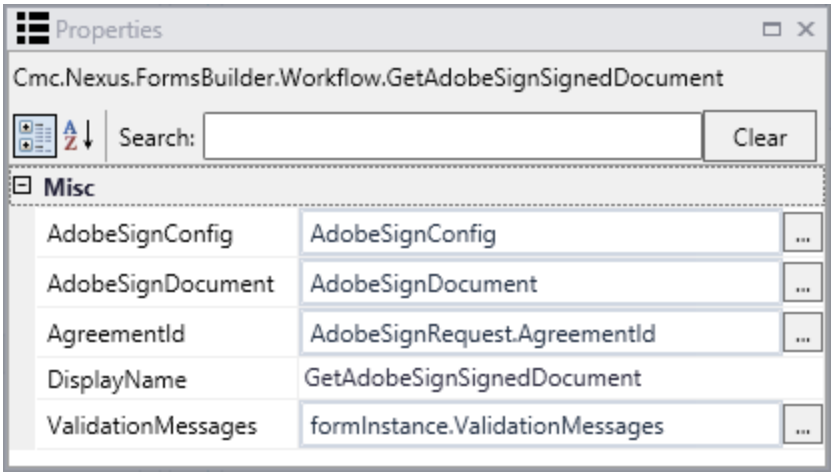


Specify the properties using the names of the variables created above as follows:



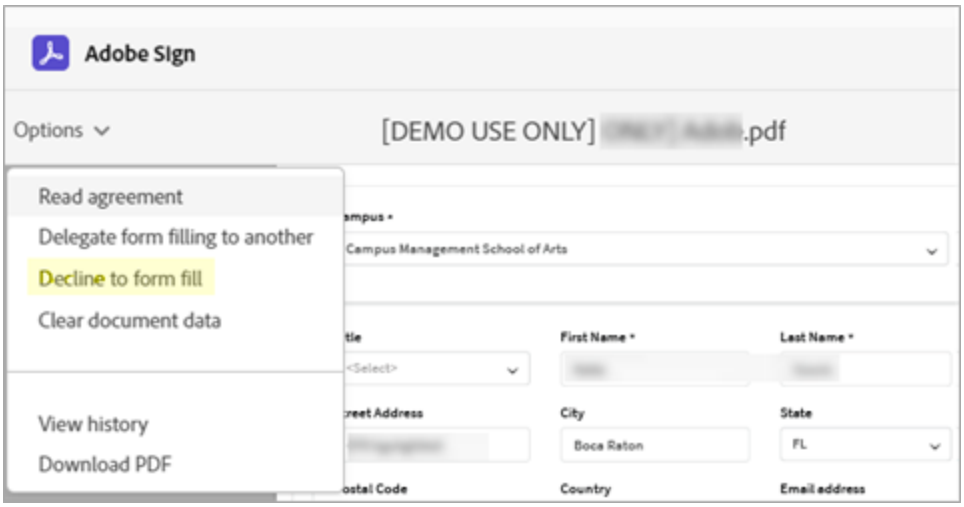
7. Create a variable named **AdobeSignSignedDocument**. In the Variable type field, select **AdobeSignDocument**. Set the Scope to **StateMachine**.
8. Drop a **GetAdobeSignSignedDocument** activity below the GetAdobeSignRecipientStatus activity.

   Specify the GetAdobeSignSignedDocument properties as follows:



9. **Save** the workflow locally and continue with the next set of steps.

Adobe Sign provides signers the **Decline to form fill** option.



If the decline option is selected by the primary signer:

- The signer will access a webpage (IFrame) stating "*You declined to sign <PDF name>.*"
- The signer will receive an email stating that Adobe Sign terminated the agreement exchange.
- Adobe Sign will not return control of the sequence to Forms Builder and the workflow will not continue. The workflow will remain as a persisted instance. You will need to terminate persisted workflows periodically using either Sequence Designer (refer to Delete Persisted Workflow Instances) or Workflow Composer (refer to Persisted Workflows).

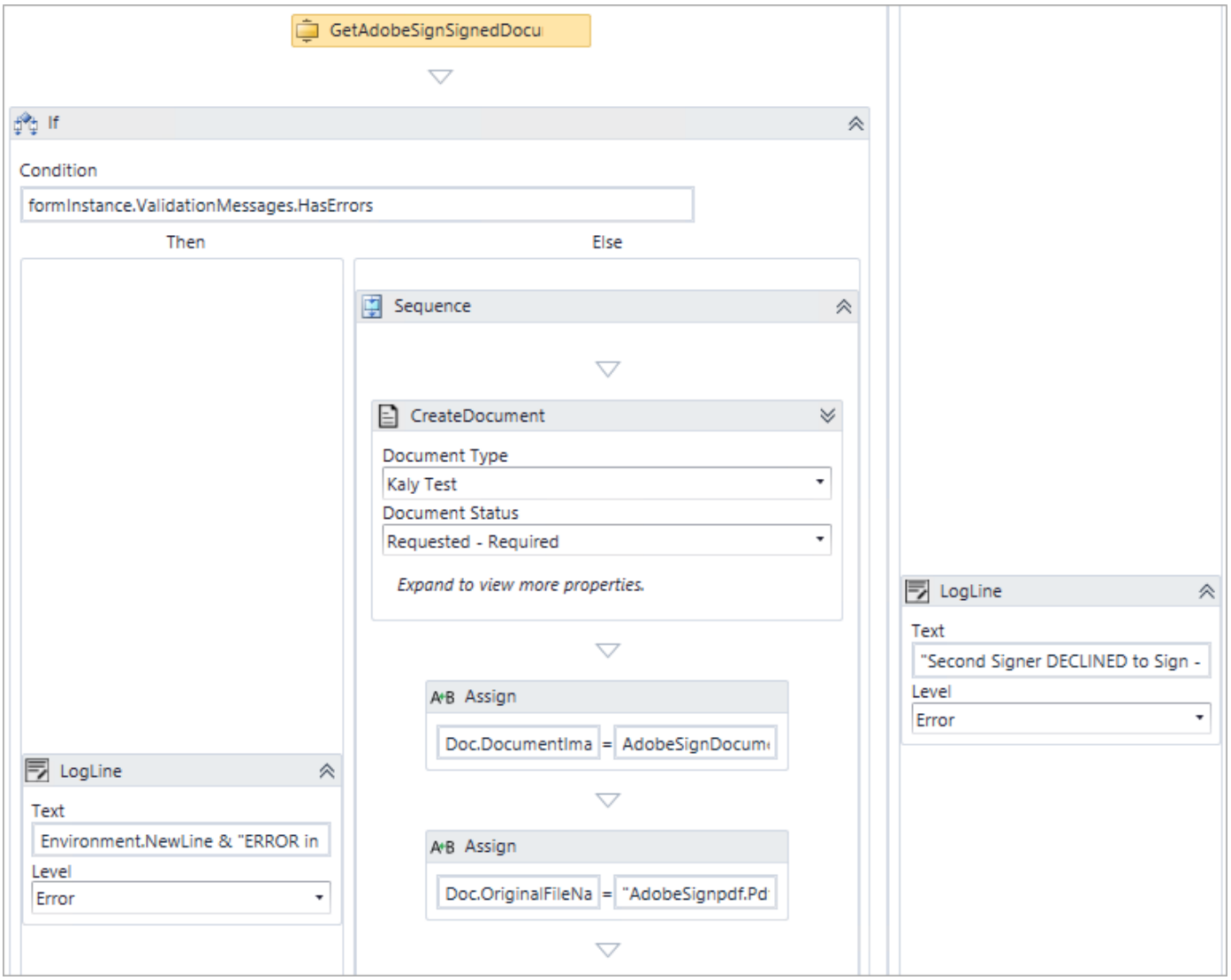
If the decline option is selected by the secondary or any additional signer:

- The signer will access a webpage (IFrame) stating "*You declined to sign <PDF name>.*"
- The primary signer will receive an email stating that Adobe Sign terminated the agreement exchange.

- Adobe Sign will return control of the sequence to Forms Builder and the workflow will continue. You can use the GetAdobeSignRecipientStatus activity to determine status of signers. If the status is “Cancelled”, the workflow can handle the sequence accordingly, e.g., with notification to a staff member.

Create and Save the Document in Anthology Student

To convert an Adobe Sign document to a DocumentEntity that can be attached to a record in Anthology Student, add CreateDocument and SaveDocument activities to the workflow. These activities will be placed below the GetAdobeSignSignedDocument activity.



1. Drop an **If** activity below the GetAdobeSignSignedDocument activity and specify the Condition as **formInstance.ValidationMessages.HasErrors**.
2. Add a **LogLine** activity to the *Then* branch using the following message Text:.

Environment.NewLine & "ERROR in GetSignedDocuments: " & formInstance.ValidationMessages.Item(0).Message

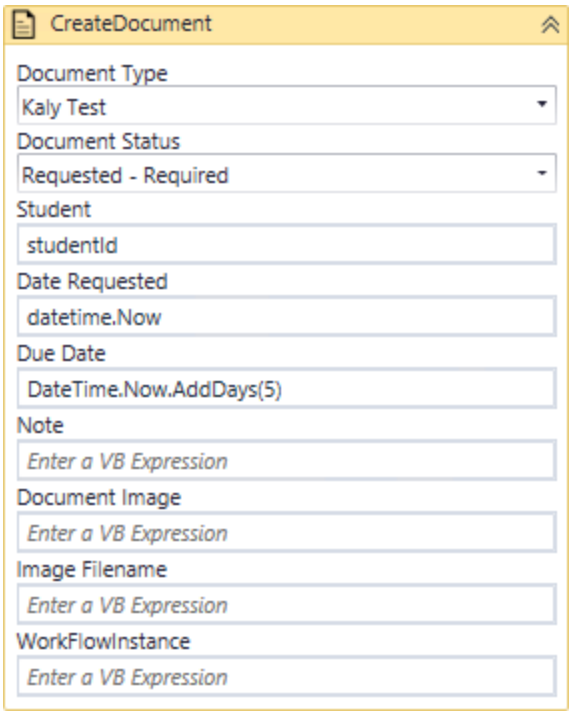
3. Create a variable named **Doc**. Set the Scope to **StateMachine**.

   In the Variable type field, select **Browse for Type**. In the "Browse and Select a .Net Type" window, scroll down to Cmc.Nexus.Crm.Contracts > Cmc.Nexus.Crm.Entities and select **DocumentEntity**.

4. Drop a **CreateDocument** activity into the *Else* branch.

   Specify the properties as shown below activity and use the drop-down field in the activity to select appropriate values:

   - Module = Select a Module from the drop-down list, e.g., Admissions.
   - Document Type = Select a Type (Template) from the drop-down list.
   - Document Status = Select a Type from the drop-down list.
   - Student = Specify a Student Id or use a variable.
   - Due Date = Specify a date or use a variable, e.g., DateTime.Now
   - Document (OutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
   - Validation Messages = formInstance.ValidationMessages



5. Below the CreateDocument activity, drop an **Assign** activity for each row in the following table and type the indicated values:

"To" Field	Value
Doc.OriginalFileName	"AdobeSignPdf.Pdf"
Doc.DocumentImage	AdobeSignSignedDocument.Content
Doc.ImageType	"Pdf"
Doc.IsDocumentAddedManually	true

6. Drop a **SaveDocument** activity below the last Assign activity.

   Specify the properties for the SaveDocument activity as follows:

   - Document (InOutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
   - Validation Messages = **formInstance.ValidationMessages**

7. **Save** the workflow locally and continue with the next set of steps.

Final Steps

1. Select **Publish**. The New Workflow Definition Version window is displayed.
2. If you want the workflow to be run as soon as the event occurs on the entity, select **Enable This Workflow Version?**, otherwise leave the check box cleared.
3. Select **Save**, then **Cancel** to close the publisher window.
4. In the **Sequence List**, select and complete the form that contains the Adobe Sign fields.
5. Verify that the signed Adobe Sign document is available at the return URL.

While testing your workflow, make sure that you terminate the previous instance of the workflow before running an updated version of the same workflow. In Workflow Composer, select **Open Persisted Workflow**, select your workflow instance, and select **Terminate**.

Persisted workflow instances can be also deleted from the Sequence Designer workspace. For more information, refer to Delete Persisted Workflow Instances.

DocuSign Forms

You can configure Forms Builder to request signatures from students in DocuSign as they are filling out forms. Signatures can be requested for one or many Forms Builder forms or any other documents supported by DocuSign. DocuSign integration requires values to be configured in Form Designer. For more information, refer to DocuSign Settings.

The integration of Forms Builder and DocuSign is out of the box functionality that is available once Forms Builder is installed. No additional integration is needed. Customers will have to configure DocuSign as described in Move from Test to Production for DocuSign to work with Forms Builder.

For information about supported file formats and limitations, refer to https://support.docusign.com/guides/ndse-user-guide-supported-file-formats.

DocuSign Settings

When Forms Builder is installed, it is installed without an account for DocuSign. Customers can create a free developer account (sandbox) for test purposes at the following URLs: https://go.docusign.com/o/sandbox/ or https://www.docusign.com/developer-center.

Sandbox accounts operate in a DocuSign demo environment, which is identical to production except that documents sent through demo are not legally binding and have test stamps on them. Sandbox accounts do not expire and have enterprise level features enabled by default that allow you to test all scenarios before going live.

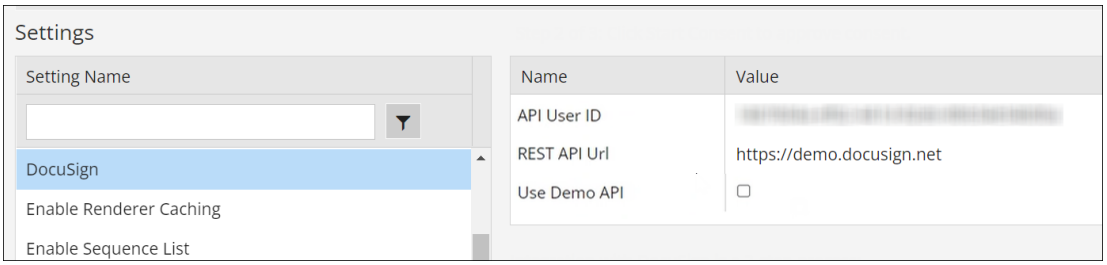
Customers wishing to integrate with DocuSign need to obtain their own licensing information with DocuSign to use in a live environment. Additionally, other required OAuth framework settings need to be updated in the Properties pane of the Settings screen.

Important: Do not use your live account credentials in test mode.

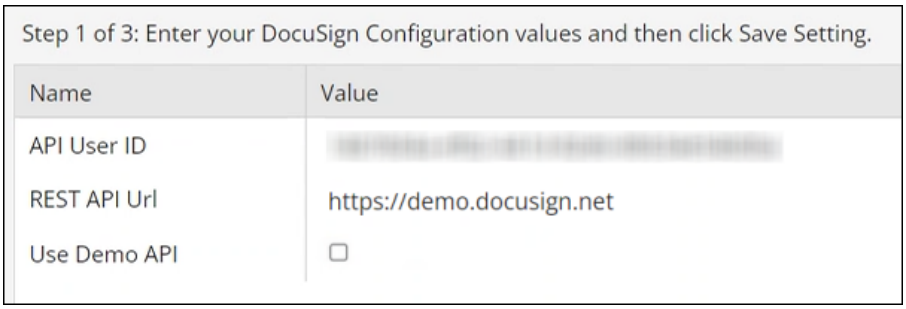
Setting up Forms Builder for DocuSign

Perform the following steps in a new implementation and in an upgrade scenario.

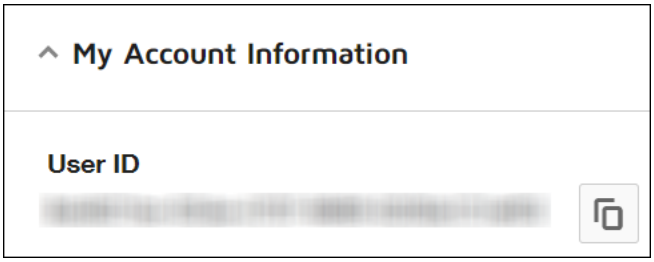
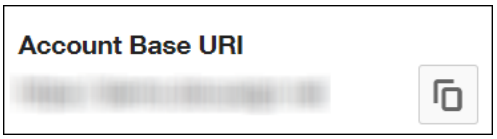
1. On the Forms Builder home page, select the **Settings** tile and select **DocuSign** in the Settings pane. The Properties pane is populated with DocuSign properties.



2. In Step 1 of 3, update the following fields in the Properties pane:



DocuSign Properties

Name	Value
API User ID	Log on to DocuSign at https://admin.docusign.com/, and on the Apps and Keys page in DocuSign (Settings > Apps and keys), navigate to your account and select to copy the User ID value.  Paste the copied value into this field.
Rest API Url	https://demo.docusign.net This the default DocuSign URL that can be used in a demo environment. Alternatively, in a production environment, on the Apps and Keys page in DocuSign (Settings > Apps and keys), navigate to your account and select to copy the **Account Base URI** value.  The DocuSign e-Signature REST API lets you e-sign documents, request signatures, and automate your forms and data. This URL may vary based upon the region of your account. Customers can contact their DocuSign representative if they are not sure about the base URL for their account. The following link from DocuSign help provides different options: https://docs.docusign.com/. **Note**: In an upgrade scenario, note the value of the **Rest API URL** from the previous version. Copy its value into this field in version 4.1.
Use Demo API	By default, this check box is cleared which means production integration APIs will be saved in the database. Select the check box to use demo integration APIs.

3. Select **Save Setting**.

   After the page is refreshed, select **DocuSign Settings** in the left pane to view Step 2.

4. In Step 2 of 3, the **Use Demo API** field will be hidden and other fields will no longer be editable as their values were saved in the previous step.

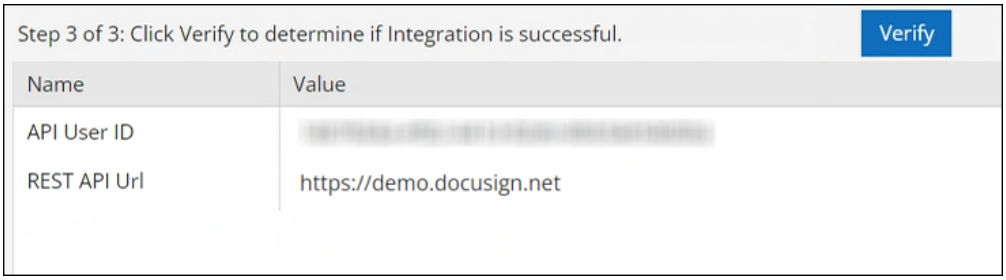
   Select **Start Consent**.



The API User ID specified above will be consumed to impersonate the user logged on to DocuSign. You will be redirected to DocuSign to enable consent. When consent is obtained, focus will return to Forms Builder.

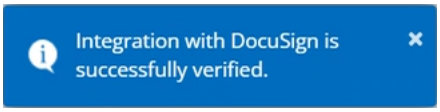
Select **DocuSign Settings** in the left pane to view step 3.

5. In Step 3 of 3, select **Verify** to complete the integration process with DocuSign.



In the background, the value of the Account ID will be fetched from DocuSign and stored in Forms Builder.

Once integration with DocuSign is successful, the following message will be displayed:



6. Specify the **DocuSign Error Message Text** in the Settings panel.

   Select 

to clear saved integration settings. For example, administrators can clear configurations of a demo account and then type information specific to a production account.

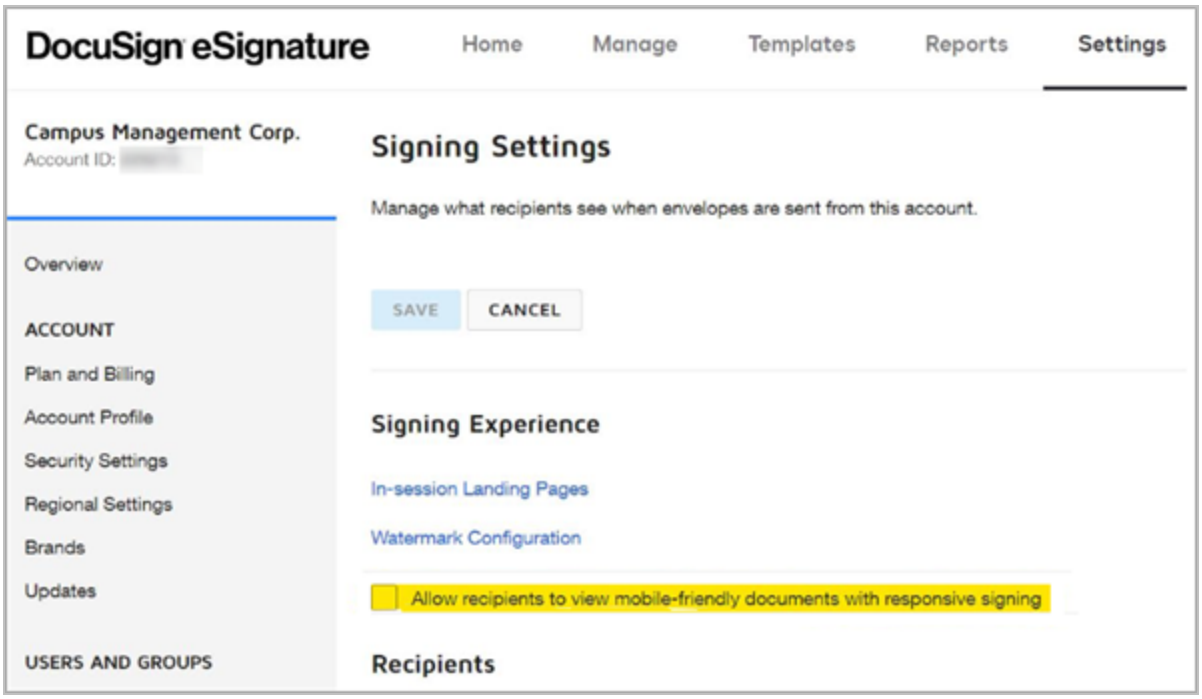
   After configurations are cleared, the integration process must be repeated from step 1.

7. **Save** the setting.
8. Once the DocuSign configuration is complete, reset IIS or wait an hour for the changes to take effect.

Responsive Signing

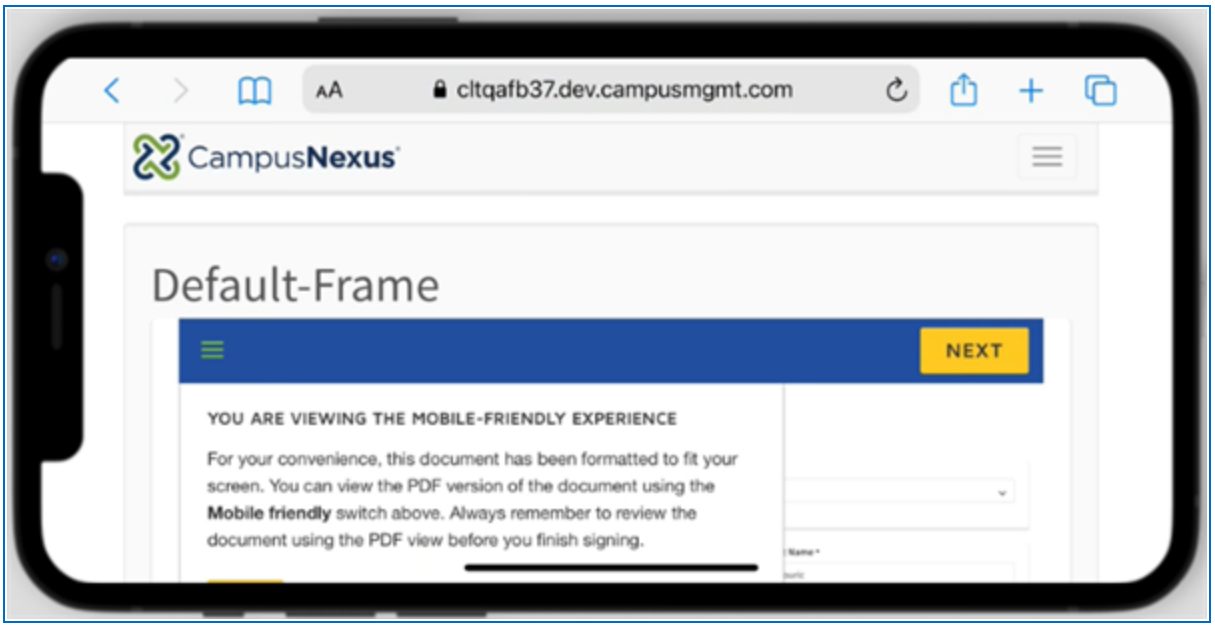
DocuSign supports responsive signing based on the signer's device type (mobile, tablet). Responsive signing automatically converts documents to HTML and allows users to preview the conversion across device types prior to sending. Signers will access the mobile-friendly HTML and can toggle to view the original document.

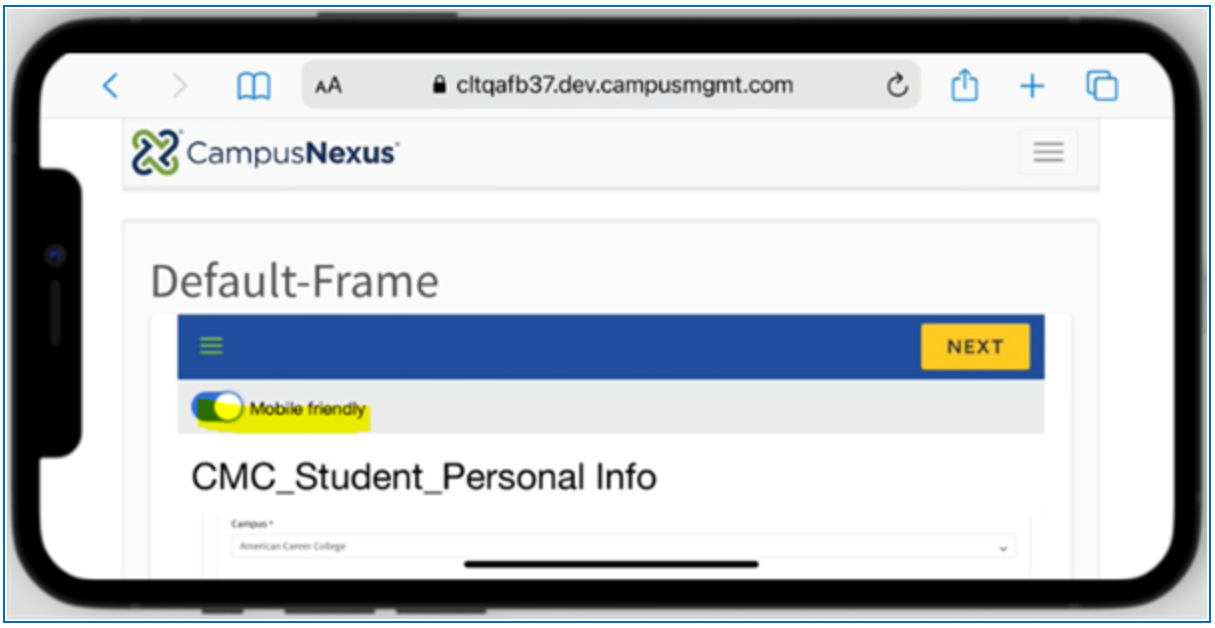
The responsive signing feature is disabled by default and must be enabled by an account administrator. To enable it, log in to your DocuSign account, select **Settings** > **Allow recipients to view mobile-friendly documents with responsive signing**.

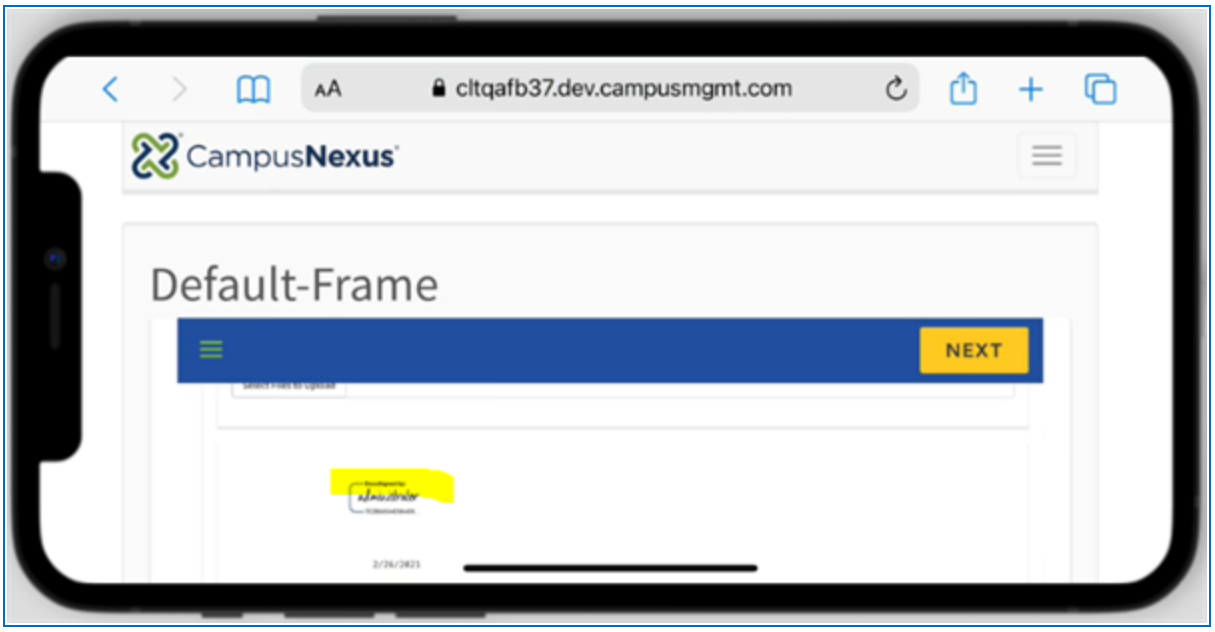


For more information, refer to https://developers.docusign.com/docs/esign-rest-api/esign101/concepts/responsive/.

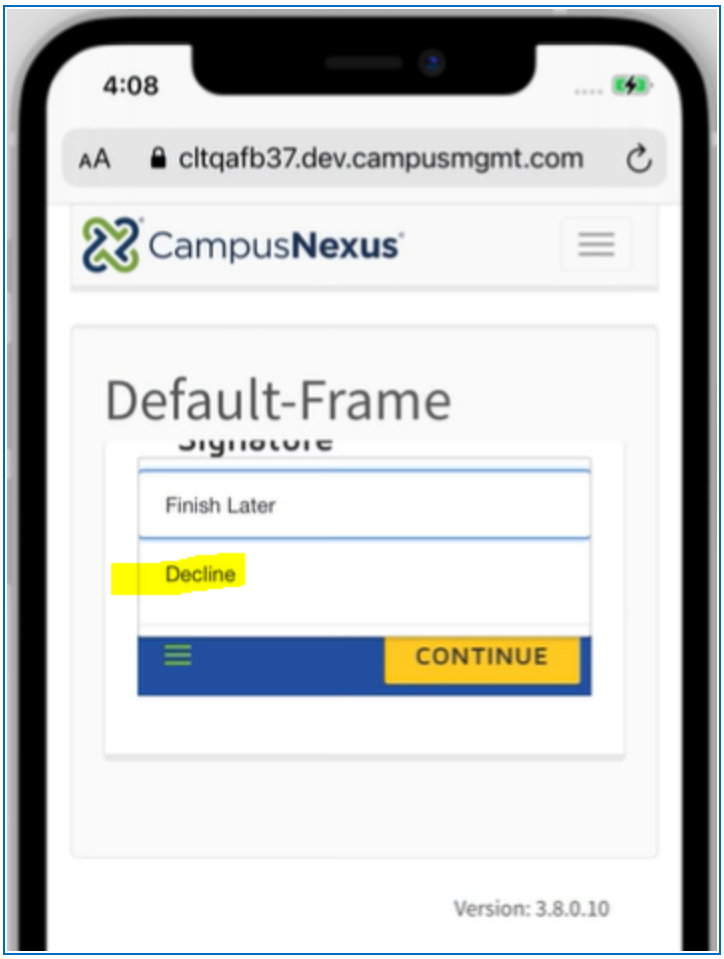
Users will be givin instructions about switching to the PDF view. They can tap the Next buttons to navigate through the document, sign it and submit it.

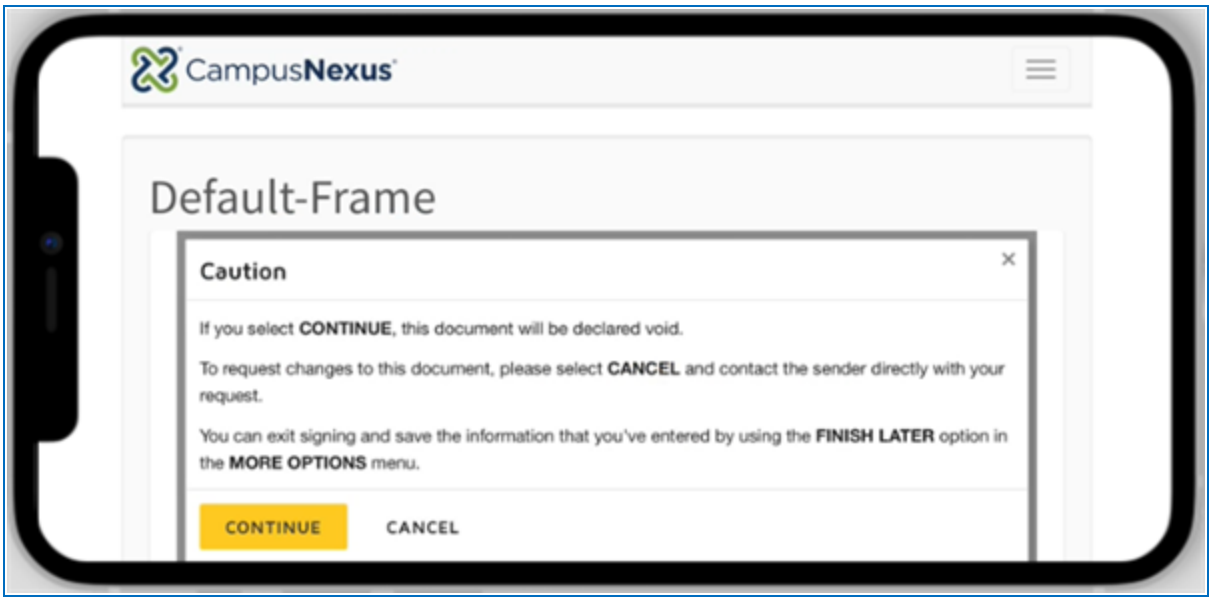






The Decline option is available on the first 'Continue' screen. When Decline is selected, a 'Caution' note indicates that the document will be declared void.


4:08
AA
cltqafb37.dev.campusmgmt.com
CampusNexus
Default-Frame
Finish Later
Decline
CONTINUE
Version: 3.8.0.10


CampusNexus
Default-Frame
Caution
If you select CONTINUE, this document will be declared void.
To request changes to this document, please select CANCEL and contact the sender directly with your request.
You can exit signing and save the information that you've entered by using the FINISH LATER option in the MORE OPTIONS menu.
CONTINUE
CANCEL

DocuSignWorkflow Sample - Single Signer

The following procedure details a sample workflow and forms for a sequence that contains single signer.

Prerequisites

1. The DocuSign properties are configured in Forms Builder. Refer to DocuSign Settings.
2. A form sequence with the necessary DocuSign components for a single signer is created. In our example, the **Anonymous** property of the sequence is set to **false**.

   If you create a non-authenticated sequence (Anonymous=true), make sure that the user's email address is included in the sequence. The email address is required as the DocuSign recipient address.
3. A form that contains an IFrame component is created. In our example, this form is named *Default-Frame* with the following properties (case sensitive):
   - Name = **docuSignFrame**
   - Url = **{{vm.models.frameUrl}}**

   The IFrame form can be part of the initial sequence; however, in the procedure below, we are adding the form to the workflow.

   > ⚠ We recommend that you copy the original *Default-Frame* form, edit the copy, and use it in your sequences. Save a backup copy of your form.

To optimize the display of your e-signature forms on mobile devices, you may want to add the CSS described in this Note.

Enhancements in Forms Builder 3.6

The DocuSign component provides additional values on the Type property: Approve, Attachment, Checkbox, Company, Date, Decline, Email, Email Address, Envelope Id, Number, Ssn, and Text.

The DocuSign component provides an automatic transition (auto-redirect) from the Default-Frame form to the confirmation form after a successful DocuSign session. The auto-redirect obsoletes the "DocuSign Confirmation Message Text" setting.

The auto-redirect depends on a forward direction in the WaitForFormBookmark activity in the transition after the DocuSign redirect state (typically Default-Frame), in particular if DisplayName has been modified.

- If there is only a single button and DisplayName has been customized but Transition Type was left as "Default", the auto-redirect moves forward to next form state.
- If there are two buttons and DisplayName(s) have been customized but Transition Type was left as "Default", the auto-redirect will assume the rightmost button (alphabetically last) is the transition for next state.

Best Practice is always to specify Display Order and Transition Type (“MoveForward” or “MoveBack”) when button Display Name(s) have been customized so behavior is known. The Transition Type of “Default” was kept for compatibility for forms built prior to Transition Type being available on WaitForFormBookmark with default Display Names "Next" and "Back”.

Create the Workflow

Open the Workflow and Review Arguments, Variables, and Logging Requirements

1. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
2. In Workflow Composer, drag the State icons and Transition lines to so that you can easily locate each item in the StateMachine workflow.

The workflow requires the following arguments and variables. You can create these arguments and variables before working on the activities, or you can add them when they are needed for a specific activity (as described below).

> ⚠ Keep in mind that arguments are passed in JSON format and that JSON elements are case sensitive. **Be sure to match the casing of argument names in Workflow Composer and Form Designer.**

Arguments

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*
studentEntity	In/Out	StudentEntity	*Default value not supported*
frameUrl	In/Out	String	*Default value not supported*

Variables

Name	Variable type	Scope	Default
studentId	Int32	Welcome	*Enter a VB expression*
renderedFormImage	String	StateMachine	*Enter a VB expression*
Pdf	Byte[]	StateMachine	*Enter a VB expression*
DocuSignConfig	DocuSignConfig	StateMachine	*Enter a VB expression*
DocuSignDocument	DocuSignDocument	StateMachine	New DocuSignDocument
DocuSignRecipient	DocuSignRecipient	StateMachine	New DocuSignRecipient
DocuSignRequest	DocuSignRequest	StateMachine	*Enter a VB expression*
SignedDocument	DocuSignDocument	StateMachine	*Enter a VB expression*
Doc	DocumentEntity	StateMachine	*Enter a VB expression*
URL	String	StateMachine	*Enter a VB expression*

LogLine Activities

While testing and troubleshooting the workflow, we recommend adding LogLine activities at critical stages within the workflow. The following expression will provide logging for specific objects (replace <object> with the object name):

Newtonsoft.Json.JsonConvert.SerializeObject(**<object>**, Newtonsoft.Json.Formatting.Indented)

For example, to capture errors related to the CreateDocuSignRequest activity, you would place a LogLine activity with the following expression below the CreateDocuSignRequest activity.

Newtonsoft.Json.JsonConvert.SerializeObject(**DocuSignRequest**, Newtonsoft.Json.Formatting.Indented)

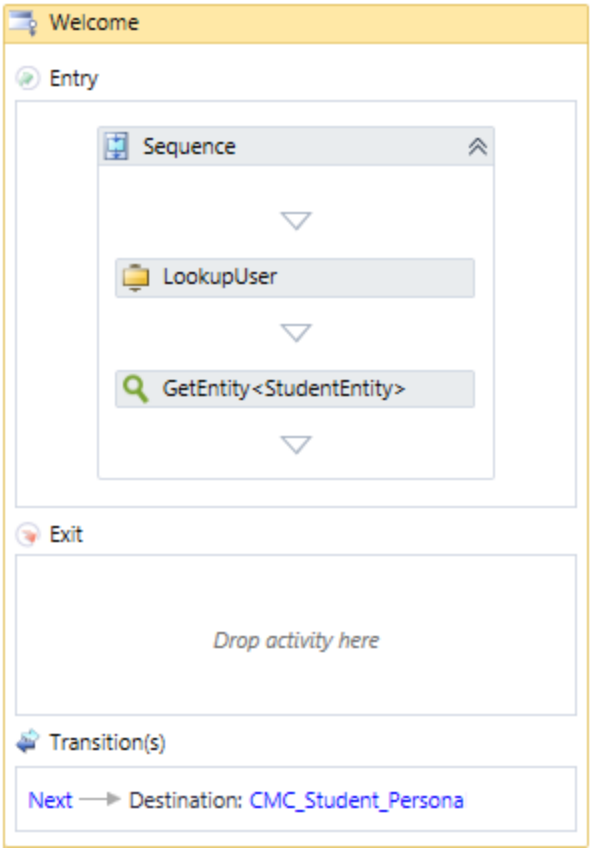
We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

Capture the Login User Information

> If your sequence is non-authenticated (Anonymous=true), the LookupUser and GetEntity activities are not applicable, but a CreateEntity activity may be needed instead. Be sure that any anonymous sequence includes the user's email address. It is required as the DocuSign recipient address.

The first step is to capture the information of the user who logged in to the form.

1. Create a variable named **studentId** of Type **Int32**.
2. Drop a **LookupUser** activity in the Entry area of the Welcome form.
   - In the UserId property, specify **studentId**. This is the variable created in the previous step.
   - In the UserName property, specify **formInstance.UserName**. The user name is retrieved from the login account information.



3. Drop a **GetEntity** activity in the below the LookupUser activity.
   - In the EntityId property, specify **studentId**.
   - In the Result property, specify **studentEntity**.

Workflow Composer automatically wraps the activities in a Sequence.

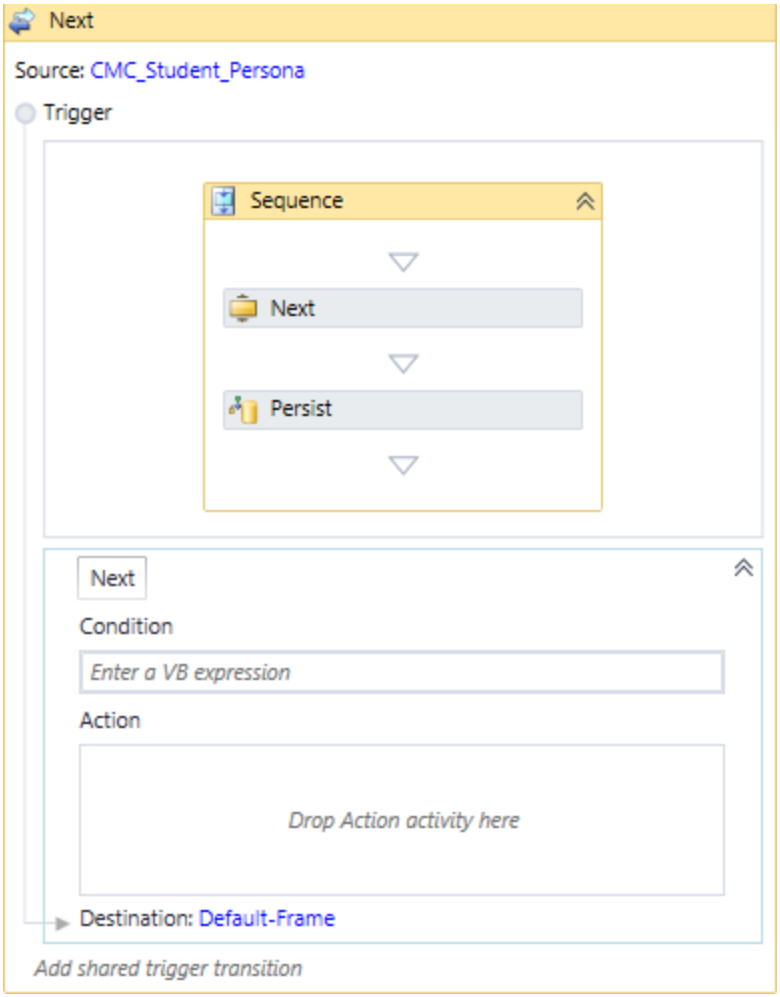
4. **Save** the workflow locally and continue with the next set of steps.

Persist the Workflow Instance After Collecting the Form Data

Since the workflow can be busy for a long time, we recommend that you persist the workflow instance before creating the PDF.

1. Double-click the **Next** transition from the form that is used to collect the data, *CMC_Student_Personal Info*.
2. Drag a **Persist** activity into the Trigger area below the Next activity. The Persist activity does not require any properties to be specified.

   Workflow Composer automatically wraps the Next and Persist activities in a Sequence.



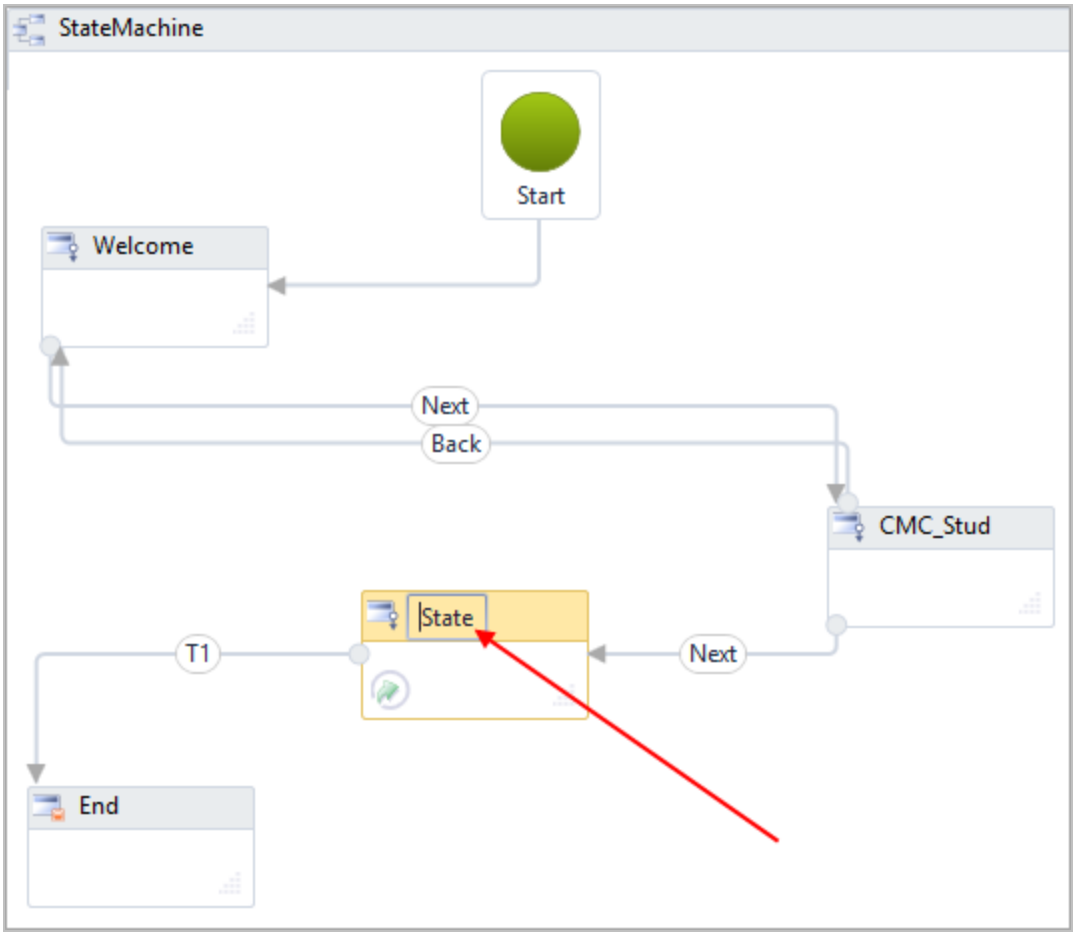
3. **Save** the workflow locally and continue with the next set of steps.

Add an IFrame Form to the Workflow

1. In Workflow Composer, select the **Arguments** tab.
2. Create an argument named **frameUrl**. Set the Direction to **In/Out**. Set the Type to **String**.

   > ⚠ Be sure to use the exact casing shown here.

3. Select the **StateMachine** link in the Designer pane.
4. Drop a **State** activity onto the transition line after the Next transition from the form that contains the DocuSign components (*CMC_Student_Personal Info*).



5. Change the default name of the State activity to the name of the form that contains the IFrame. In our case, the name is **Default-Frame**.
6. **Save** the workflow locally and continue with the next set of steps.

Create a PDF of the DocuSign Form

In the following steps we will create a set of activities in the *Default-Frame* State, which represents the IFrame form.

1. Double-click the State icon of the form that contains the IFrame component.
2. Create a variable named **URL**. Set the Variable type to **String**. Set the Scope to **StateMachine**. This variable will be assigned to the PDF that will be created from the form.

   As of Forms Builder 3.5, the URL input argument for the PrintUrlToPdf activity is optional. When the URL is not specified, the activity constructs the URL for all forms traversed in the sequence.

3. Drop an **Assign** activity into the Entry area of the State.
   a. In the *To* field, specify the variable name **URL**.
   b. In the *Value* field, specify the following:

      formInstance.RendererBaseUrl + "#/viewCreator/" + formInstance.WorkflowDefinitionId.ToString + "/forms=CMC_Student_Personal+Info"

      Where "/forms=*CMC_Student_Personal+Info*" indicates the Forms Builder form that contains the DocuSign component.

      Notes

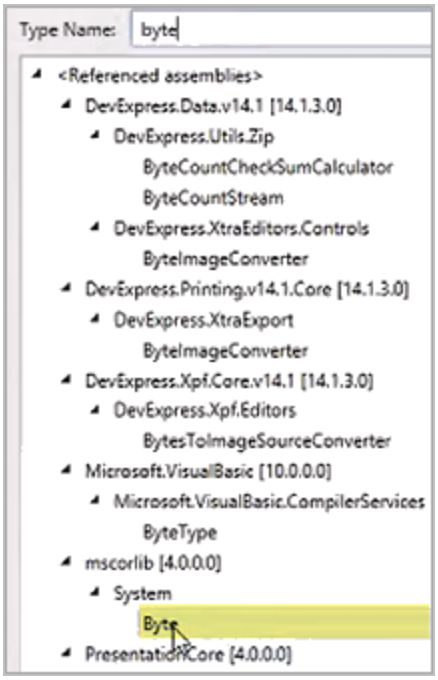
      - If the form name has a space, replace the space with a + sign as shown in the example: *CMC_Student_Personal+Info*
      - If multiple forms are sent to DocuSign, specify a comma-separated list of form names in the "/forms=" attribute.

4. Drop a **Persist** activity below the Assign activity.
5. Create a variable with a name like **Pdf**. Set the Scope to **StateMachine**. This variable will hold the document image created from the form.

   Make sure you use the same variable name when you reference this variable later in the workflow. (This applies to any other variable.)

   a. In the Variable type field, select **Array of [T]** and select **Browse for Type**.
   b. In the "Browse and Select a .Net Type" window, specify "**byte**", select the System variable "**Byte**", and

select **OK**.



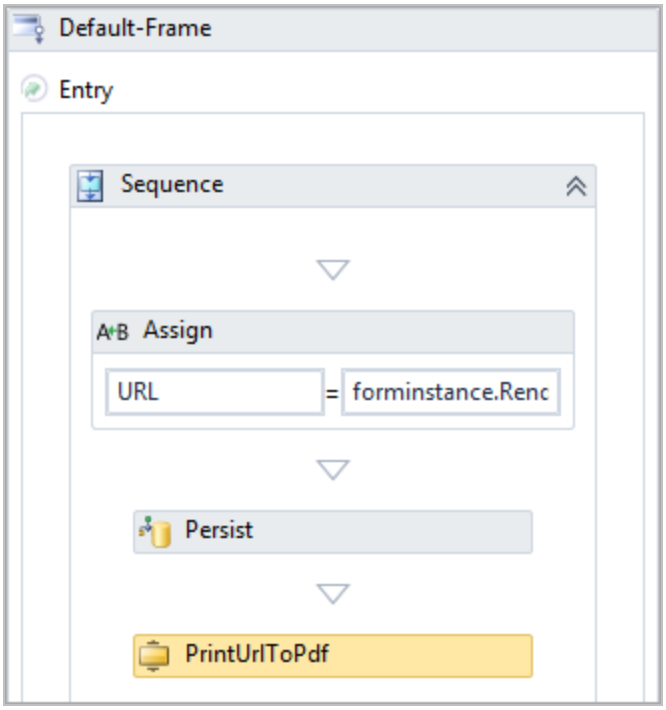
6. Drop a **PrintUrlToPdf** activity below the Persist activity.

   Specify the properties for the activity as follows:

   - PdfDocument = **Pdf** (This is the name of the variable created above.)
   - Url = **URL** (This is the name of the variable created above.)

     As of Forms Builder 3.5, the URL input argument for the PrintUrlToPdf activity is optional. When the URL is not specified, the activity constructs the URL for all forms traversed in the sequence.

   - Validation Messages = **formInstance.ValidationMessages**



7. **Save** the workflow locally and continue with the next set of steps.

Get the DocuSign Configuration and Pass the Recipient Information

In the following steps we will continue to add activities to the *Default-Frame* State, which represents the IFrame form.

1. Create a variable named **DocuSignConfig**. Set the Scope to **StateMachine**.

   In the Variable type field, select **Browse for Type**. In the "Browse and Select a .Net Type" window, scroll down to Cmc.Nexus.FormsBuilder.Contracts > Cmc.Nexus.FormsBuilder.Entities and select **DocuSignConfig**.

2. In the State of the IFrame form, drop a **GetDocuSignConfig** activity below the PrintUrlToPdf activity.

   Specify the properties for the activity as follows:

   - DocuSignConfig = **DocuSignConfig**
   - Validation Messages = **formInstance.ValidationMessages**

   The GetDocuSignConfig activity retrieves the User Name, Password, Integrators Key, and REST API Url from the DocuSign settings in Forms Builder. These values enable the workflow to log in to DocuSign.

   As of Forms Builder 3.4, the DocuSignConfig.TestMode assignment (=true or =false) is no longer supported or functional. Assign statements containing it can be deleted.

   To process the DocuSign request, the Email Subject and Return URL properties need to be assigned to the DocuSignConfig variable. These properties are required.

> ⚠ The only two properties for the DocuSignConfig object that should ever be modified in the workflow definition are **EmailSubject** and **ReturnUrl**. The values for all the other properties are retrieved from the DocuSign Settings saved in the database. Any modification done to the values for the other DocuSignConfig properties in the workflow definition will likely result in errors when the DocuSign portion of the sequence is executed.

3. Create the following variables:

Variable Name	Type	Scope	Default
DocuSignRecipient	Cmc.Nexus.FormsBuilder.Entities.DocuSignRecipient	StateMachine	New DocuSignRecipient
DocuSignDocument	Cmc.Nexus.FormsBuilder.Entities.DocuSignDocument	StateMachine	New DocuSignDocument
DocuSignRequest	Cmc.Nexus.FormsBuilder.Entities.DocuSignRequest	StateMachine	N/A

4. Below the GetDocuSignConfig activity, drop an **Assign** activity for each row in the following table and type the indicated values:

"To" Field	Value	Notes
DocuSignConfig.EmailSubject	"CMC Campus DocuSign Testing"	This is the email that the end user will receive after the signing process is done.
DocuSignConfig.ReturnUrl	formInstance.RendererBaseUrl+"#/docusigncomplete"	This is the URL that will display the document after the signing process is done.
DocuSignDocument.DocumentId	"1"	When multiple DocuSign documents are included in the same DocuSign envelope, every document needs a unique Id. In this example, there is only one document.
DocuSignDocument.Name	"StudentInfoPdf.pdf"	Name of the signed document.
DocuSignDocument.Content	Pdf	Name of the variable that holds the document image (refer to Create a PDF of the Form). In our example the name of the variable is **Pdf**.
DocuSignRecipient.FirstName	studentEntity.FirstName	First name of the user who submitted the signed document.
DocuSignRecipient.LastName	studentEntity.LastName	Last name of the user who submitted the signed document.

"To" Field	Value	Notes
DocuSignRecipient.Email	"tester@anthology.com"	Email address of the person who will receive the signed DocuSign document.
DocuSignRecipient.ConcealSSN	True	By default, the value is set to **True**, which masks the Ssn field value typed by the recipient. Set the value to **False** to display the typed value.
DocuSignRecipient.SignerId	"1"	The SignerId should match the Signer property in the DocuSign component on the form. Allowed values are "1" to "5".

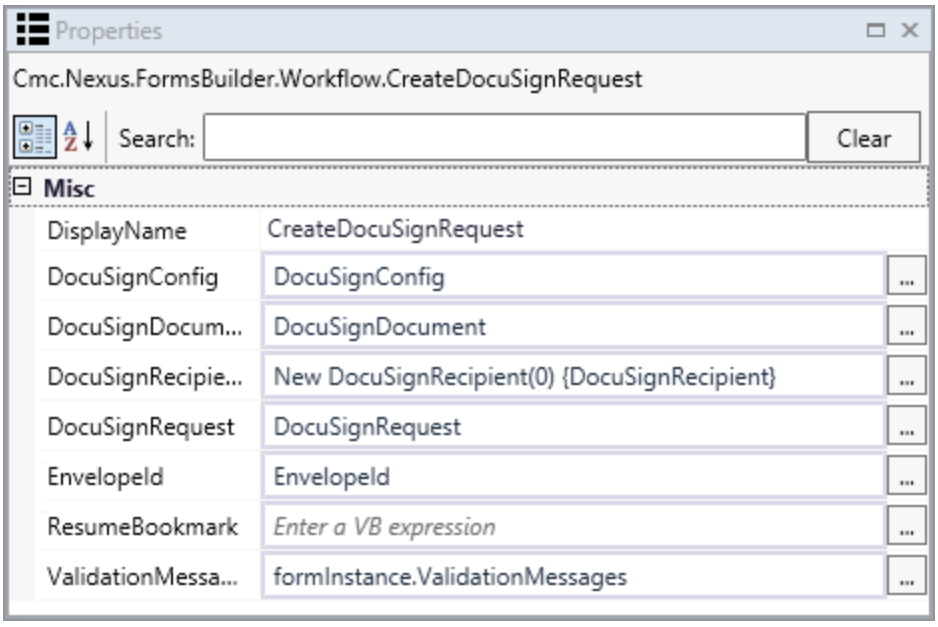
5. **Save** the workflow locally and continue with the next set of steps.

Create the DocuSign Request and Specify the IFrame URL

In the following steps we will continue to add activities to the *Default-Frame* State, which represents the IFrame form.

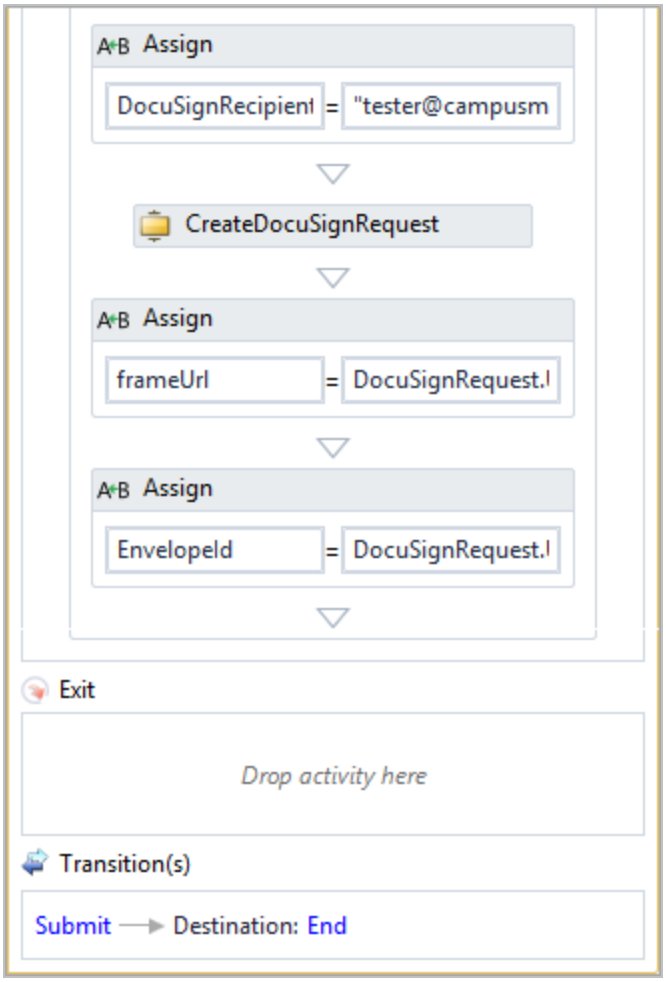
1. Drop the **CreateDocuSignRequest** activity below the previous activity.

   Specify the properties for the CreateDocuSignRequest activity using the names of the variables created above as follows:



   Warning: Do not enter anything in ResumeBookmark if you are creating Single Signer workflows. This could cause InstanceHandleConflictExceptions and aborted workflows. Fill this in only for Multi-Signer workflows.

   The out argument DocuSignRequest returns the envelope Id and URL of the signed DocuSign document.



2. Drop an **Assign** activity below the CreateDocuSignRequest activity.
    a. In the *To* field, specify **frameUrl**. (This is the argument associated with the IFrame form.)

    Be sure to use the exact casing shown here.

    b. In the *Value* field, specify **DocuSignRequest.Url**.
3. Drop another **Assign** activity into the workflow.
    a. In the *To* field, specify **EnvelopId**.
    b. In the *Value* field, specify **DocuSignRequest.EnvelopeId**.

Notes

- This assignment allows for the reuse of the EnvelopeId. If EnvelopeId has a value, it will be reused; if not a new one is generated (and can be assigned to a variable for reuse).

- This assignment is needed only for the primary/single signer (logged in Student) use cases. Signatures for multiple signers will be done outside of the workflow within DocuSign.

4. **Save** the workflow locally and continue with the next set of steps.

Receive the Signed DocuSign Document

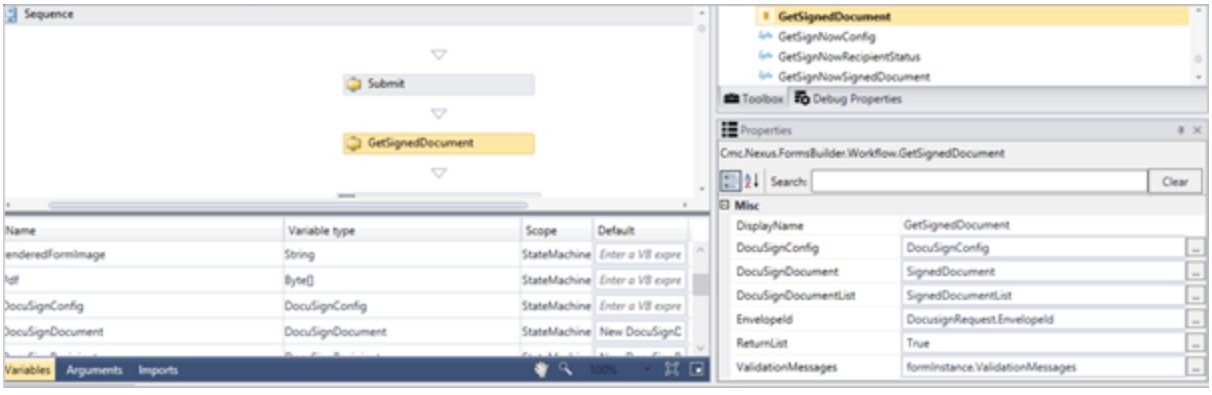
In the following steps we will create activities in the Submit transition which follows the IFrame form.

1. Double-click the **T1** transition after the *Default-Frame* state and rename it as **Submit**.
2. Drop a **WaitForFormBookmark** activity into the Trigger area of the Submit transition.
3. Rename the WaitForFormBookmark activity as **Submit**.
4. Create a variable named **SignedDocument**. In the Variable type field, select **DocuSignDocument**. Set the Scope to **StateMachine**.
5. Create another variable named **SignedDocumentList**. In the Variable type field, select **List Collection (system.collection.Generic.List<T>)** and inside the List, select the **DocuSignDocument** Entity **(Cmc.Nexus.FormsBuilder.Entities.DocusignDocument)**. Set the Scope to **StateMachine**.
6. Drop a **GetSignedDocument** activity below the Submit activity.

   Specify the properties for the activity as follows:

   - DocusignConfig = **DocusignConfig** (This is the variable created for the DocusignConfig Entity)
   - DocuSignDocument = **SignedDocument** (This is the name of the variable created above.)
   - DocusignDocumentList = SignedDocumentList (This is the name of the variable created above.)
   - EnvelopeId = **DocuSignRequest.EnvelopeId**
   - ReturnList = True
   - Validation Messages = **formInstance.ValidationMessages**

   Workflow Composer automatically wraps the activities in a Sequence.



7. **Save** the workflow locally and continue with the next set of steps.

If it is necessary to troubleshoot the receipt of the signed document, you might want to consider the procedure of Write the PDF to Disk.

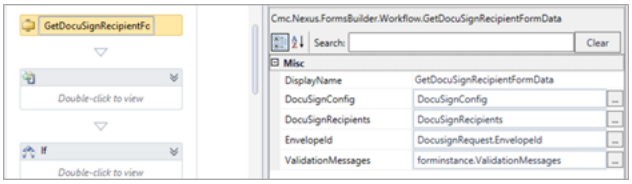
Receive Data Submitted by the Signer

In the following steps we will create activities in the Submit transition which follows the IFrame form to fetch the data submitted by the signer and add the data in the renderer log.

1. Create a variable named **TagName**. In the Variable type field, select **String**. Set the Scope to **StateMachine**.
2. Create a variable named **Value**. In the Variable type field, select **String**. Set the Scope to **StateMachine**.
3. Create a variable named **DocuSignRecipients**. In the Variable type field, select **Array of <T>** and from the T dropdown, select **Cmc.Nexus.FormsBuilder.Entities.DocuSignRecipient**. Set the Scope to **StateMachine**.
4. Create a variable named **DocuSignFormData**. In the Variable type field, select the **List Collection (system.collection.Generic.List<T>)** and inside the List select the **DocuSignFormDataitem** Entity(**Cmc.Nexus.FormsBuilder.Entities.DocuSignFormDataitem.**). Set the Scope to **StateMachine**.
5. Drop a **GetDocuSignRecipientFormData** activity below **GetSignedDocument**activity.

   Specify properties for the activity as follows:

   - DocuSignConfig = **DocuSignConfig** (This is the variable created for DocusignConfig Entity)
   - DocuSignRecipients = **DocuSignRecipients** (This is the variable created above)
   - EnvelopeId = **DocusignRequest.EnvelopeId**
   - ValidationMessage = **forminstance.ValidationMessages**

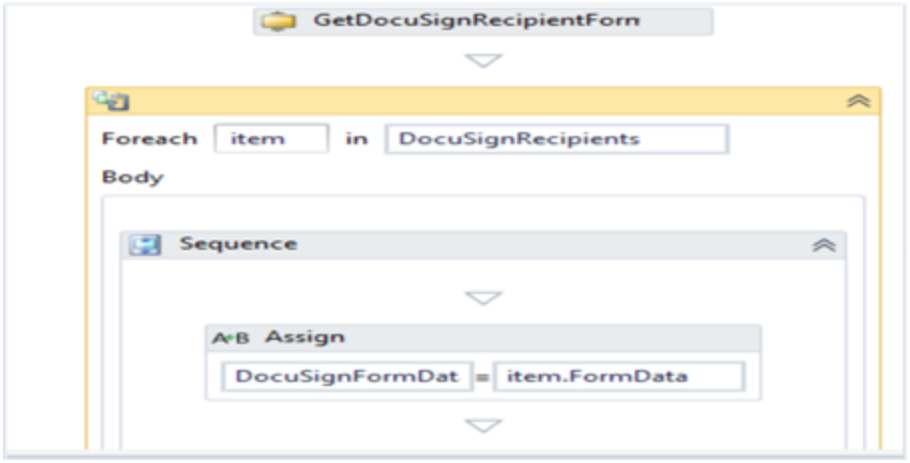


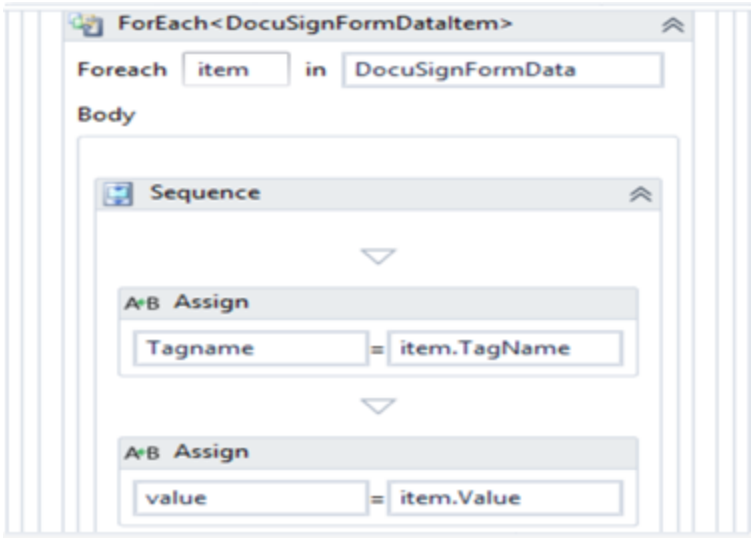
6. Drop a ForEach<> activity below the **GetDocuSignRecipientFormData** activity. Specify the **TypeArgument** as **Cmc.Nexus.FormsBuilder.Entities.DocuSignRecipient** And **Values** as '**DocuSignRecipients**' (This is the name of the variable created above).
7. Drop an **Assign** activity inside the **ForEach<>** activity, and specify the following values:
   - **To** = DocuSignFormData
   - **Value** = item.FormData
8. Drop a **ForEach<>** activity below the **Assign** activity.

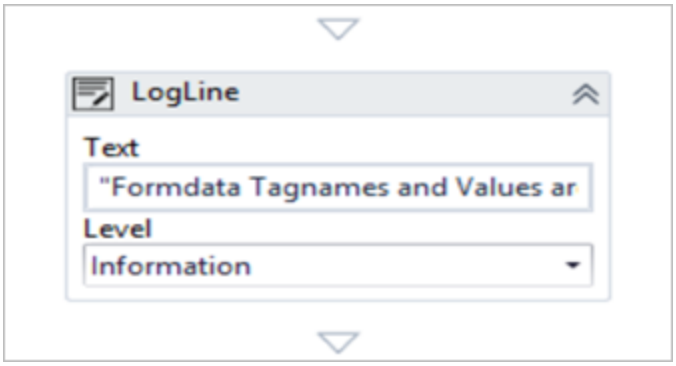
   Specify the **TypeArgument** as **Cmc.Nexus.FormsBuilder.Entities.DocuSignFormDataItem**.

   Specify the Value as in '**DocuSignFormData** (This is the name of the variable created above)

9. Drop an **Assign** activity inside the **ForEach<>** activity, and specify the following values
    - **To** = TagName
    - **Value** = item.TagName
10. Drop an **Assign** activity inside the **ForEach<>** activity, and specify the following values:
    - **To** = Value
    - **Value** = item.Value
11. Drop a **Logline** activity and specify the following values:
    - **Text** = "Formdata Tagnames and Values are :"& Tagname & Value
    - **Level** = Information







Create and Save the Document in Anthology Student

To convert a DocuSign document to a DocumentEntity that can be attached to a record in Anthology Student, add the CreateDocument and SaveDocument activities to the workflow. To attach multiple documents/attachments in Anthology Student, add the CreateDocument and SaveDocument activity in a ForEach<> loop. These activities will be placed in the Submit transition.

1. Create a variable named **Doc**. Set the Scope to **StateMachine**.

   In the Variable type field, select **Browse for Type**. In the "Browse and Select a .Net Type" window, scroll down to Cmc.Nexus.Crm.Contracts > Cmc.Nexus.Crm.Entities and select **DocumentEntity**.

2. Drop a **ForEach<>** activity below the **ForEach<>** activity dropped below the **GetDocuSignRecipientFormData** activity.

   Specify the Foreach <item> in '**SignedDocumentList**' (This is the name of the variable created above).

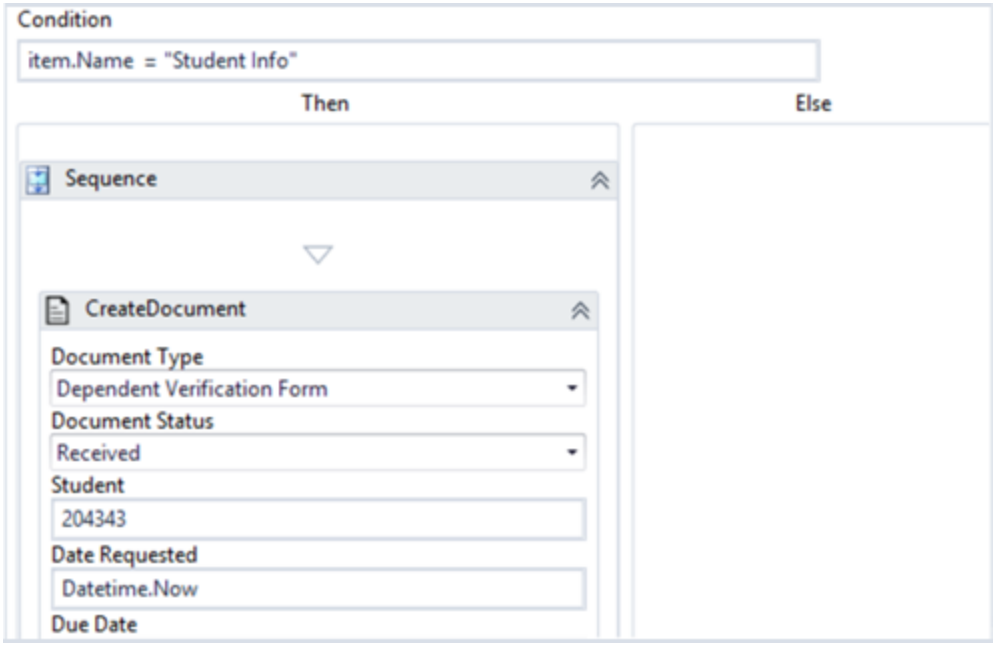
3. Drop an **If** condition activity inside the **Foreach<>** Activity. Specify the condition as **"item.Name = "Student Info""**.

   Note: Student Info is the name of the signed pdf document in the assign activity that was specified by the user in **Get the DocuSign Configuration and Pass the Recipient Information**. The If condition ensures that the signed pdf document is saved in the Document Center in Anthology Student and all attachments attached by the signer along with the signed summary report are also saved to the Document Center in Anthology Student.

4. Drop a **CreateDocument** activity and specify the following under the **Then** condition:

   Specify the required properties for the activity as follows:

   - Module = Select a Module from the drop-down list.
   - Document Type = Select a Type (Template) from the drop-down list.
   - Document Status = Select a Type from the drop-down list.
   - Student = Specify a Student Id or use a variable.
   - Due Date = Specify a date or use a variable, e.g., DateTime.Now
   - Document (OutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
   - Validation Messages = formInstance.ValidationMessages



5. Below the CreateDocument activity, drop an **Assign** activity for each row in the following table and type the indicated values:

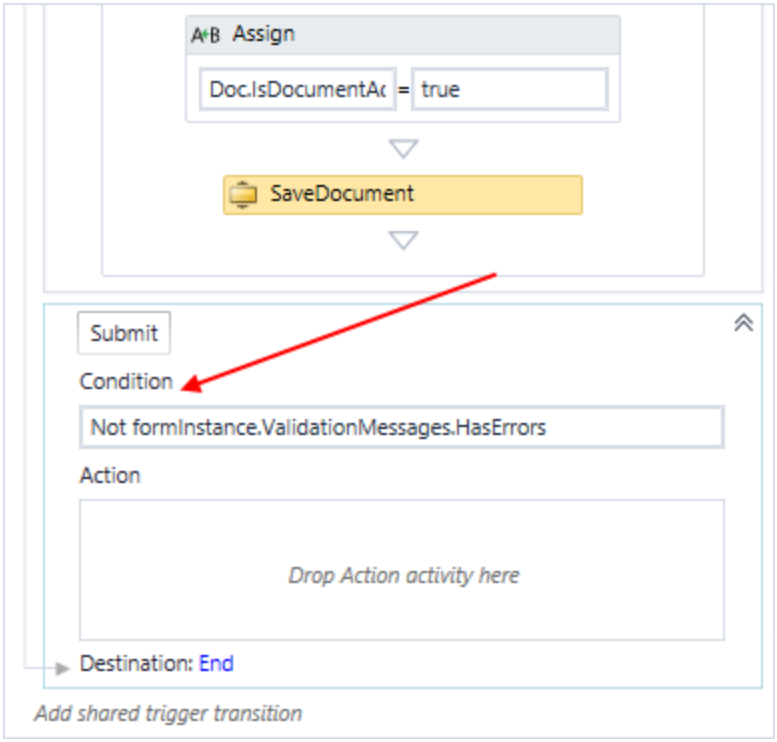
"To" Field	Value
Doc.DocumentImage	SignedDocument.Content
Doc.OriginalFileName	"Signedpdf.Pdf"
Doc.ImageType	"Pdf"
Doc.IsDocumentAddedManually	true

6. Drop a **SaveDocument** activity below the CreateDocument activity.

   Specify the properties for the SaveDocument activity as follows:

   - Document (InOutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
   - Validation Messages = **formInstance.ValidationMessages**

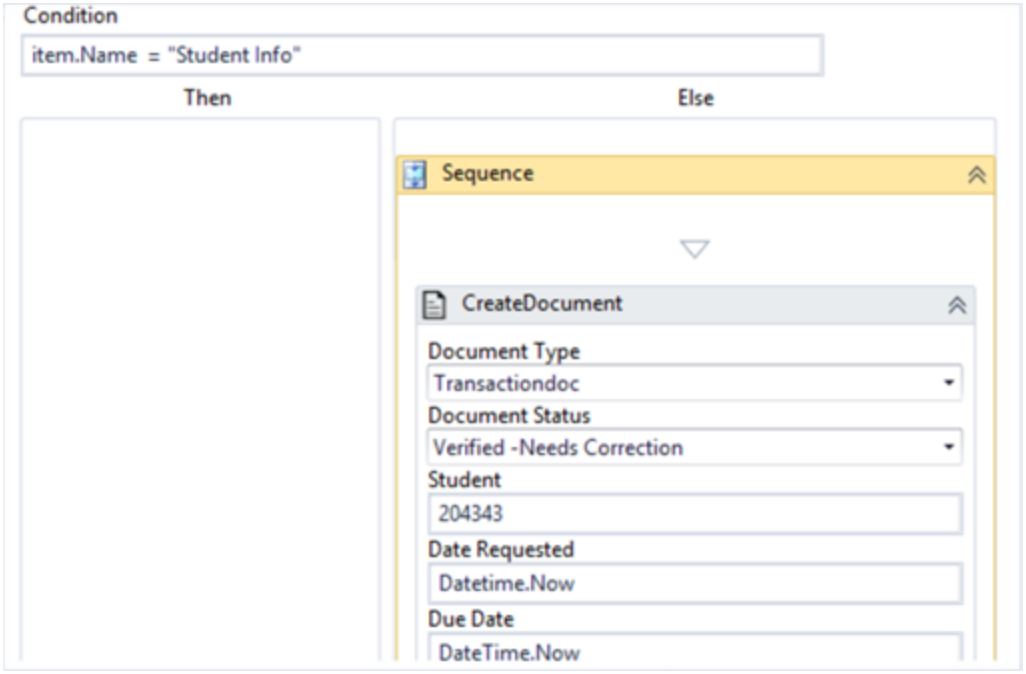
7. In the Condition field of the Submit transition, specify the following (to prevent users from submitting a form that has validation errors): **Not formInstance.ValidationMessages.Has Errors**



8. **Save** the workflow locally and continue with the next set of steps.
9. Drop a **CreateDocument** activity and specify the following under the **Else** condition:

   Specify the required properties for the activity as follows:

   - Module = Select a Module from the drop-down list.
   - Document Type = Select a Type (Template) from the drop-down list.
   - Document Status = Select a Type from the drop-down list.
   - Student = Specify a Student Id or use a variable.
   - Due Date = Specify a date or use a variable, e.g., DateTime.Now
   - Document (OutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
   - Validation Messages = formInstance.ValidationMessages



10. Below the CreateDocument activity, drop an **Assign** activity for each row in the following table and type the indicated values:

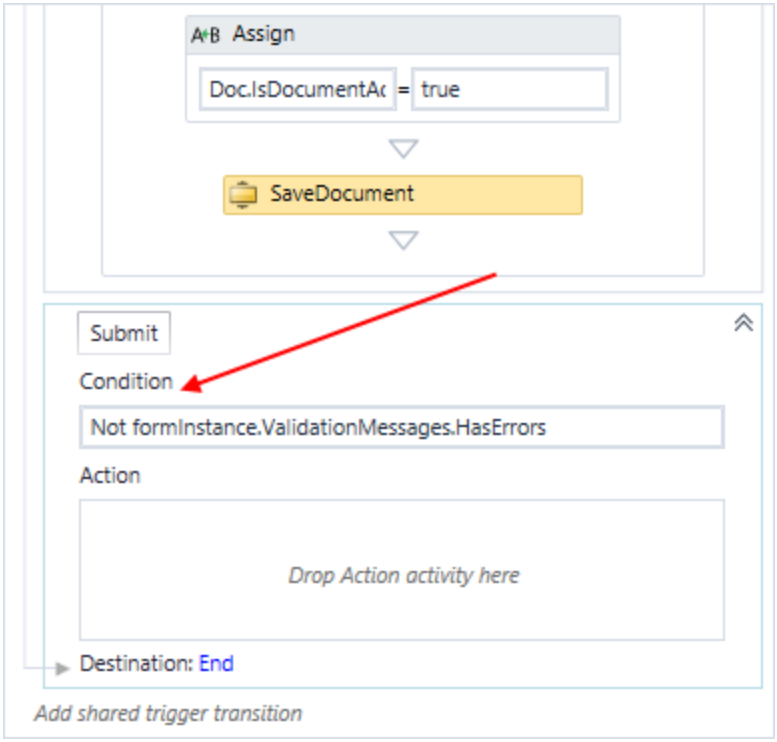
"To" Field	Value
Doc.DocumentImage	SignedDocument.Content
Doc.OriginalFileName	"Signedpdf.Pdf"
Doc.ImageType	"Pdf"
Doc.IsDocumentAddedManually	true

11. Drop a **SaveDocument** activity below the CreateDocument activity.

    Specify the properties for the SaveDocument activity as follows:

    - Document (InOutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
    - Validation Messages = **formInstance.ValidationMessages**

12. In the Condition field of the Submit transition, specify the following (to prevent users from submitting a form that has validation errors): **Not formInstance.ValidationMessages.Has Errors**



13. **Save** the workflow locally and continue with the next set of steps.

Final Steps

1. Select **Publish**. The New Workflow Definition Version window is displayed.
2. If you want the workflow to be run as soon as the event occurs on the entity, select **Enable This Workflow Version?**, otherwise leave the check box cleared.
3. Select **Save**, then **Cancel** to close the publisher window.
4. In the **Sequence List**, select and complete the form that contains the DocuSign fields.
5. Verify that the signed DocuSign document is available at the return URL.
6. Verify that the signed document is added to the Anthology Student database.

While testing your workflow, make sure that you terminate the previous instance of the workflow before running an updated version of the same workflow. In Workflow Composer, select **Open Persisted Workflow**, select your workflow instance, and select **Terminate**.

Persisted workflow instances can be deleted from the Sequence Designer workspace. For more information, refer to Delete Persisted Workflow Instances.

DocuSignWorkflow Sample - Multiple Signers

The following procedure details a sample workflow and forms for a sequence that contains multiple signers.

Prerequisites

1. The DocuSign properties are configured in Forms Builder. Refer to DocuSign Settings.
2. A form sequence is created. In our example, the sequence includes the following forms:
    a. *Welcome*
    b. *CMC_Student_Personal Info* — This is an admissions application form that collects the personal data of a student and contains four DocuSign components (signature and date fields for two signers).

    The signing process for the primary signer is the same as described in DocuSignWorkflow Sample - Single Signer. The signing process for the primary signer is an embedded process within the form window. For the secondary signer, DocuSign sends an email along with the document to be signed to the secondary signer. The signing process for primary and secondary signer takes place within the same DocuSign request and the same envelop. DocuSign returns events when each signer signs the document and when the document is complete. The workflow responds to these events.
    c. *CoSigner* — This form informs the student that a co-signer is needed and provides two text boxes for the co-signer's name and email address.
    d. *Default-Frame* — This form contains an IFrame component with the following properties (case sensitive):
    Name = **docuSignFrame**
    Url = **{{vm.models.frameUrl}}**

    > ⚠ We recommend that you copy the original *Default-Frame* form, edit the copy, and use it in your sequences. Save a backup copy of your form.

    e. *Default-DocuSignWait* — This form is provided with the Forms Builder installation. It contains two HTML components.

    The first HTML component displays the text "*Current user may close this browser tab. This page will auto submit when all the signatures are collected from DocuSign.*"

    The second HTML component contains the following JavaScript code which disables the Next button until the co-signer's signature is received:

```
<script ng-if="!vm.models.signer2Complete" ng-cloak>
$(document).ready(function(){
var elem = $("[value='Automatically continues when all signatures collected']");
elem.css("cursor", "not-allowed");
elem.css("opacity", "0.65");
});
</script>
```

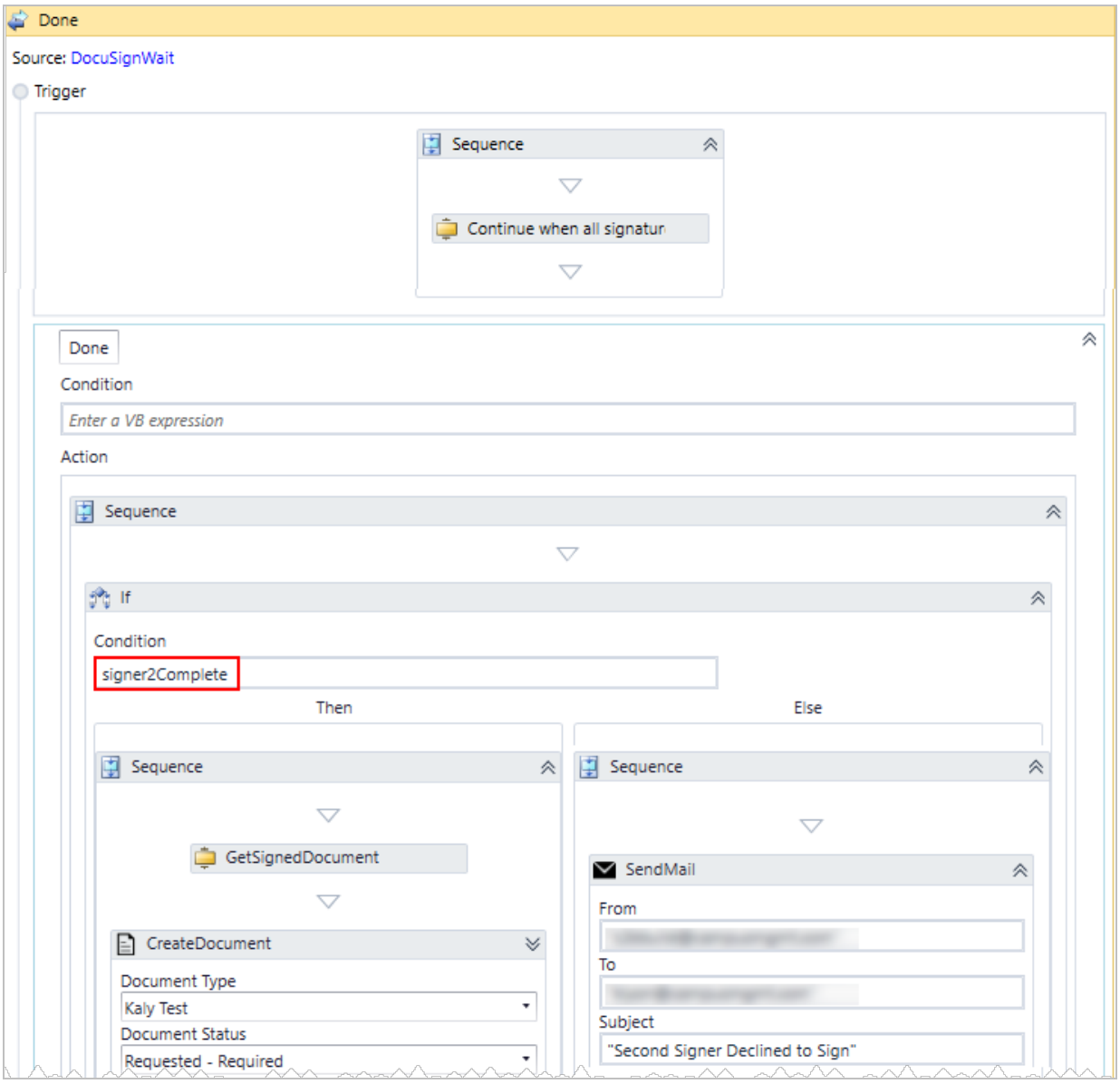


The *Default-DocuSignWait* form will be overwritten when Forms Builder is upgraded. Copy the original form, edit the copy, and use it in your sequences. Save a backup copy of your form.

Even though the Default-DocuSignWait form is provided for you, you must still add the Boolean argument **signer2Complete**that enforces the wait. The argument needs to be added in the transition after the DocuSignWait form.

In Forms Builder 3.7 and later, when any signer after the first/primary signer declines to sign the document, Webhook sets **signer2Complete** to false upon return to the form sequence. To handle this scenario in the workflow for the sequence, use the signer2Complete argument as a Condition in an If activity. By checking the signer2Complete value, the If Then/Else branches can then be designed as needed.

- The Then branch would be entered if signer2Complete = true, i.e., all signers signed without issue and the signed document is retrieved.
- The Else branch would be entered if signer2Complete = false, i.e., one of the signers declined. The subsequent activities could be to send an email notification to the first signer and/or update the document status in Anthology Student.



The argument signer2Complete needs to be defined in the Arguments tab. Be sure to match the casing in the JavaScript in the HTML component of the Default-DocuSignWait form. The argument is also used in the CreateDocuSignRequest below.

Name	Direction	Argument type	Default value
event	In/Out	ConstructedEvent	*Default value not supported*
studentEntity	In/Out	StudentEntity	*Default value not supported*
frameUrl	In/Out	String	*Default value not supported*
signer2Complete	In/Out	Boolean	*Default value not supported*
signer2Name	In/Out	String	*Default value not supported*

Variables | Arguments | Imports

This logic can be repeated as needed for additional signers.

To optimize the display of your e-signature forms on mobile devices, you may want to add the CSS described in this Note.

Values in the Type Property

The DocuSign component provides additional values on the Type property: Approve, Attachment, Checkbox, Company, Date, Date Signed, Decline, Email, Email Address, Envelope Id, Full Name, Initial, Number, Signature, Ssn, and Text.

The DocuSign component provides an automatic transition (auto-redirect) from the Default-Frame form to the confirmation form after a successful DocuSign session. The auto-redirect obsoletes the "DocuSign Confirmation Message Text" setting.

The auto-redirect depends on a forward direction in the WaitForFormBookmark activity in the transition after the DocuSign redirect state (typically Default-Frame), in particular if DisplayName has been modified.

- If there is only a single button and DisplayName has been customized but Transition Type was left as "Default", the auto-redirect moves forward to next form state.
- If there are two buttons and DisplayName(s) have been customized but Transition Type was left as "Default", the auto-redirect will assume the rightmost button (alphabetically last) is the transition for next state.

As a best practice, specify the Display Order and Transition Type (“MoveForward” or “MoveBack”) when button Display Name(s) have been customized as the behavior will be known. The Transition Type of “Default” was kept for compatibility for forms built prior to Transition Type being available on WaitForFormBookmark with default Display Names "Next" and "Back”.

In sequences for multiple signers, you must set **Transition Type = MoveForward** on the WaitForFormBookmark activity in the transition from the Default-Frame form to the DocuSignWait form. Refer to Transition from the IFrame Form to the DocuSignWait Form.

Enhancements in Forms Builder 3.6

The DocuSign component provides additional values on the Type property: Approve, Attachment, Checkbox, Company, Date, Decline, Email, Email Address, Envelope Id, Number, Ssn, and Text.

The DocuSign component provides an automatic transition (auto-redirect) from the Default-Frame form to the confirmation form after a successful DocuSign session. The auto-redirect obsoletes the "DocuSign Confirmation Message Text" setting.

The auto-redirect depends on a forward direction in the WaitForFormBookmark activity in the transition after the DocuSign redirect state (typically Default-Frame), in particular if DisplayName has been modified.

- If there is only a single button and DisplayName has been customized but Transition Type was left as "Default", the auto-redirect moves forward to next form state.
- If there are two buttons and DisplayName(s) have been customized but Transition Type was left as

"Default", the auto-redirect will assume the rightmost button (alphabetically last) is the transition for next state.

Best Practice is always to specify Display Order and Transition Type (“MoveForward” or “MoveBack”) when button Display Name(s) have been customized so behavior is known. The Transition Type of “Default” was kept for compatibility for forms built prior to Transition Type being available on WaitForFormBookmark with default Display Names "Next" and "Back”.

In sequences for multiple signers, you must set **Transition Type = MoveForward** on the WaitForFormBookmark activity in the transition from the Default-Frame form to the DocuSignWait form. Refer to Transition from the IFrame Form to the DocuSignWait Form.

Test the Multiple Signer Feature

To test the multiple signature feature with DocuSign, the test environment needs to meet specific requirements. DocuSign will try to call APIs on Renderer when secondary signers complete their signing process via email. This feature is referred to as *Webhook* and *DocuSign Connect*.

- If you are testing with a DocuSign account in test mode:
  - Renderer must be hosted on port 80 or 443.
  - Port 80 or 443 must be open on the firewall depending on which port Renderer is using for DocuSign to communicate. You can request an exception from your IT team for port 80 or 443 for an IP address range. The IP address ranges used for demo accounts are listed at this link: https://trust.docusign.com/en-us/trust-certifications/ip-ranges/
- If you are testing with a live DocuSign account:
  - Renderer must be hosted on port 443 (https).
  - If hosting Renderer on port 443 is not possible, a DocuSign representative for your account will have to make an exception.

To confirm that multi-signature forms are completing the workflow process, please confirm that Webhook responses are received and displayed in the log. For more information, refer to Webhook Responses for Multiple Signer Sequences.

Set Up DocuSign Account Preferences

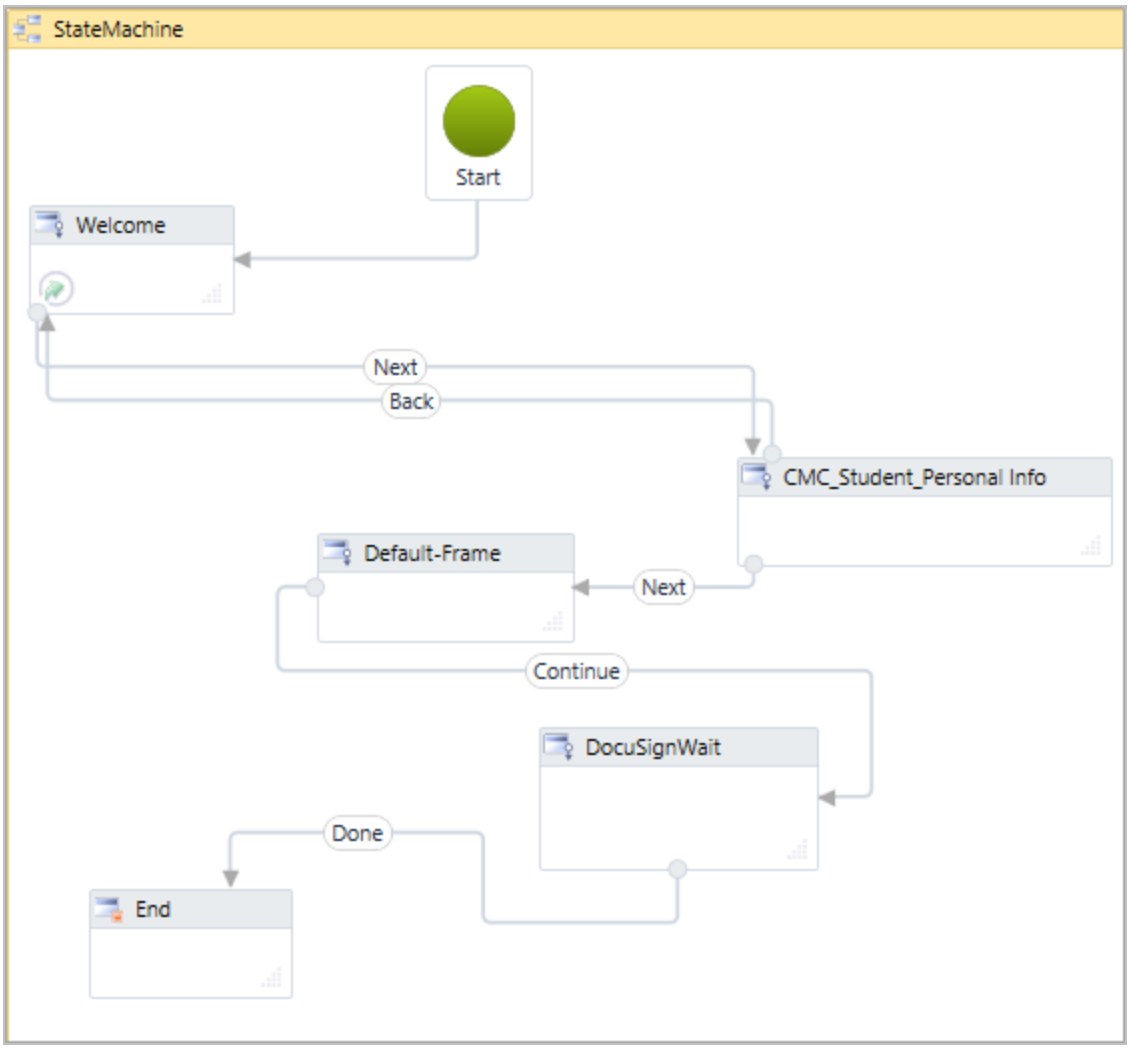
The Multiple Signer feature requires a specific setting in your DocuSign account. To enable the feature, log in to your DocuSign account and navigate to **Account Administration > Features**. Select the **DocuSign Connect** check box. This feature must be enabled for the *WebHook* functionality to work.

Another setting that is available in your Anthology Student account is found under **Manage Account > Email Notifications**. Here you configure who receives the DocuSign email when the institution is a signer or a sender.

Create the Workflow

Open the Workflow, Review Arguments, Variables, and Logging Requirements

1. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
2. In Workflow Composer, drag the State icons and Transition lines to so that you can easily locate each item in the StateMachine workflow.



The workflow requires the following arguments and variables. You can create these arguments and variables before working on the activities, or you can add them when they are needed for a specific activity (as described below).

> ⚠ Keep in mind that arguments are passed in JSON format and that JSON elements are case sensitive. **Be sure to match the casing of argument names in Workflow Composer and Form Designer.**

Arguments

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*
studentEntity	In/Out	StudentEntity	*Default value not supported*
frameUrl	In/Out	String	*Default value not supported*
signer2Complete	In/Out	Boolean	*Default value not supported*

Variables

Name	Variable type	Scope	Default
URL	String	CMC_Student_Personal Info	*Enter a VB expression*
renderedFormImage	String	StateMachine	*Enter a VB expression*
Pdf	Byte[]	StateMachine	*Enter a VB expression*
DocuSignConfig	DocuSignConfig	StateMachine	*Enter a VB expression*
DocuSignDocument	DocuSignDocument	StateMachine	New DocuSignDocument
DocuSignRecipient	DocuSignRecipient	StateMachine	New DocuSignRecipient
DocuSignRequest	DocuSignRequest	StateMachine	*Enter a VB expression*
SignedDocument	DocuSignDocument	StateMachine	*Enter a VB expression*
Doc	DocumentEntity	StateMachine	*Enter a VB expression*
DocuSignRecipient2	DocuSignRecipient	StateMachine	New DocuSignRecipient
studentId	Int32	StateMachine	*Enter a VB expression*

LogLine Activities

While testing and troubleshooting the workflow, we recommend adding LogLine activities at critical stages within the workflow. The following expression will provide logging for specific objects (replace <object> with the object name):

```
Newtonsoft.Json.JsonConvert.SerializeObject(<object>, Newtonsoft.Json.Formatting.Indented)
```

For example, to capture errors related to the CreateDocuSignRequest activity, you would place a LogLine activity with the following expression below the CreateDocuSignRequest activity.

```
Newtonsoft.Json.JsonConvert.SerializeObject(DocuSignRequest, Newtonsoft.Json.Formatting.Indented)
```

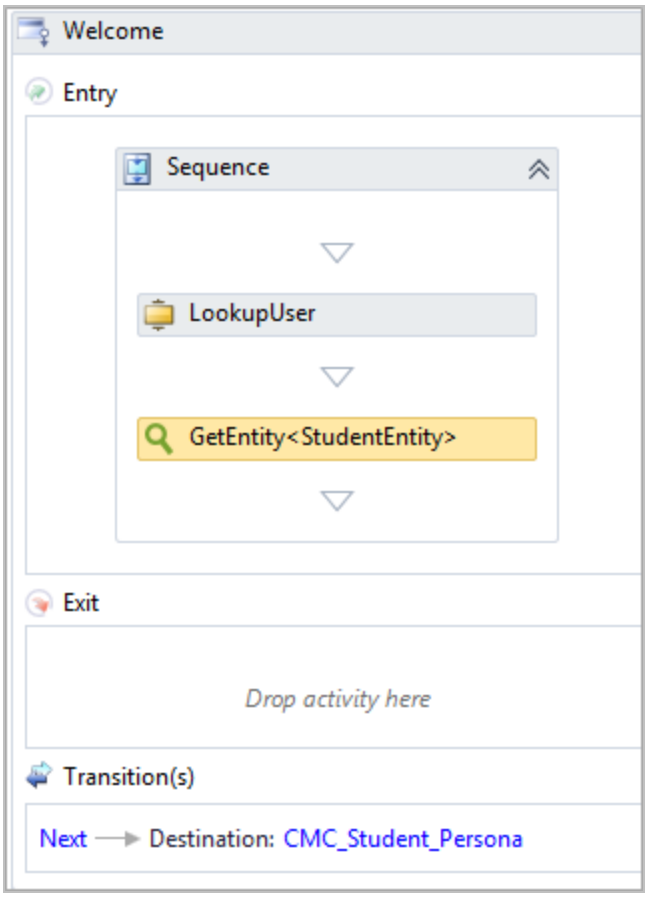
We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

LookupUser and GetEntity

If your sequence is non-authenticated (Anonymous=true), the LookupUser and GetEntity activities are not applicable, but a CreateEntity activity may be needed instead. Be sure that any anonymous sequence includes the user's email address. It is required as the DocuSign recipient address.

Use the *Welcome* form to retrieve the Student Id and Student Entity.

1. Double-click the icon of the **Welcome** state.
2. Create a variable named **studentid** of Type **Int32**.
3. Drop a **LookupUser** activity into the Entry area of the Welcome state.
    - In the **UserId** property, specify the **studentid** variable.
    - In the **UserName** property, specify **formInstance.UserName**.
4. Drop a **GetEntity** activity below the LookupUser activity.
    - In the **EntityId** property, specify the **studentid** variable.
    - In the **Result** property, specify **studentEntity**.



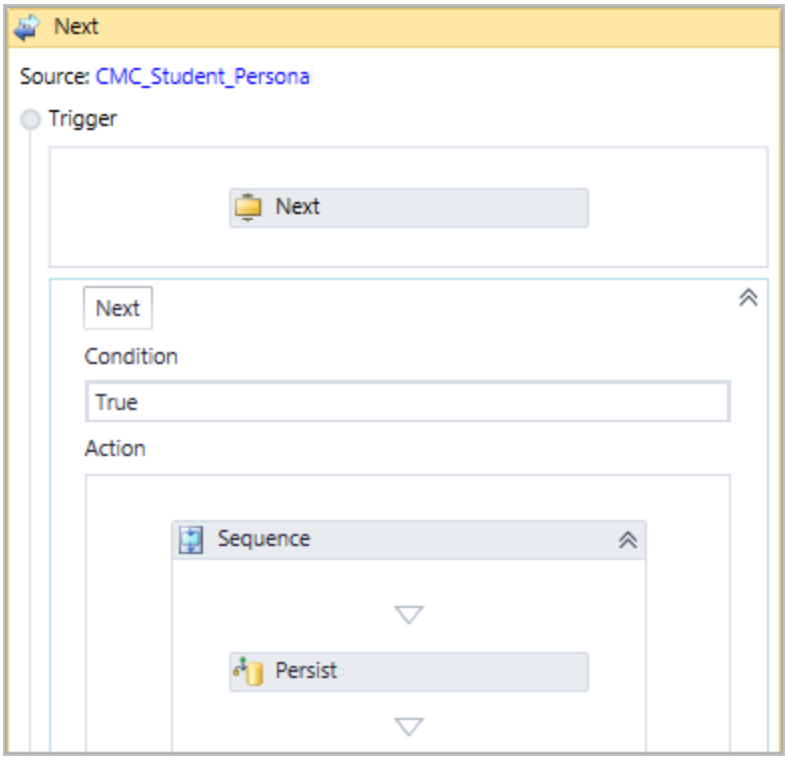
5. **Save** the workflow locally and continue with the next set of steps.

Persist the Workflow Instance After Collecting the Form Data

Since the workflow can be busy for a long time, we recommend that you persist the workflow instance before creating the PDF.

1. Double-click the **Next** transition from the form that is used to collect the data, *CMC_Student_Personal Info*.
2. Set the Condition to **True**.
3. Drag a **Persist** activity into the Trigger area below the Next activity. The Persist activity does not require any properties to be specified.

   Workflow Composer automatically wraps the Next and Persist activities in a Sequence.



4. **Save** the workflow locally and continue with the next set of steps.

Create a PDF of the DocuSign Form

In the following steps we add activities to the Next transition to create a PDF of the form.

1. Create a variable named **URL**. Set the Variable type to **String**. Set the Scope to **StateMachine**. This variable will be assigned to the PDF that will be created from the form.

   > As of Forms Builder 3.5, the URL input argument for the PrintUrlToPdf activity is optional. When the URL is not specified, the activity constructs the URL for all forms traversed in the sequence.

2. Drop an **Assign** activity into the Entry area of the State.

a. In the *To* field, specify the variable name **URL**.

b. In the *Value* field, specify the following:

formInstance.RendererBaseUrl + "#/viewCreator/" + formInstance.WorkflowDefinitionId.ToString + "/forms=CMC_Student_Personal+Info"

Where "/forms=*CMC_Student_Personal+Info*" indicates the Forms Builder form that contains the DocuSign component.

Notes:

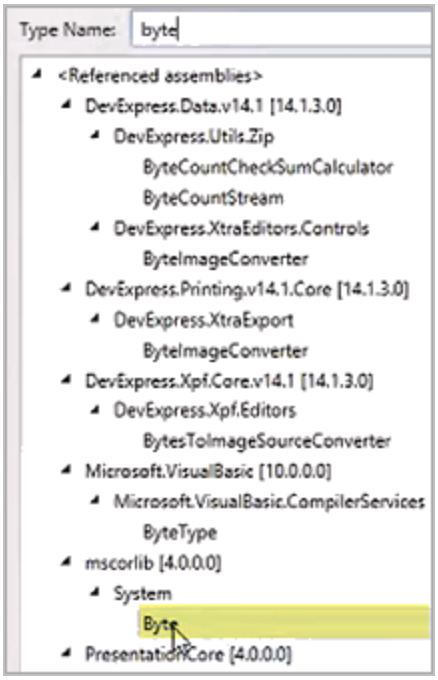
- If the form name has a space, replace the space with a + sign as shown in the example: *CMC_Student_Personal+Info*
- If multiple forms are sent to DocuSign, specify a comma-separated list of form names in the "/forms=" attribute.

3. Create a variable with a name like **Pdf**. Set the Scope to **StateMachine**. This variable will hold the document image created from the form.

Make sure you use the same variable name when you reference this variable later in the workflow. (This applies to any other variable.)

a. In the Variable type field, select **Array of [T]** and select **Browse for Type**.

b. In the "Browse and Select a .Net Type" window, specify **byte**, select the System variable **Byte**, and select **OK**.



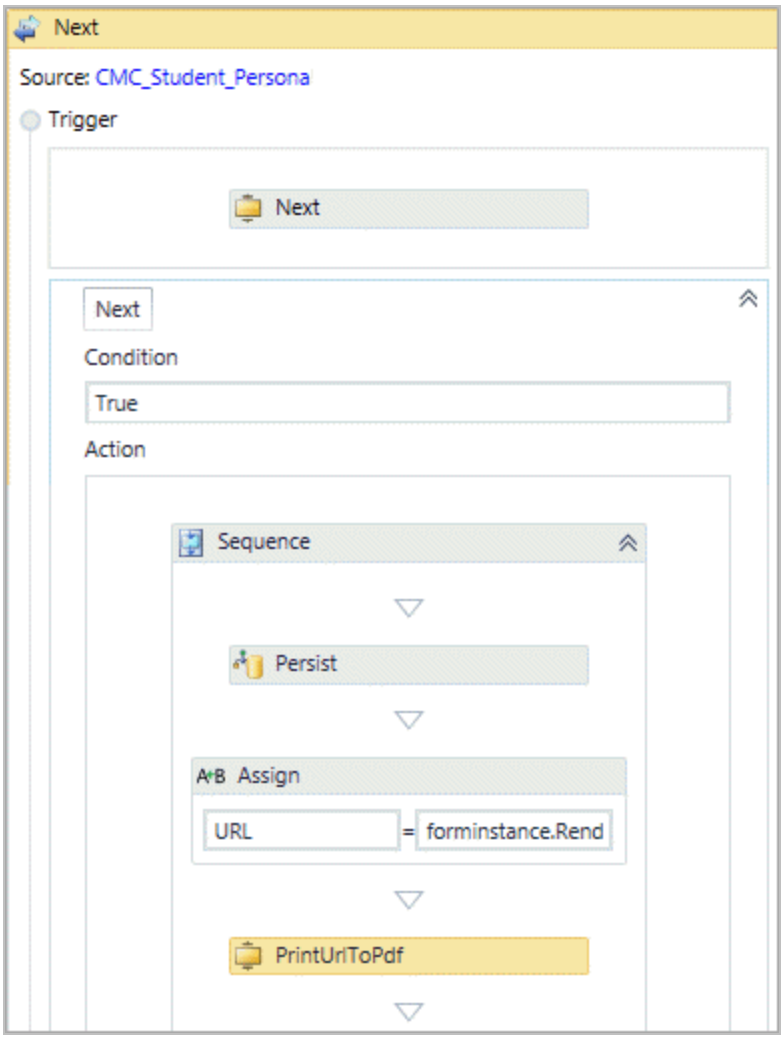
4. Drop a **PrintUrlToPdf** activity below the Persist activity.

Specify the properties for the activity as follows:

- PdfDocument = **Pdf** (This is the name of the variable created above.)
- Url = **URL** (This is the name of the variable created above.)

  As of Forms Builder 3.5, the URL input argument for the PrintUrlToPdf activity is optional. When the URL is not specified, the activity constructs the URL for all forms traversed in the sequence.

- Validation Messages = **formInstance.ValidationMessages**



5. **Save** the workflow locally and continue with the next set of steps.

Get the DocuSign Configuration and Pass the Recipient Information

In the following steps we will continue to add activities to the Next transition from the *CMC_Student_Personal Info* form.

1. Create a variable named **DocuSignConfig**. Set the Scope to **StateMachine**.

   In the Variable type field, select **Browse for Type**. In the "Browse and Select a .Net Type" window, scroll down to Cmc.Nexus.FormsBuilder.Contracts > Cmc.Nexus.FormsBuilder.Entities and select **DocuSignConfig**.

2. Drop a **GetDocuSignConfig** activity below the PrintUrlToPdf activity.

   Specify the properties for the activity as follows:

   - DocuSignConfig = **DocuSignConfig**
   - Validation Messages = **formInstance.ValidationMessages**

   The GetDocuSignConfig activity retrieves the User Name, Password, Integrators Key, and REST API Url from the DocuSign settings in Forms Builder. These values enable the workflow to log in to DocuSign.

   To process the DocuSign request, the Email Subject and Return URL properties need to be assigned to the DocuSignConfig variable. These properties are required.

   > ⚠ The only two properties for the DocuSignConfig object that should ever be modified in the workflow definition are **EmailSubject** and **ReturnUrl**. The values for all the other properties are retrieved from the DocuSign Settings saved in the database. Any modification done to the values for the other DocuSignConfig properties in the workflow definition will likely result in errors when the DocuSign portion of the sequence is executed.

3. Create the following variables:

Variable Name	Type	Scope	Default
DocuSignRecipient	Cmc.Nexus.FormsBuilder.Entities.DocuSignRecipient	StateMachine	New DocuSignRecipient
DocuSignRecipient2	Cmc.Nexus.FormsBuilder.Entities.DocuSignRecipient	StateMachine	New DocuSignRecipient
DocuSignDocument	Cmc.Nexus.FormsBuilder.Entities.DocuSignDocument	StateMachine	New DocuSignDocument
DocuSignRequest	Cmc.Nexus.FormsBuilder.Entities.DocuSignRequest	StateMachine	N/A

4. Create the following argument:

   - Name = **signer2Complete**
   - Direction = **In/Out**
   - Argument type = **Boolean**

5. Below the GetDocuSignConfig activity, drop an **Assign** activity for each row in the following table and type the indicated values:

"To" Field	Value	Notes
DocuSignConfig.EmailSubject	"CMC DocuSign Test at " + DateTime.Now.ToString()	This is the email that the end user will receive after the signing process is done.

"To" Field	Value	Notes
DocuSignConfig.ReturnUrl	formInstance.RendererBaseUrl +"#/docusigncomplete"	This is the URL that will display the document after the signing process is done.
DocuSignDocument.DocumentId	"1"	The SignerId should match the Signer property in the DocuSign component on the form. Allowed values are "1" to "5".
DocuSignDocument.Name	"Student Info Pdf"	Name of the signed document.
DocuSignDocument.Content	Pdf	Name of the variable that holds the document image (refer to Create a PDF of the Form). In our example the name of the variable is **Pdf**.
DocuSignRecipient.Name	"RecipientName First"	Name of the user who submitted the signed document. This could be the student name retrieved from the StudentEntity.
DocuSignRecipient.Email	"tester@anthology.com"	Email address of the person who will receive the signed DocuSign document. This could be the email address retrieved from the StudentEntity.
DocuSignRecipient.ConcealSSN	True	By default, the value is set to **True**, which masks the Ssn field value typed by the recipient. Set the value to **False** to display the typed value.
DocuSignRecipient2.Name	"RecipientName Second"	Name of the co-signer who submitted the signed document. This could be a parent name from the StudentRelationshipAddressEntity.
DocuSignRecipient2.Email	"tester2@anthology.com"	Email address of the co-signer who will receive the signed DocuSign document. This could be the email address retrieved from the StudentRelationshipAddressEntity.
DocuSignRecipient2.ConcealSSN	True	By default, the value is set to **True**, which masks the Ssn field value typed by the recipient. Set the value to **False** to display the typed value.
DocuSignRecipient2.SignerId	"2"	The SignerId should match the Signer property in the DocuSign component on the form. Allowed values are "1" to "5".

"To" Field	Value	Notes
DocuSignRecipient2.RoutingOrder	"2"	In Forms Builder 3.6 and later, you can set up a sequential routing order, where each recipient receives the email notification once the previous recipient has completed their action. You can also have a mix of sequential and parallel routing. To set a parallel order, such that some recipients receive the document at the same time, set the same value for the routing order. When you use a routing order, you can also route an envelope to the same person multiple times. For example, you can send a document to student to approve, then send it on to a parent to sign, and finally send a copy the student again. Prerequisite: You must have permission to create a routing order. Refer to Allow Sequential Signing.

"To" Field	Value	Notes
DocuSignRecipient2. VariableToSetOnComplete	"signer2Complete"	The value "signer2Complete" is the argument created in the previous step. This argument is used as follows: • On the DocuSignRecipient2 variable – When the secondary signer completes signing, DocuSign will send an event to the Forms Builder server, and the Forms Builder server will set the variable from false to true. • In the ResumeBookmark property of the CreateDocuSignRequest activity –Refer to Note on the CreateDocuSignRequest activity. The string "Continue when all signatures are collected" is the value of the custom field named "Bookmark" that is returned by DocuSign. • In the Done transition – Refer to Receive the Signed DocuSign Document. • In the JavaScript associated with the DocuSignWait form. **Note**: Do not use *VariableToSetOnComplete* in single signer sequences.

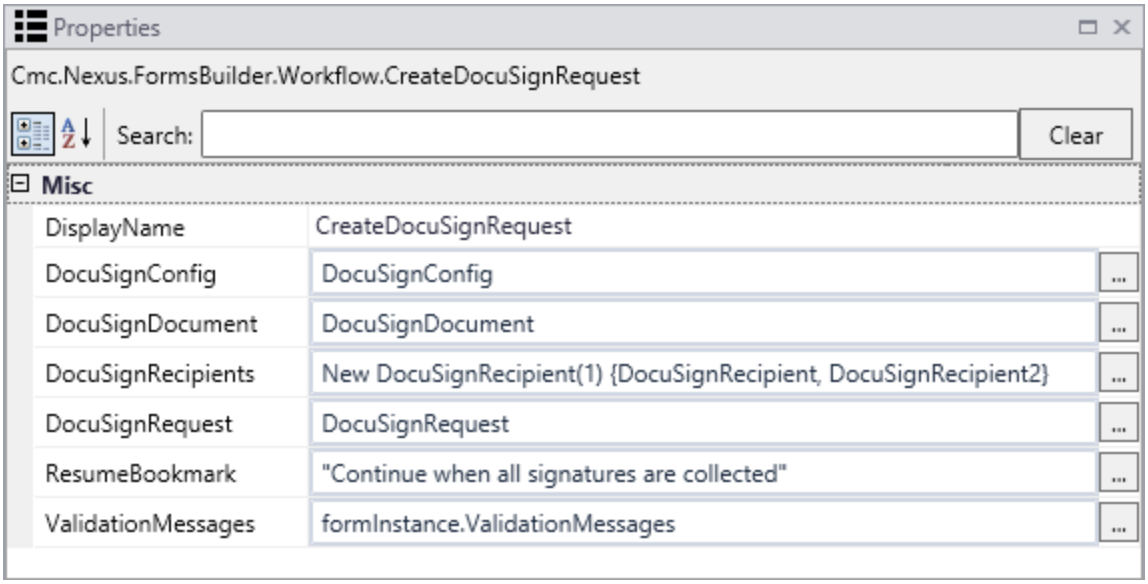
6. **Save** the workflow locally and continue with the next set of steps.

Create the DocuSign Request and Specify the IFrame URL

In the following steps we will continue to add activities to the Next transition from the *CMC_Student_Personal Info* form.

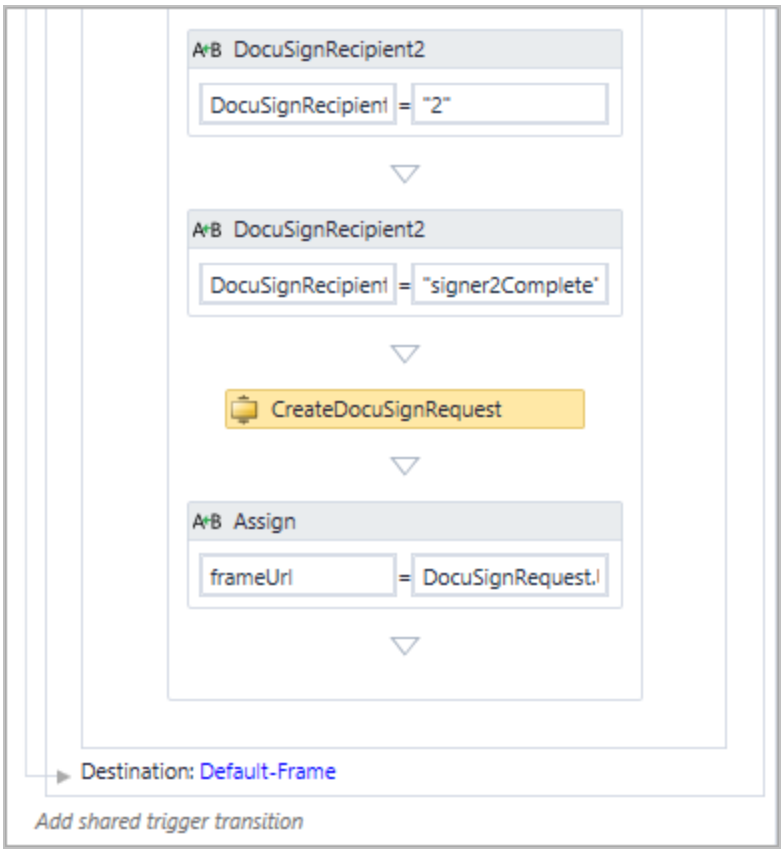
1. Drop the **CreateDocuSignRequest** activity below the previous activity.

   Specify the properties for the CreateDocuSignRequest activity using the names of the variables created above as follows:



Notes:

- The expression for DocuSignRecipients passes multiple items (the primary and secondary recipient) within an array.
- The ResumeBookmark value will be passed to DocuSign as a custom field. When the envelope has been completed, DocuSign will return this flag and any process that has to happen after the bookmark can resume automatically without any user intervention.
- The out argument DocuSignRequest returns the envelope Id and URL of the signed DocuSign document.



2. Drop an **Assign** activity below the CreateDocuSignRequest activity.
    a. In the *To* field, specify **frameUrl** (This is the argument associated with the IFrame form.)

    ⚠ Be sure to use the exact casing shown here.

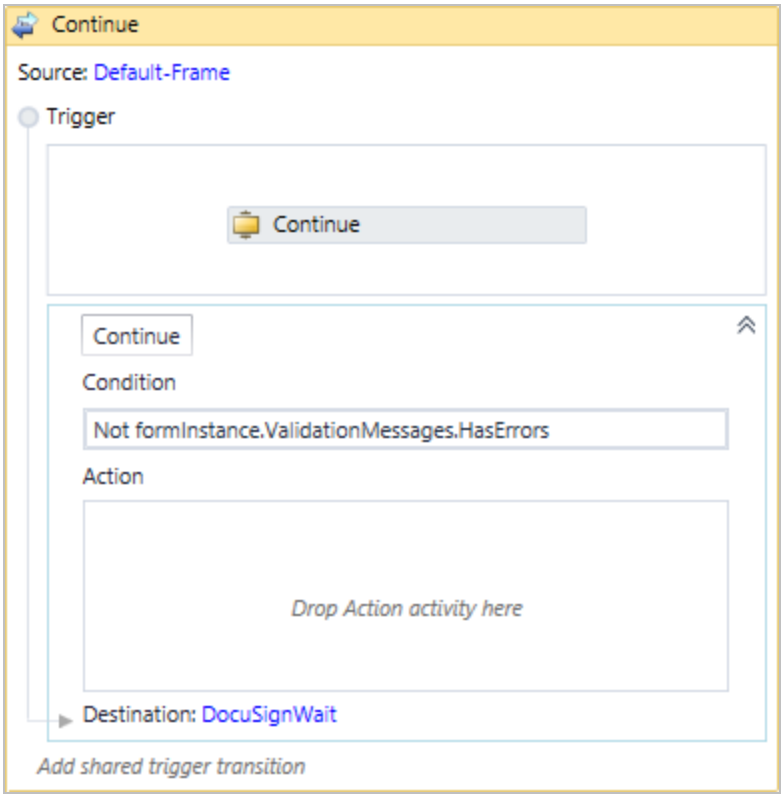
    b. In the *Value* field, specify **DocuSignRequest.Url**.
3. **Save** the workflow locally and continue with the next set of steps.

Transition from the IFrame Form to the DocuSignWait Form

The IFrame form (*Default-Frame*) will receive the DocuSign document with the first signature, but the signing process is not yet complete. We want the workflow to continue with the *DocuSignWait* form upon receipt of the first signature. The workflow will pause on the *DocuSignWait* form until the second signature is received. If user closes the form or logs out after submitting the first signature, upon return to the form, the workflow will display the *DocuSignWait* form.

1. Double-click the transition from the *Default-Frame* form to the *DocuSignWait* form.
2. In the Trigger area, name the WaitForFormBookmark as **Continue**.
3. Select the property **Transition Type = MoveForward** on the WaitForFormBookmark activity.

4. Specify the Condition as **Not formInstance.ValidationMessages.HasErrors**.



5. **Save** the workflow locally and continue with the next set of steps.

Receive the Signed DocuSign Document

In the following steps we will create activities in the Submit transition which follows the IFrame form.

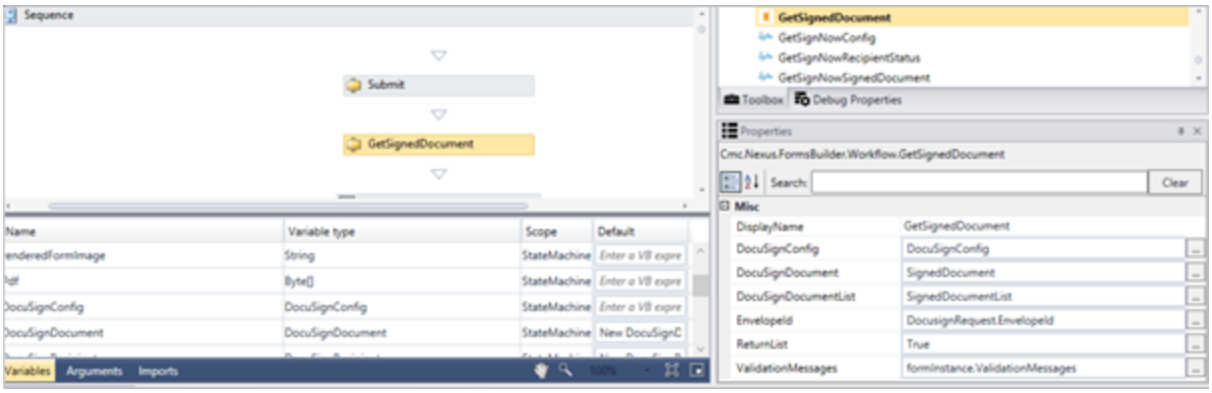
1. Double-click the **T1** transition after the *Default-Frame* state and rename it as **Submit**.
2. Drop a **WaitForFormBookmark** activity into the Trigger area of the Submit transition.
3. Rename the WaitForFormBookmark activity as **Submit**.
4. Create a variable named **SignedDocument**. In the Variable type field, select **DocuSignDocument**. Set the Scope to **StateMachine**.
5. Create another variable named **SignedDocumentList**. In the Variable type field, select **List Collection (system.collection.Generic.List<T>)** and inside the List, select the **DocuSignDocument** Entity **(Cmc.Nexus.FormsBuilder.Entities.DocusignDocument)**. Set the Scope to **StateMachine**.
6. Drop a **GetSignedDocument** activity below the Submit activity.

   Specify the properties for the activity as follows:

   - DocusignConfig = **DocusignConfig** (This is the variable created for the DocusignConfig Entity)
   - DocuSignDocument = **SignedDocument** (This is the name of the variable created above.)

- DocusignDocumentList = SignedDocumentList (This is the name of the variable created above.)
- EnvelopeId = **DocuSignRequest.EnvelopeId**
- ReturnList = True
- Validation Messages = **formInstance.ValidationMessages**

Workflow Composer automatically wraps the activities in a Sequence.



7. **Save** the workflow locally and continue with the next set of steps.

If it is necessary to troubleshoot the receipt of the signed document, you might want to consider the procedure of Write the PDF to Disk.

Receive Data Submitted by Signers

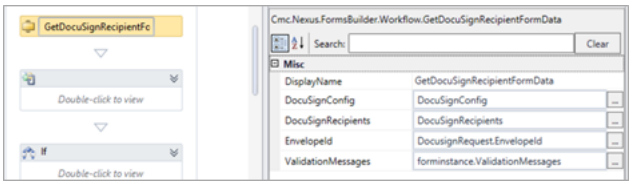
In the following steps we will create activities in the Submit transition which follows the IFrame form to fetch the data submitted by signers and add the data in the renderer log.

1. Create a variable named **TagName**. In the Variable type field, select **String**. Set the Scope to **StateMachine**.
2. Create a variable named **Value**. In the Variable type field, select **String**. Set the Scope to **StateMachine**.
3. Create a variable named **DocuSignRecipients**. In the Variable type field, select **Array of <T>** and from the T dropdown, select **Cmc.Nexus.FormsBuilder.Entities.DocuSignRecipient**. Set the Scope to **StateMachine**.
4. Create a variable named **DocuSignFormData**. In the Variable type field, select the **List Collection (system.collection.Generic.List<T>)** and inside the List select the **DocuSignFormDataitem** Entity(**Cmc.Nexus.FormsBuilder.Entities.DocuSignFormDataitem.**). Set the Scope to **StateMachine**.
5. Drop a **GetDocuSignRecipientFormData** activity below **GetSignedDocument** activity.

   Specify properties for the activity as follows:

   - DocuSignConfig = **DocuSignConfig** (This is the variable created for DocusignConfig Entity)
   - DocuSignRecipients = **DocuSignRecipients** (This is the variable created above)

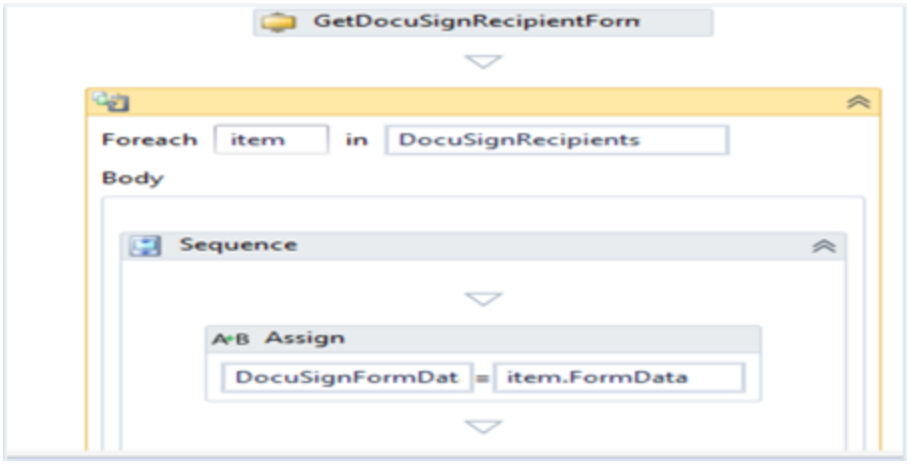
- EnvelopeId = **DocusignRequest.EnvelopeId**
- ValidationMessage = **forminstance.ValidationMessages**

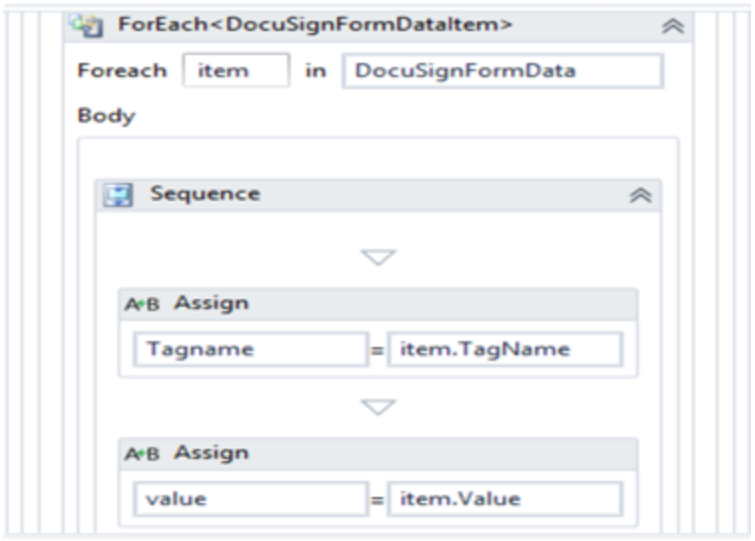


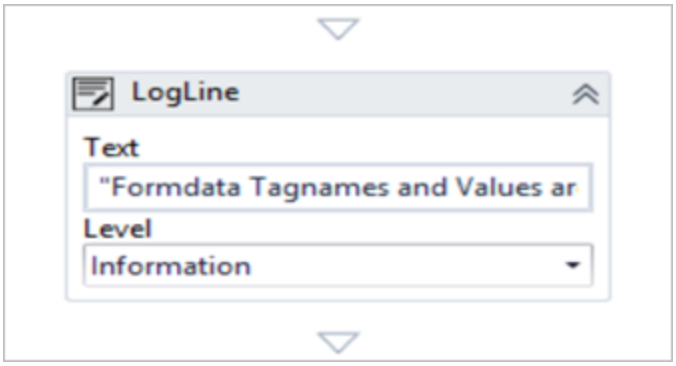
6. Drop a ForEach<> activity below the **GetDocuSignRecipientFormData** activity. Specify the **TypeArgument** as **Cmc.Nexus.FormsBuilder.Entities.DocuSignRecipient** And **Values** as '**DocuSignRecipients**' (This is the name of the variable created above).
7. Drop an **Assign** activity inside the **ForEach<>** activity, and specify the following values:
   - **To** = DocuSignFormData
   - **Value** = item.FormData
8. Drop a **ForEach<>** activity below the **Assign** activity.

   Specify the **TypeArgument** as **Cmc.Nexus.FormsBuilder.Entities.DocuSignFormDataItem**.

   Specify the Value as in '**DocuSignFormData** (This is the name of the variable created above)
9. Drop an **Assign** activity inside the **ForEach<>** activity, and specify the following values
   - **To** = TagName
   - **Value** = item.TagName
10. Drop an **Assign** activity inside the **ForEach<>** activity, and specify the following values:
    - **To** = Value
    - **Value** = item.Value
11. Drop a **Logline** activity and specify the following values:
    - **Text** = "Formdata Tagnames and Values are :"& Tagname & Value
    - **Level** = Information







Create and Save the Document in Anthology Student

To convert a DocuSign document to a DocumentEntity that can be attached to a record in Anthology Student, add the CreateDocument and SaveDocument activities to the workflow. To attach multiple documents/attachments in Anthology Student, add the CreateDocument and SaveDocument activity in a ForEach<> loop. These activities will be placed in the Submit transition.

1. Create a variable named **Doc**. Set the Scope to **StateMachine**.

   In the Variable type field, select **Browse for Type**. In the "Browse and Select a .Net Type" window, scroll down to Cmc.Nexus.Crm.Contracts > Cmc.Nexus.Crm.Entities and select **DocumentEntity**.

2. Drop a **ForEach<>** activity below the **ForEach<>** activity dropped below the **GetDocuSignRecipientFormData** activity.

   Specify the Foreach <item> in '**SignedDocumentList**' (This is the name of the variable created above).

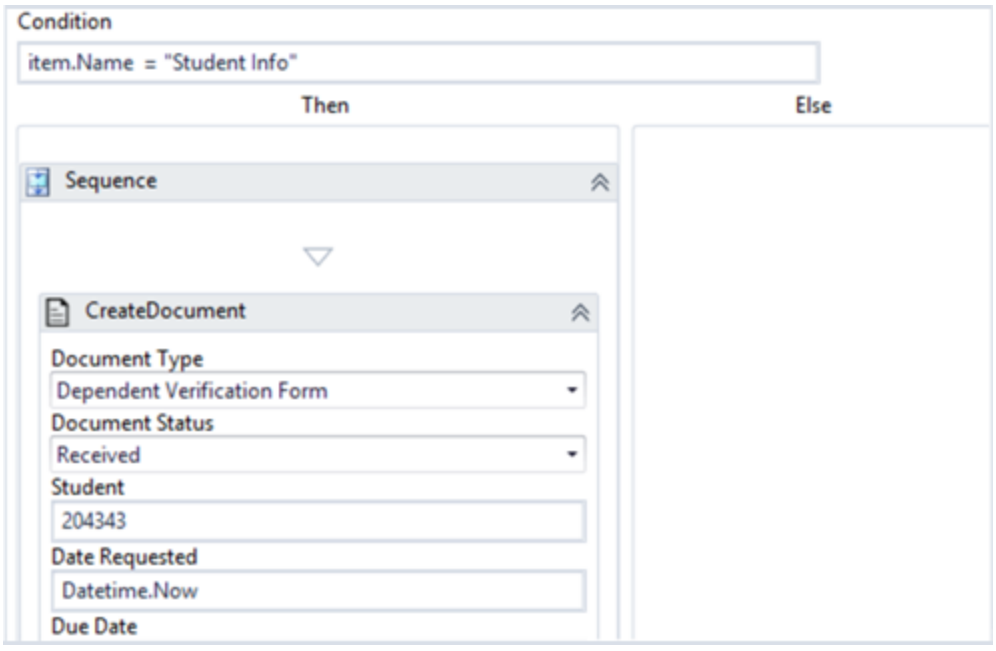
3. Drop an **If** condition activity inside the **Foreach<>** Activity. Specify the condition as **"item.Name = "Student Info""**.

   > Student Info is the name of the signed pdf document in the assign activity that was specified by the user in **Get the DocuSign Configuration and Pass the Recipient Information**. The If condition ensures that the signed pdf document is saved in the Document Center in Anthology Student and all attachments attached by the signer along with the signed summary report are also saved to the Document Center in Anthology Student.

4. Drop a **CreateDocument** activity and specify the following under the **Then** condition:

   Specify the required properties for the activity as follows:

   - Module = Select a Module from the drop-down list.
   - Document Type = Select a Type (Template) from the drop-down list.
   - Document Status = Select a Type from the drop-down list.
   - Student = Specify a Student Id or use a variable.
   - Due Date = Specify a date or use a variable, e.g., DateTime.Now
   - Document (OutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
   - Validation Messages = formInstance.ValidationMessages



5. Below the CreateDocument activity, drop an **Assign** activity for each row in the following table and type the indicated values:

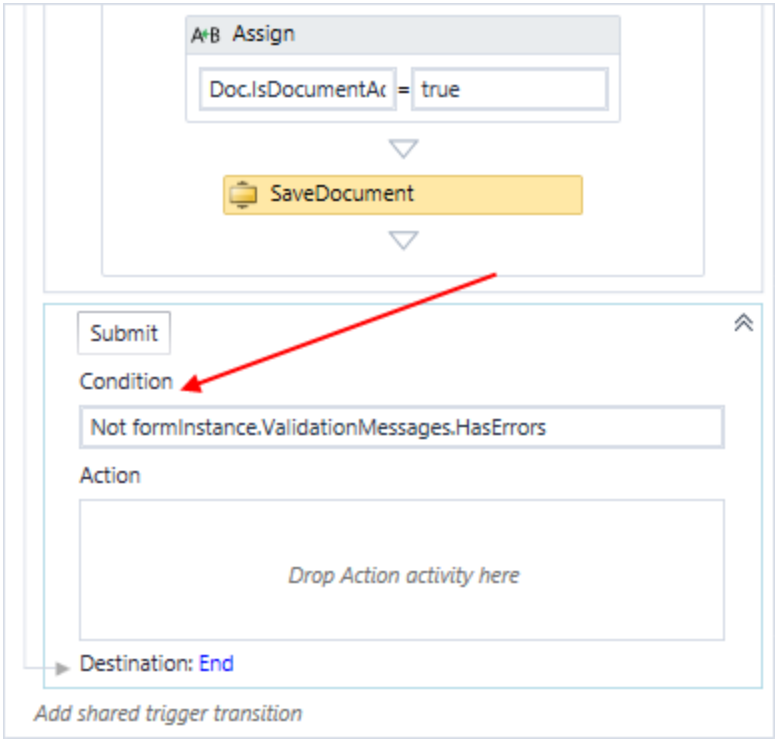
"To" Field	Value
Doc.DocumentImage	SignedDocument.Content
Doc.OriginalFileName	"Signedpdf.Pdf"
Doc.ImageType	"Pdf"
Doc.IsDocumentAddedManually	true

6. Drop a **SaveDocument** activity below the CreateDocument activity.

   Specify the properties for the SaveDocument activity as follows:

   - Document (InOutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
   - Validation Messages = **formInstance.ValidationMessages**

7. In the Condition field of the Submit transition, specify the following (to prevent users from submitting a form that has validation errors): **Not formInstance.ValidationMessages.Has Errors**



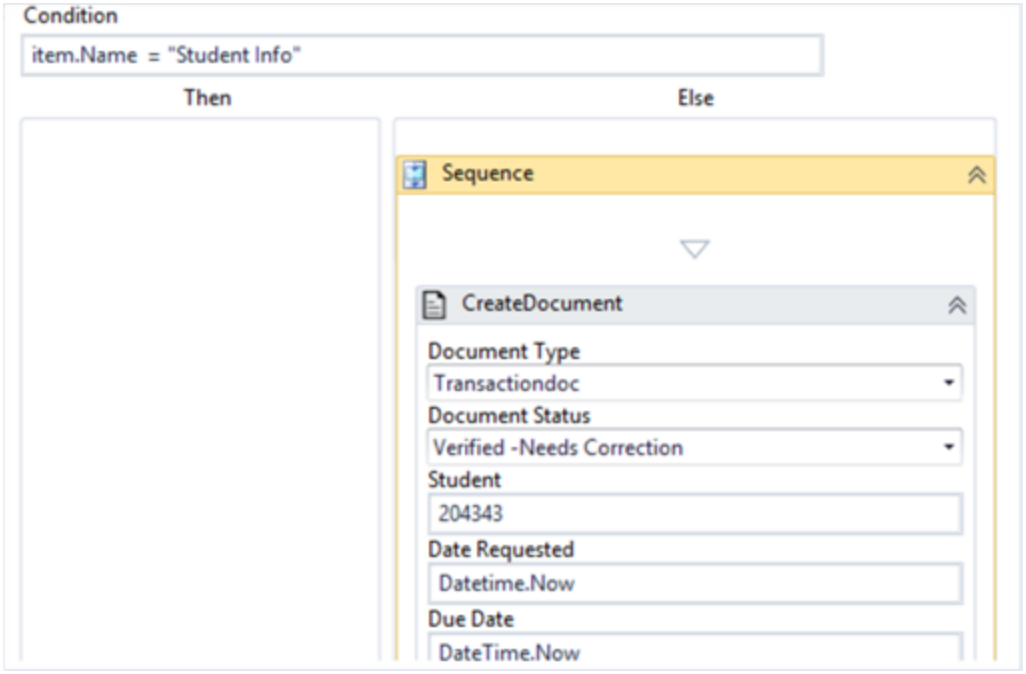
8. **Save** the workflow locally and continue with the next set of steps.

9. Drop a **CreateDocument** activity and specify the following under the **Else** condition:

   Specify the required properties for the activity as follows:

   - Module = Select a Module from the drop-down list.
   - Document Type = Select a Type (Template) from the drop-down list.
   - Document Status = Select a Type from the drop-down list.
   - Student = Specify a Student Id or use a variable.

- Due Date = Specify a date or use a variable, e.g., DateTime.Now
- Document (OutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
- Validation Messages = formInstance.ValidationMessages



10. Below the CreateDocument activity, drop an **Assign** activity for each row in the following table and type the indicated values:

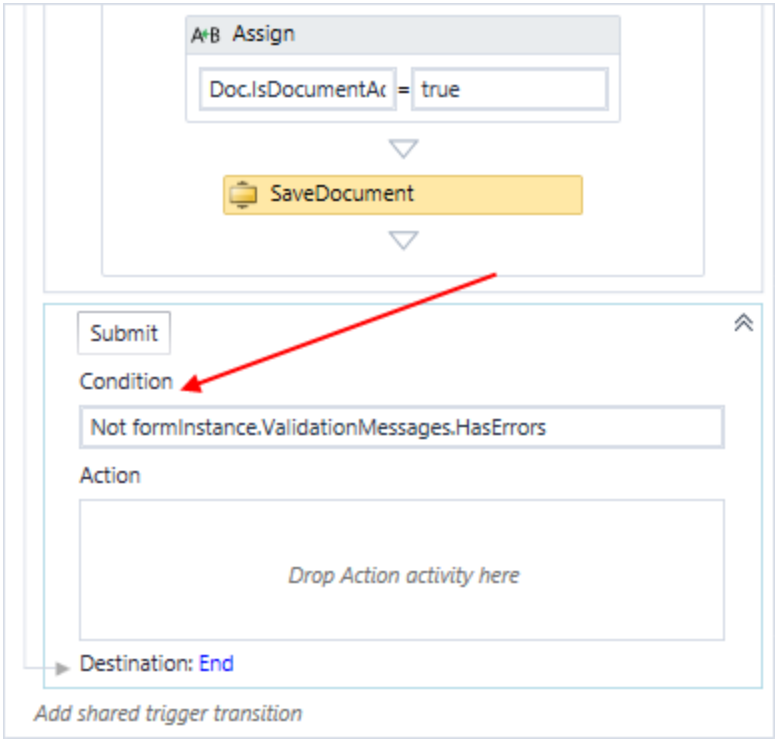
"To" Field	Value
Doc.DocumentImage	SignedDocument.Content
Doc.OriginalFileName	"Signedpdf.Pdf"
Doc.ImageType	"Pdf"
Doc.IsDocumentAddedManually	true

11. Drop a **SaveDocument** activity below the CreateDocument activity.

    Specify the properties for the SaveDocument activity as follows:

    - Document (InOutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
    - Validation Messages = **formInstance.ValidationMessages**

12. In the Condition field of the Submit transition, specify the following (to prevent users from submitting a form that has validation errors): **Not formInstance.ValidationMessages.Has Errors**



13. **Save** the workflow locally and continue with the next set of steps.

Final Steps

1. Select **Publish**. The New Workflow Definition Version window is displayed.
2. If you want the workflow to be run as soon as the event occurs on the entity, select **Enable This Workflow Version?**, otherwise leave the check box cleared.
3. Select **Save**, then **Cancel** to close the publisher window.
4. In the **Sequence List**, select and complete the form that contains the DocuSign fields.
5. Verify that the signed DocuSign document is available at the return URL.

While testing your workflow, make sure that you terminate the previous instance of the workflow before running an updated version of the same workflow. In Workflow Composer, select **Open Persisted Workflow**, select your workflow instance, and select **Terminate**.

Persisted workflow instances can be deleted from the Sequence Designer workspace. For more information, refer to Delete Persisted Workflow Instances.

Move from Test to Production

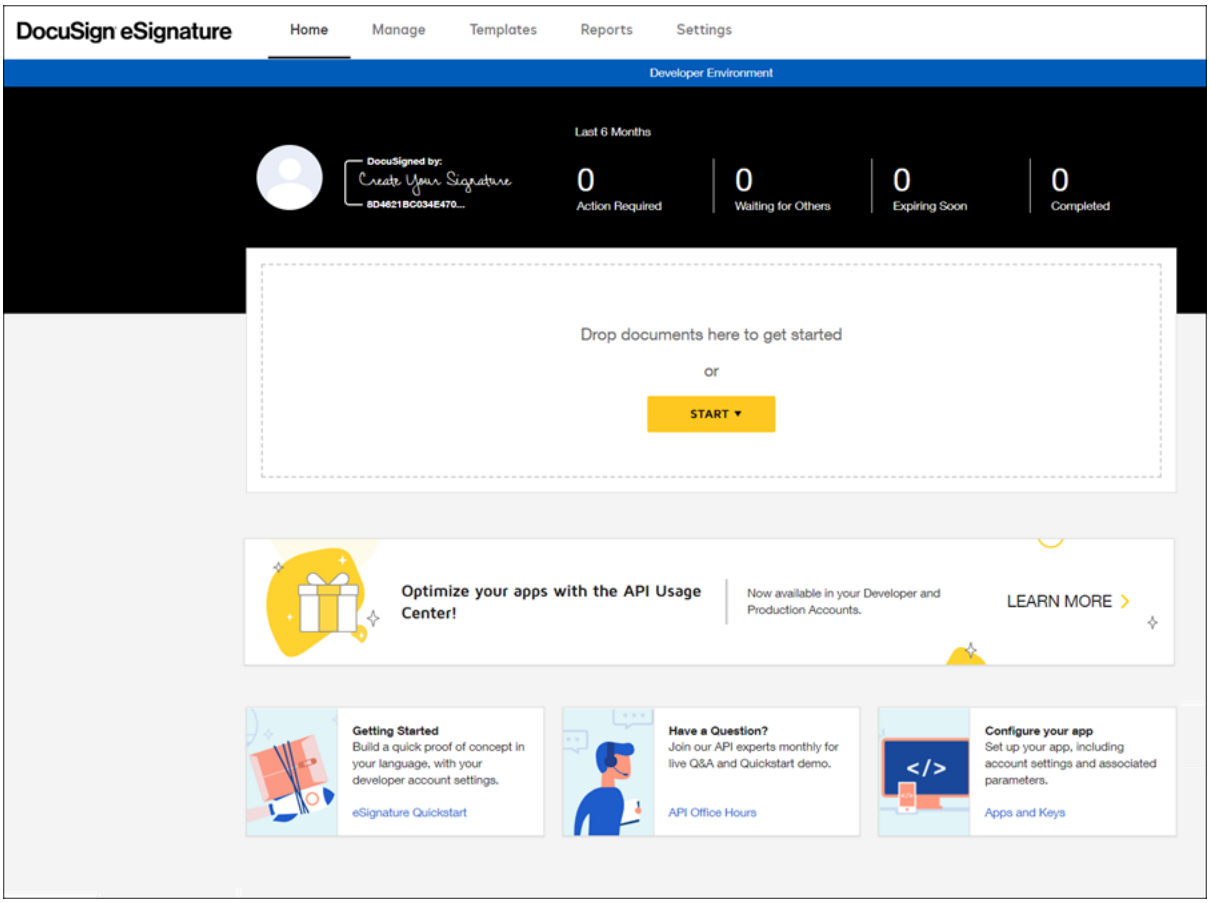
Perform these actions to ensure that your solution will function in the production environment.

- Update settings to integrate with DocuSign. For more information, refer to DocuSign Settings.
- If you used **DocuSignConfig.TestMode=true** in your DocuSign workflow, delete this assignment or change it to **false**.

  As of Forms Builder 3.4, the DocuSignConfig.TestMode assignment (=true or =false) is no longer supported or functional. Assign statements containing it can be deleted.
- Ensure that the account you are using has Send Envelope permissions.
- Once the setup is complete, call DocuSign to configure following items:
  - Enable the In-Session permission to your account.
  - Don't Enforce In Session Certificate.
- If you are planning to have multiple signers per document, enable *DocuSign Connect* to allow *Webhook*.
- If you use the resource files or templates in the demo environment, transition them to your production environment.
- For templates specifically, the Template ID will change from the demo environment to your production environment. Adjust your code accordingly.
- Perform post certification testing prior to the go-live date to verify functionality. Envelopes used for production testing will be credited to your account upon verification. Contact your account manager for details.

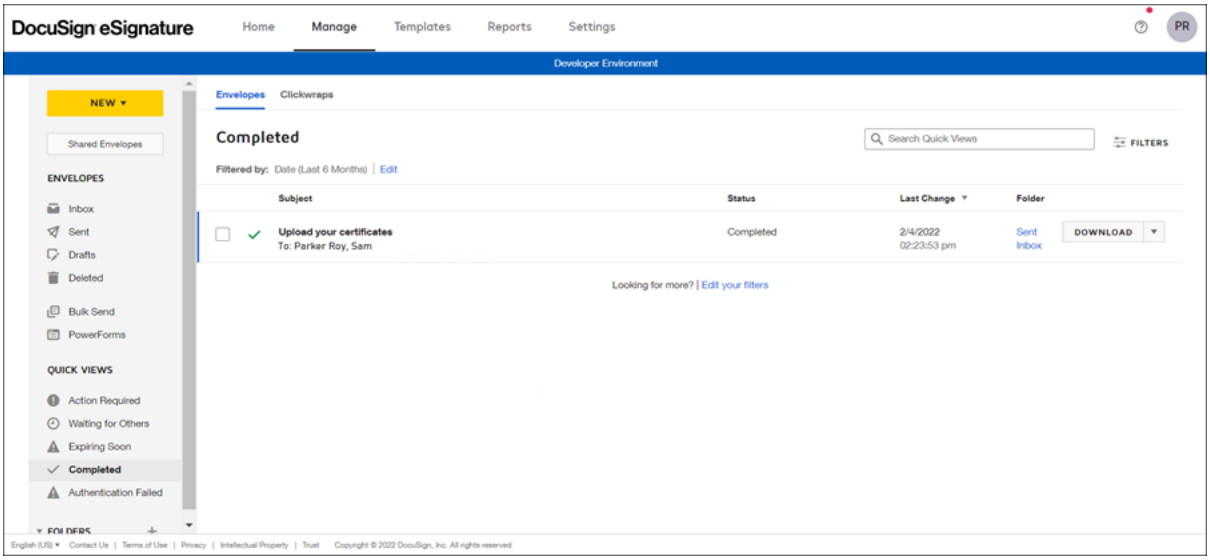
Log in to DocuSign

Users can log in to DocuSign (https://admin.docusign.com/) and access documents, history, permissions, etc.



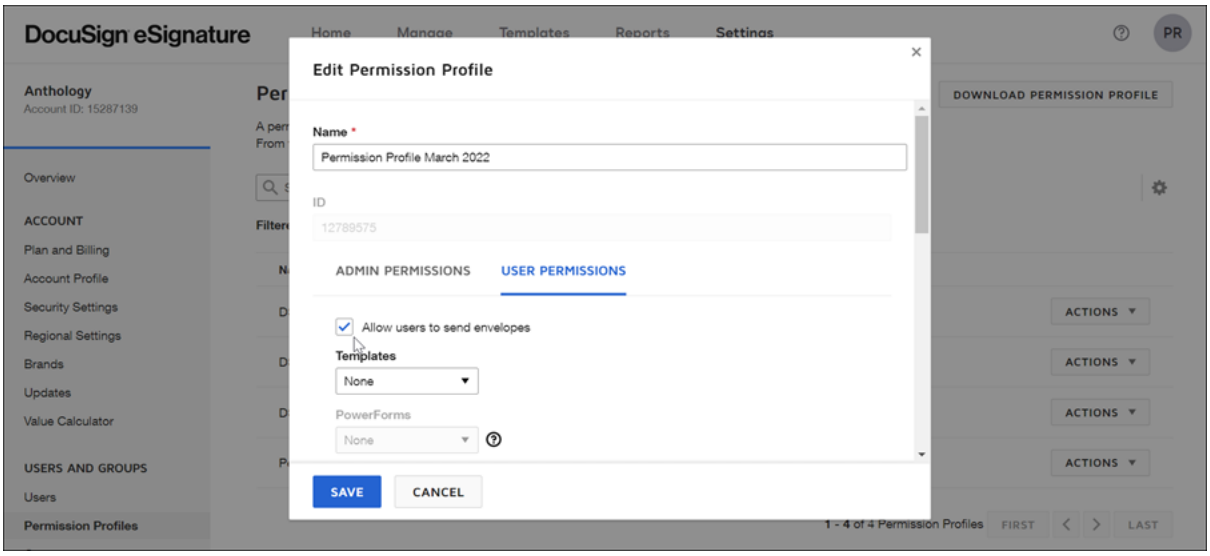
Manage Tab

Select the Manage tab to view the details.



Permissions

On the Permissions screen (Settings > Permission Profiles), edit the required profile and check if the **Allow users to send envelopes** permission is selected for your account.



SignNow Forms

Forms Builder 3.7 and later provides integration with SignNow, a cloud-based electronic signature service that lets you send, sign, track, and manage signature processes. For more information, refer to https://www.signnow.com/.

SignNow Settings



Important:

Before you can use SignNow with Forms Builder, you must:

- Obtain a SignNow **staff sender account** and a **signer account**.
  The sender and signer accounts must have the same email address.
- Purchase a SignNow **API subscription**.

For more information, refer to https://www.signnow.com/.

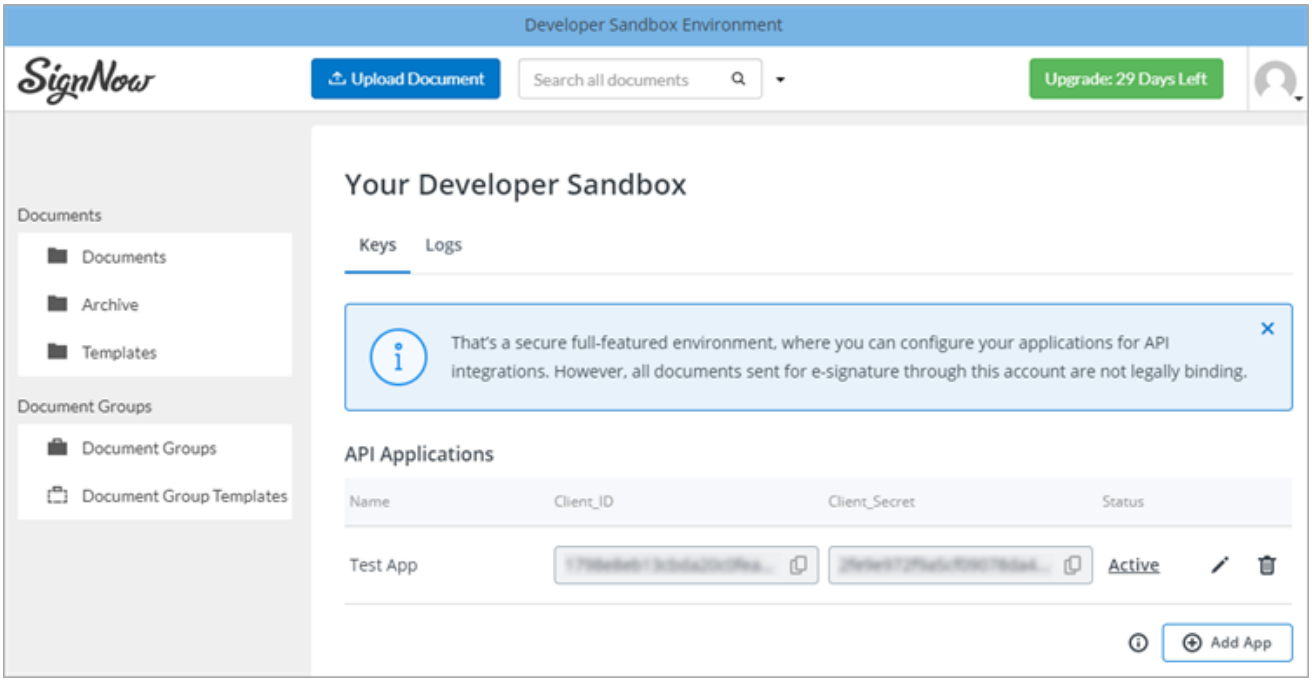
To enable SignNow signatures on Forms Builder forms, you need to configure your SignNow credentials and API key in Forms Builder Settings as described below.

The API key is only usable for the main account (sender account). The API plan currently comes with 1 seat for the sender account and signer accounts do not need a license.

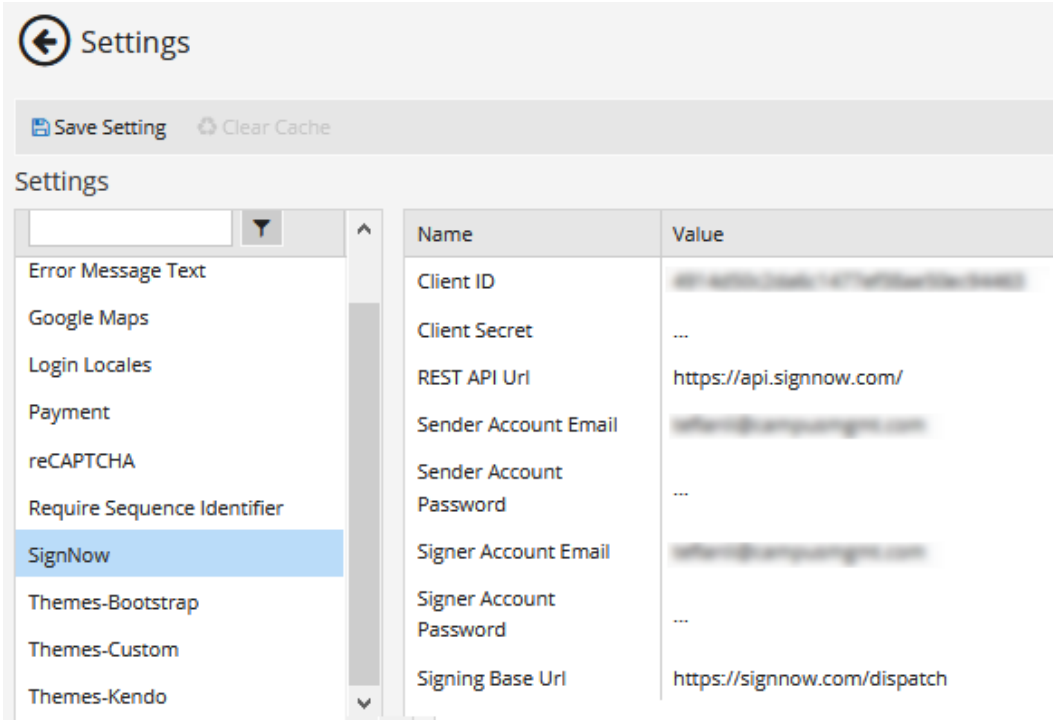
Set Up Forms Builder for SignNow

You can create a sandbox account to get your documents signed via the SignNow API for test purposes. When you are ready for production, contact your SignNow account manager and update the Forms Builder Settings with the appropriate information.

1. To create a sandbox account, go to https://www.signnow.com/api, enter your email address, and select **Get Started**.
2. Open the SignNow account activation message and select **Activate**.
3. Create a password and select **Activate Account**. Your developer sandbox page will display your **Client ID** and **Client Secret**.



4. On the Forms Builder home page, select the **Settings** tile and select **SignNow**. The Properties pane will be populated with the SignNow properties.



The settings require two URLs. Specify the correct Urls values depending on whether you are using SignNow in a test environment or in a production environment. The image above shows the Urls used in production.

Url	Testing	Production (example)
REST API Url	https://api-eval.signnow.com/	https://api.signnow.com/
Signing Base Url	https://eval.signnow.com/dispatch	https://signnow.com/dispatch

Contact your SignNow account manager or support@signnow.com to obtain the correct values for your production environment.

5. Complete the following fields in the Settings pane:

SignNow Properties

Name	Value
Client ID	Copy and paste the Client Id and Client Secret from the Developer Sandbox page. Contact your SignNow account manager or support@signnow.com to obtain the correct values for your production environment. These values are needed during the token exchange between Forms Builder and the SignNow API. The Client Secret is encrypted in the Forms Builder database.
Client Secret	
REST API Url	For test purposes use: https://api-eval.signnow.com/ Contact your SignNow account manager or support@signnow.com to obtain the correct values for your production environment.
Sender Account	Specify email address and password for the sender and signer account. Use the same email address both accounts.
Signer Account	
Signing Base Url	For test purposes use: https://eval.signnow.com/dispatch Contact your SignNow account manager or support@signnow.com to obtain the correct values for your production environment.

The SignNow testing Url and production Url do not share data and require separate keys.

⚠ Whenever you edit a field in a Settings panel with hidden fields (i.e., sensitive data that are removed from the client-side view like passwords, keys, and secrets), you must re-enter the hidden values, otherwise they will get cleared upon update and resave of all the Settings.

6. **Save** the Settings.
7. Once the SignNow configuration is complete, reset IIS or wait an hour for the changes to take effect.

Troubleshoot SignNow Settings

Error: Invalid Client

If your Forms Builder logs contain an error similar to the example below, correct the host URLs configured in the SignNow Settings.

2021-09-01 21:00:05.8169 ERROR - Cmc.Nexus.FormsBuilder.Services.SignNowService: Error getting access token: {"error":"invalid_client"}
2021-09-01 21:00:05.8169 ERROR - Cmc.Nexus.FormsBuilder.Services.SignNowService: Cmc.Nexus.FormsBuilder.Helpers.FormsBuilderException: Error getting access token: {"error":"invalid_client"}

Error: Access Denied

If your Forms Builder logs contain an error similar to the example below, change the signnow.apiuser, sender, and signer accounts in the SignNow Settings so that they use the same credentials and email addresses.

2021-09-01 22:00:05.2169 ERROR - Cmc.Nexus.FormsBuilder.Services.SignNowService: Error getting access token: {"errors":[{"code":11005001,"message":"Access denied."}]}
2021-09-01 22:00:05.3169 ERROR - Cmc.Nexus.FormsBuilder.Services.SignNowService: Cmc.Nexus.FormsBuilder.Helpers.FormsBuilderException: Error getting access token: {"errors":[{"code":11005001,"message":"Access denied."}]}
at Cmc.Nexus.FormsBuilder.Services.SignNowService.GetAccessToken(String userEmail, String userPassword, String documentId) in C:\Agt1_work\I\s\Cmc.Nexus.FormsBuilder\Services\SignNowService.cs:line 350
at Cmc.Nexus.FormsBuilder.Services.SignNowService.CreateSignNowRequest(CreateSignNowRequest request) in C:\Agt1_work\I\s\Cmc.Nexus.FormsBuilder\Services\SignNowService.cs:line 109
2021-09-01 22:00:05.4169 ERROR - Cmc.Nexus.FormsBuilder.Workflow.CreateSignNowRequest: CreateSignNowRequest had errors for WorkflowInstanceId: 1bae1731-bc64-8885-c2617081f663. Validation message: Error getting access token

SignNow Workflow Sample - Single Signer

The following procedure details a sample workflow and forms for a sequence that contains single signer.

Prerequisites

1. The SignNow properties are configured in Forms Builder. Refer to SignNow Settings.
2. A form sequence with the necessary SignNow components for a single signer is created. In our example, the **Anonymous** property of the sequence is selected.

   If you create a non-authenticated sequence, make sure that the user's email address is included in the sequence. The email address is required as the SignNow recipient address.
3. A form that contains an IFrame component is created. In our example, this form is named *Default-Frame* with the following properties (case sensitive):
   - Name = **SignNowFrame**
   - Url = **{{vm.models.frameUrl}}**

   The IFrame form can be part of the initial sequence; however, in the procedure below, we are adding the form to the workflow.

   > ⚠ We recommend that you copy the original *Default-Frame* form, edit the copy, and use it in your sequences. Save a backup copy of your form.

To optimize the display of your e-signature forms on mobile devices, you may want to add the CSS described in this Note.

Create the Workflow

Open the Workflow and Review Arguments, Variables, and Logging Requirements

1. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
2. In Workflow Composer, drag the State icons and Transition lines to so that you can easily locate each item in the StateMachine workflow.

The workflow requires the following arguments and variables. You can create these arguments and variables before working on the activities, or you can add them when they are needed for a specific activity (as described below).

> ⚠ Keep in mind that arguments are passed in JSON format and that JSON elements are case sensitive. **Be sure to match the casing of argument names in Workflow Composer and Form Designer.**

Arguments

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*
studentEntity	In/Out	StudentEntity	*Default value not supported*
frameUrl	In/Out	String	*Default value not supported*

Variables

Name	Variable type	Scope	Default
renderedFormImage	String	StateMachine	*Enter a VB expression*
Pdf	Byte[]	StateMachine	*Enter a VB expression*
SignNowConfig	SignNowConfig	StateMachine	*Enter a VB expression*
Doc	DocumentEntity	StateMachine	*Enter a VB expression*
URL	String	StateMachine	*Enter a VB expression*
studentId	Int32	StateMachine	*Enter a VB expression*
SignNowDocument	SignNowDocument	StateMachine	New SignNowDocument
SignNowRequest	SignNowRequest	StateMachine	*Enter a VB expression*
SignNowRecipients	SignNowRecipient[]	StateMachine	*Enter a VB expression*
SignNowRecipient	SignNowRecipient	StateMachine	New SignNowRecipient

LogLine Activities

While testing and troubleshooting the workflow, we recommend adding LogLine activities at critical stages within the workflow. The following expression will provide logging for specific objects (replace <object> with the object name):

Newtonsoft.Json.JsonConvert.SerializeObject(**<object>**, Newtonsoft.Json.Formatting.Indented)

For example, to capture errors related to the CreateSignNowRequest activity, you would place a LogLine activity with the following expression below the CreateSignNowRequest activity.

Newtonsoft.Json.JsonConvert.SerializeObject(**SignNowRequest**, Newtonsoft.Json.Formatting.Indented)

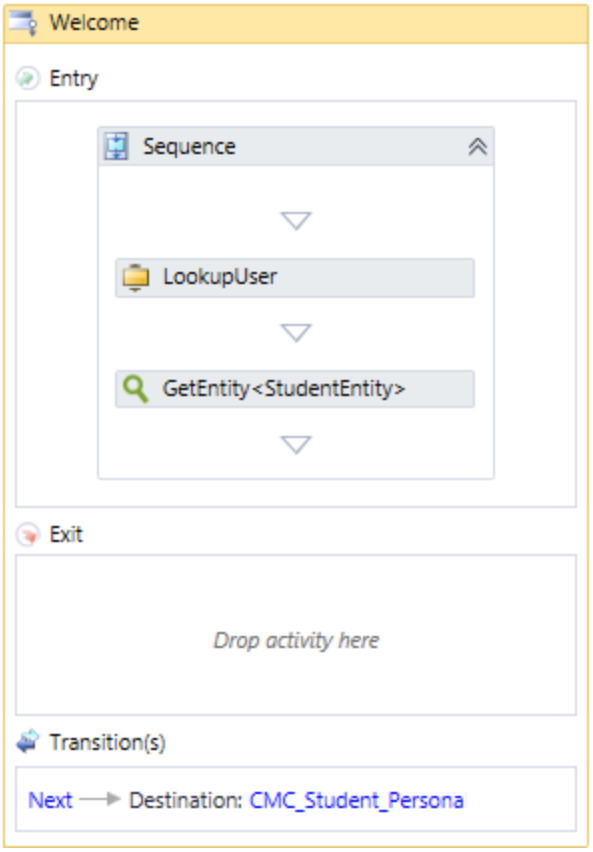
We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

Capture the Login User Information

If your sequence is non-authenticated (Anonymous=true), the LookupUser and GetEntity activities are not applicable, but a CreateEntity activity may be needed instead. Be sure that any anonymous sequence includes the user's email address. It is required as the SignNow recipient address.

The first step is to capture the information of the user who logged in to the form.

1. Create a variable named **studentId** of Type **Int32**.
2. Drop a **LookupUser** activity in the Entry area of the Welcome form.
    - In the UserId property, specify **studentId**. This is the variable created in the previous step.
    - In the UserName property, specify **formInstance.UserName**. The user name is retrieved from the login account information.



3. Drop a **GetEntity** activity in the below the LookupUser activity.
    - In the EntityId property, specify **studentId**.
    - In the Result property, specify **studentEntity**.

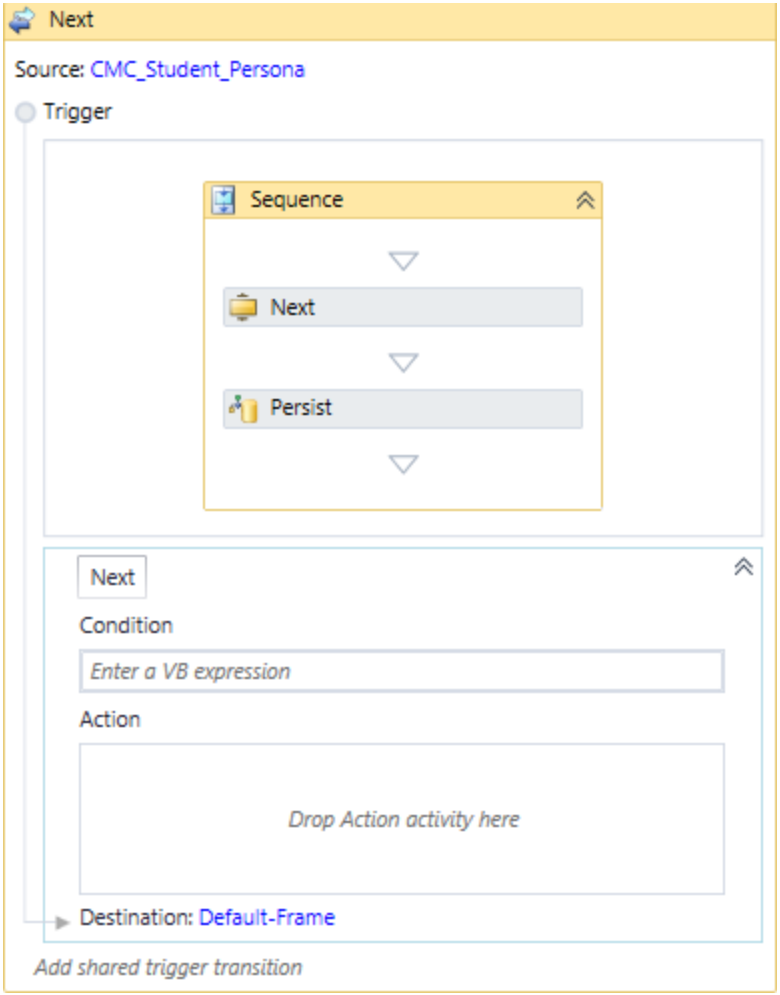
    Workflow Composer automatically wraps the activities in a Sequence.
4. **Save** the workflow locally and continue with the next set of steps.

Persist the Workflow Instance After Collecting the Form Data

Since the workflow can be busy for a long time, we recommend that you persist the workflow instance before creating the PDF.

1. Double-click the **Next** transition from the form that is used to collect the data, *CMC_Student_Personal Info*.
2. Drag a **Persist** activity into the Trigger area below the Next activity. The Persist activity does not require any properties to be specified.

   Workflow Composer automatically wraps the Next and Persist activities in a Sequence.



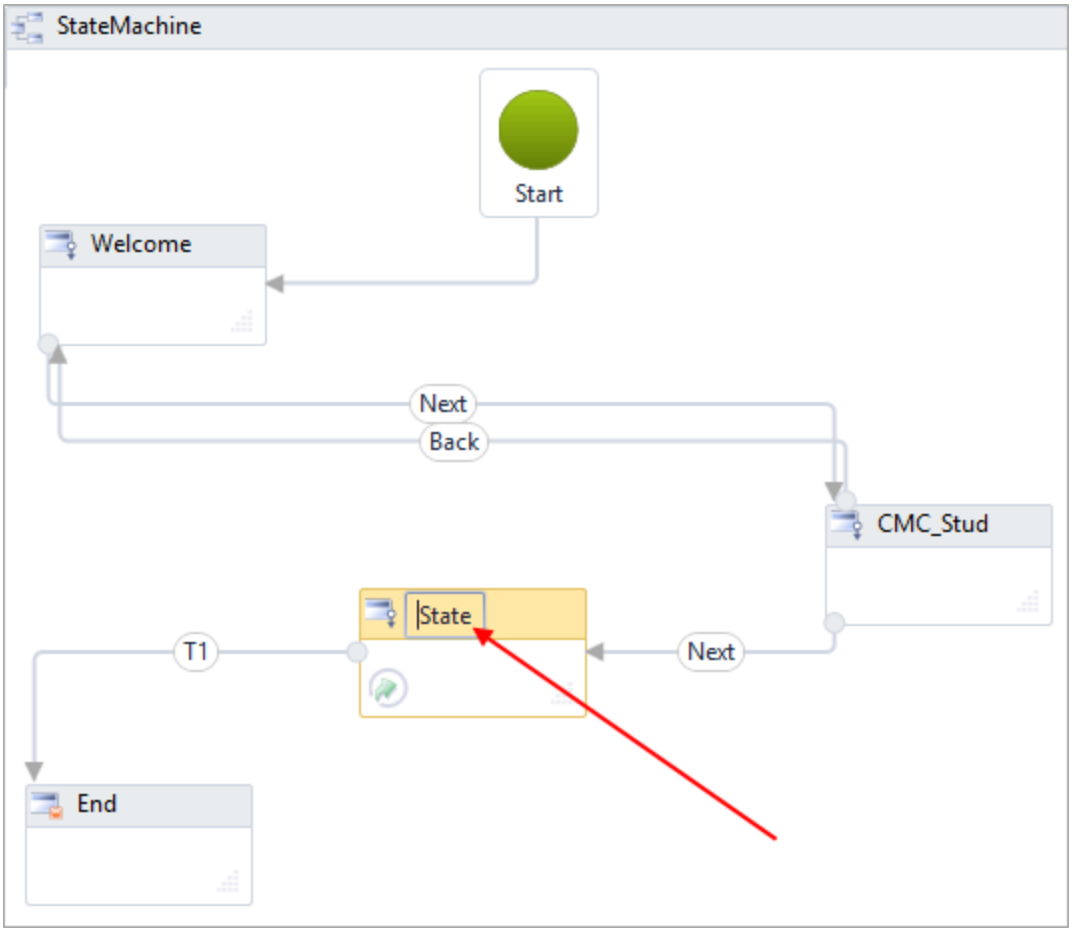
3. **Save** the workflow locally and continue with the next set of steps.

Add an IFrame Form to the Workflow

1. In Workflow Composer, select the **Arguments** tab.
2. Create an argument named **frameUrl**. Set the Direction to **In/Out**. Set the Type to **String**.

   Be sure to use the exact casing shown here.
3. Select the **StateMachine** link in the Designer pane.

4. Drop a **State** activity onto the transition line after the Next transition from the form that contains the SignNow components (*CMC_Student_Personal Info*).



5. Change the default name of the State activity to the name of the form that contains the IFrame. In our case, the name is **Default-Frame**.
6. **Save** the workflow locally and continue with the next set of steps.

Create a PDF of the SignNow Form

In the following steps we will create a set of activities in the *Default-Frame* State, which represents the IFrame form.

1. Double-click the State icon of the form that contains the IFrame component.
2. Create a variable named **URL**. Set the Variable type to **String**. Set the Scope to **StateMachine**. This variable will be assigned to the PDF that will be created from the form.
3. Drop an **Assign** activity into the Entry area of the State.

a. In the *To* field, specify the variable name **URL**.

b. In the *Value* field, specify the following:

forminstance.RendererBaseUrl + "#/viewCreator/" + formInstance.WorkflowDefinitionId.ToString + "/forms=CMC_Student_Personal_SignNow"

Where "/forms=*CMC_Student_Personal_SignNow*" indicates the Forms Builder form that contains the SignNow components.

- If the form name has a space, replace the space with a + sign, e.g., *CMC_Student_Personal+Info_SignNow*
- If multiple forms are sent to SignNow, specify a comma-separated list of form names in the "/forms=" attribute.

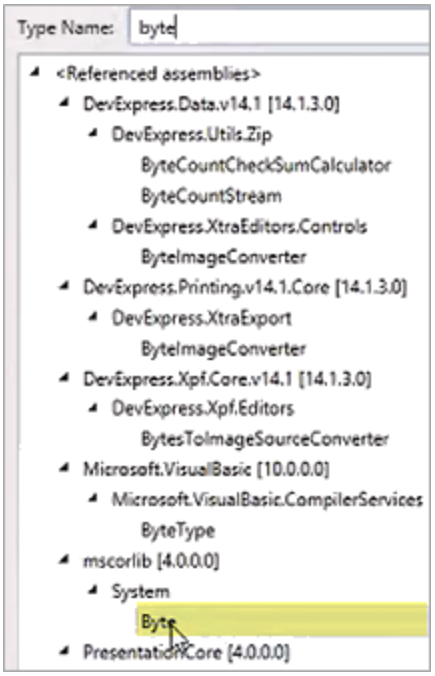
4. Drop a **Persist** activity below the Assign activity.

5. Create a variable with a name like **Pdf**. Set the Scope to **StateMachine**. This variable will hold the document image created from the form.

Make sure you use the same variable name when you reference this variable later in the workflow. (This applies to any other variable.)

a. In the Variable type field, select **Array of [T]** and select **Browse for Type**.

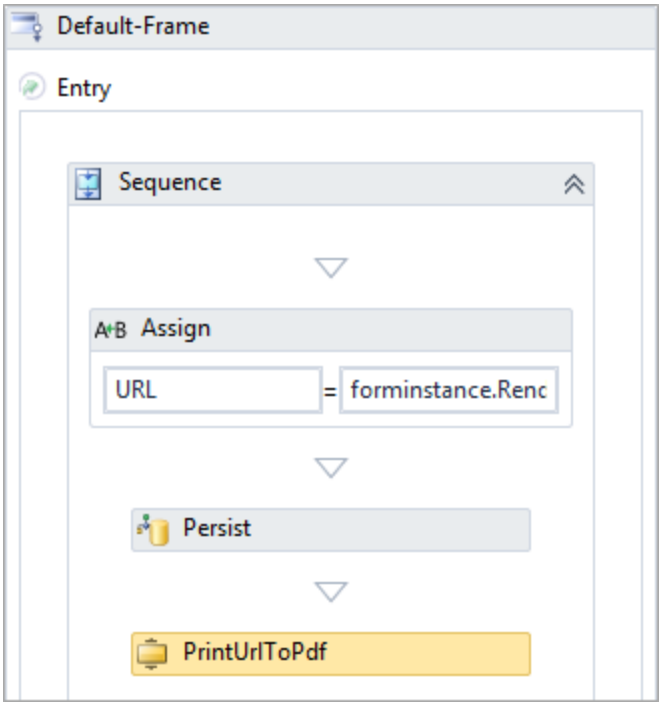
b. In the "Browse and Select a .Net Type" window, specify "**byte**", select the System variable "**Byte**", and select **OK**.



6. Drop a **PrintUrlToPdf** activity below the Persist activity.

Specify the properties for the activity as follows:

- PdfDocument = **Pdf** (This is the name of the variable created above.)
- Url = **URL** (This is the name of the variable created above.)
- Validation Messages = **formInstance.ValidationMessages**



7. **Save** the workflow locally and continue with the next set of steps.

Get the SignNow Configuration and Pass the Recipient Information

In the following steps we will continue to add activities to the *Default-Frame* State, which represents the IFrame form.

1. Create a variable named **SignNowConfig**. Set the Scope to **StateMachine**.

   In the Variable type field, select **Browse for Type**. In the "Browse and Select a .Net Type" window, scroll down to Cmc.Nexus.FormsBuilder.Contracts > Cmc.Nexus.FormsBuilder.Entities and select **SignNowConfig**.

2. In the State of the IFrame form, drop a **GetSignNowConfig** activity below the PrintUrlToPdf activity.

   Specify the properties for the activity as follows:

   - SignNowConfig = **SignNowConfig**
   - Validation Messages = **formInstance.ValidationMessages**

The GetSignNowConfig activity retrieves the User Name, Password, Integrators Key, and REST API Url from the SignNow settings in Forms Builder. These values enable the workflow to log in to SignNow.

To process the SignNow request, the Email Subject needs to be assigned to the SignNowConfig variable. This property is required.

3. Create the following variables:

Variable Name	Type	Scope	Default
SignNowDocument	Cmc.Nexus.FormsBuilder.Entities.SignNowDocument	StateMachine	New SignNowDocument
SignNowRecipient	Cmc.Nexus.FormsBuilder.Entities.SignNowRecipient	StateMachine	New SignNowRecipient
SignNowRequest	Cmc.Nexus.FormsBuilder.Entities.SignNowRequest	StateMachine	N/A

4. Below the GetSignNowConfig activity, drop an **Assign** activity for each row in the following table and type the indicated values:

"To" Field	Value	Notes
SignNowConfig.EmailSubject	"CMC Campus SignNow Testing"	This is the email that the end user will receive after the signing process is done.
SignNowDocument.DocumentId	"1"	When multiple SignNow documents are included in the same SignNow envelope, every document needs a unique Id. In this example, there is only one document.
SignNowDocument.Name	"StudentInfoPdf.pdf"	Name of the signed document.
SignNowDocument.Content	Pdf	Name of the variable that holds the document image (refer to Create a PDF of the Form). In our example the name of the variable is **Pdf**.
SignNowRecipient.FirstName	studentEntity.FirstName	First name of the user who submitted the signed document.
SignNowRecipient.LastName	studentEntity.LastName	Last name of the user who submitted the signed document.
SignNowRecipient.Email	SignNowConfig.SignerAccountEmail	Email address of the person who will receive the signed document.

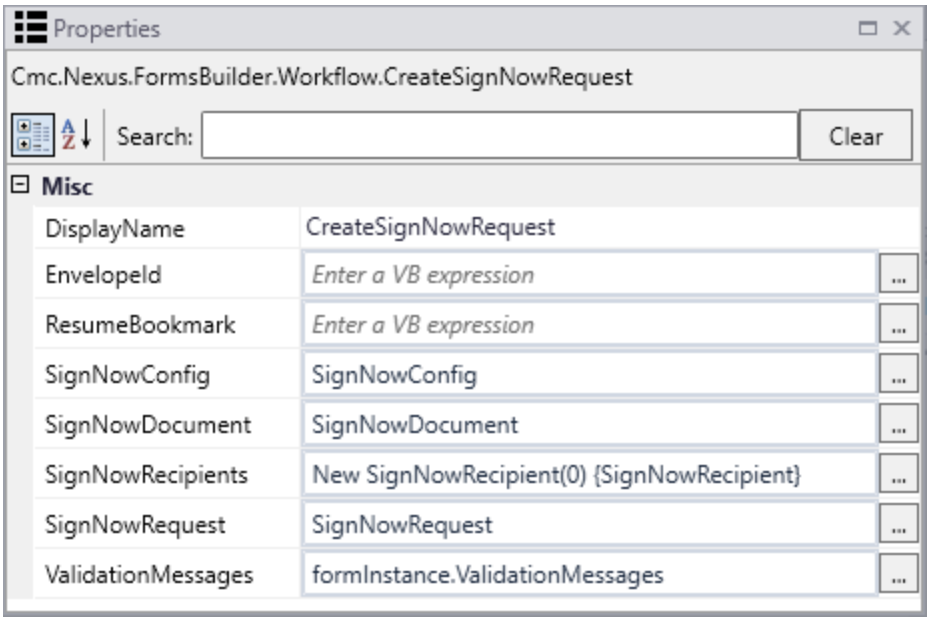
5. **Save** the workflow locally and continue with the next set of steps.

Create the SignNow Request and Specify the IFrame URL

In the following steps we will continue to add activities to the *Default-Frame* State, which represents the IFrame form.

1. Drop the **CreateSignNowRequest** activity below the previous activity.

   Specify the properties for the CreateSignNowRequest activity using the names of the variables created above as follows:



   Warning: Do not enter anything in ResumeBookmark if you are creating Single Signer workflows. This could cause InstanceHandleConflictExceptions and aborted workflows. Fill this in only for Multi-Signer workflows.

2. Drop an **If** activity below the CreateSignNowRequest activity.

   a. In the *Condition* field, specify **formInstance.ValidationMessages.HasErrors**.

   b. In the *Else* branch, drop an **Assign** activity.

      In the *To* field, specify **frameUrl**. (This is the argument associated with the IFrame form.)

      In the *Value* field, specify **SignNowRequest.Url**.

      ⚠ Be sure to use the exact casing shown here.

3. **Save** the workflow locally and continue with the next set of steps.

Receive the Signed SignNow Document

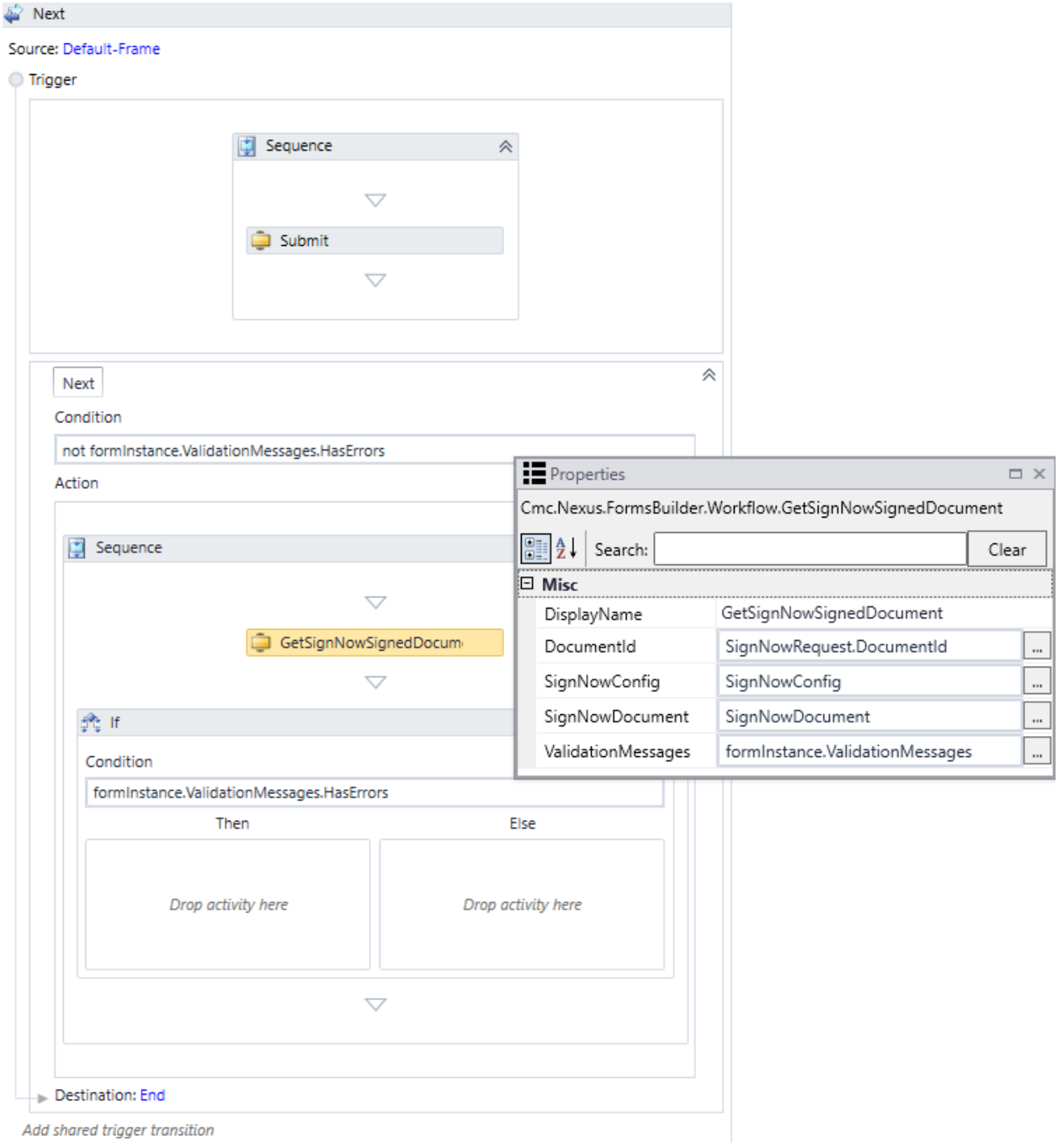
In the following steps we will create activities in the Submit transition which follows the IFrame form.

1. Double-click the **T1** transition after the *Default-Frame* state and rename it as **Submit**.
2. Drop a **WaitForFormBookmark** activity into the Trigger area of the Submit transition.
3. Rename the WaitForFormBookmark activity as **Submit**.
4. Create a variable named **SignedNowDocument**. In the Variable type field, select **SignNowDocument**. Set the Scope to **StateMachine**.
5. Drop a **GetSignNowSignedDocument** activity below the Submit activity.

   Specify the properties for the activity as follows:

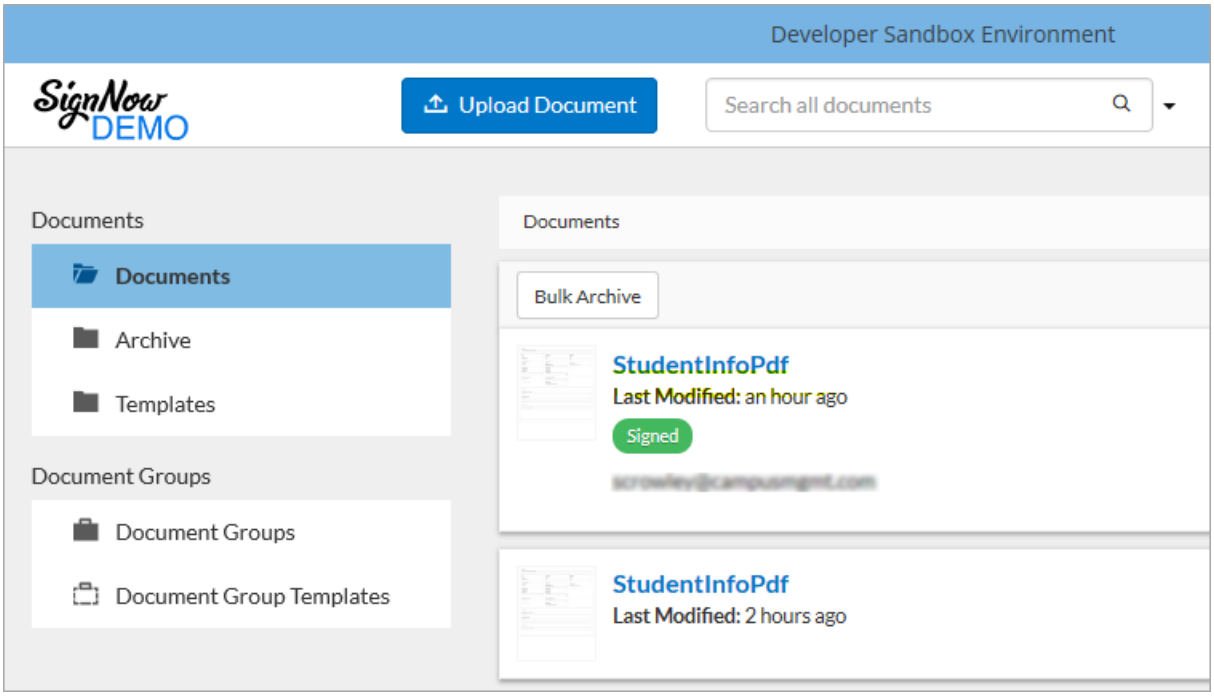
   - DocumentId= **SignNowRequest.DocumentId**
   - SignNowConfig = **SignNowConfig**
   - SignNowDocument = **SignedNowDocument** (This is the name of the variable created above.)
   - Validation Messages = **formInstance.ValidationMessages**

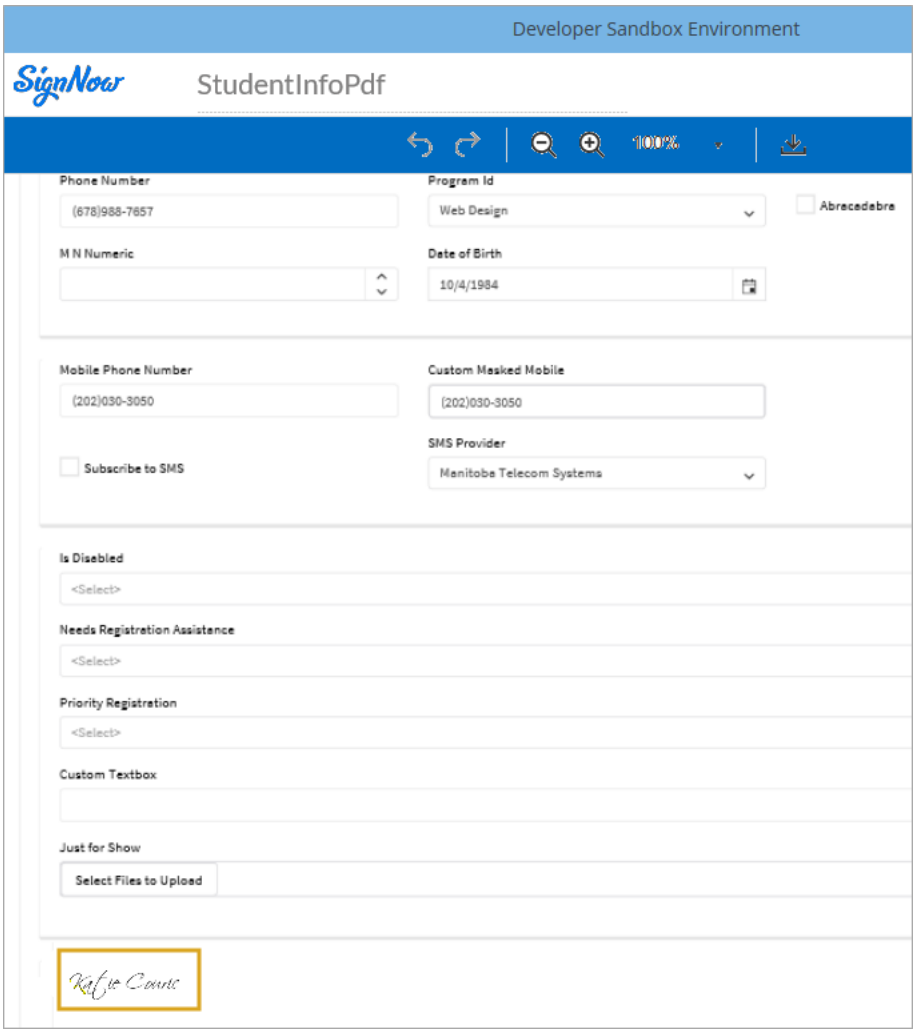
   Workflow Composer automatically wraps the activities in a Sequence.



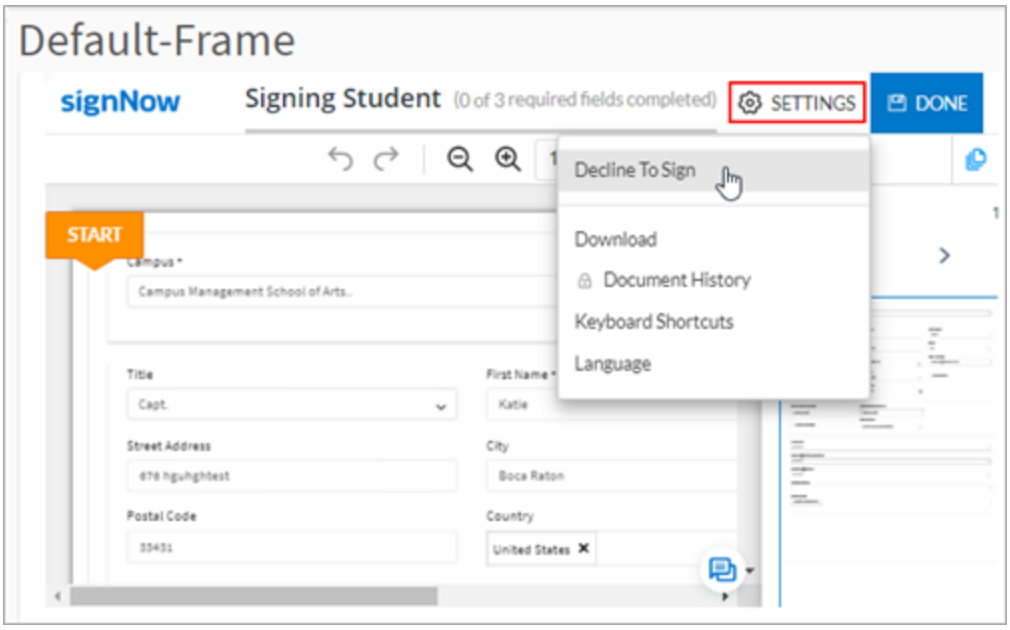
6. **Save** the workflow locally and continue with the next set of steps.

You can log in to your SignNow account and view the signed document "StudentInfoPdf".


Developer Sandbox Environment
SignNow DEMO
Upload Document
Search all documents
Documents
Documents
Archive
Templates
Document Groups
Document Groups
Document Group Templates
Documents
Bulk Archive
StudentInfoPdf
Last Modified: an hour ago
Signed
StudentInfoPdf
Last Modified: 2 hours ago


Developer Sandbox Environment
SignNow
StudentInfoPdf
100%
Phone Number
(678)988-7657
Program Id
Web Design
Abracadabra
M N Numeric
Date of Birth
10/4/1984
Mobile Phone Number
(202)030-3050
Custom Masked Mobile
(202)030-3050
Subscribe to SMS
SMS Provider
Manitoba Telecom Systems
Is Disabled
<Select>
Needs Registration Assistance
<Select>
Priority Registration
<Select>
Custom Textbox
Just for Show
Select Files to Upload

SignNow provides signers the **Decline To Sign** option.



If the decline option is selected:

- The signer will be presented with a popup stating "*You've declined to sign <PDF name>. <SignNow account email address> has been notified of your decision. You can always sign the document later if you change your mind*."
- SignNow will not return control of the sequence to Forms Builder and the workflow will not continue. The workflow will remain as a persisted instance. You will need to terminate persisted workflows periodically using either Sequence Designer (refer to Delete Persisted Workflow Instances) or Workflow Composer (refer to Persisted Workflows).

Create and Save the Document in Anthology Student

To convert a SignNow document to a DocumentEntity that can be attached to a record in Anthology Student, add CreateDocument and SaveDocument activities to the workflow. These activities will be placed in the Submit transition.

1. Create a variable named **Doc**. Set the Scope to **StateMachine**.

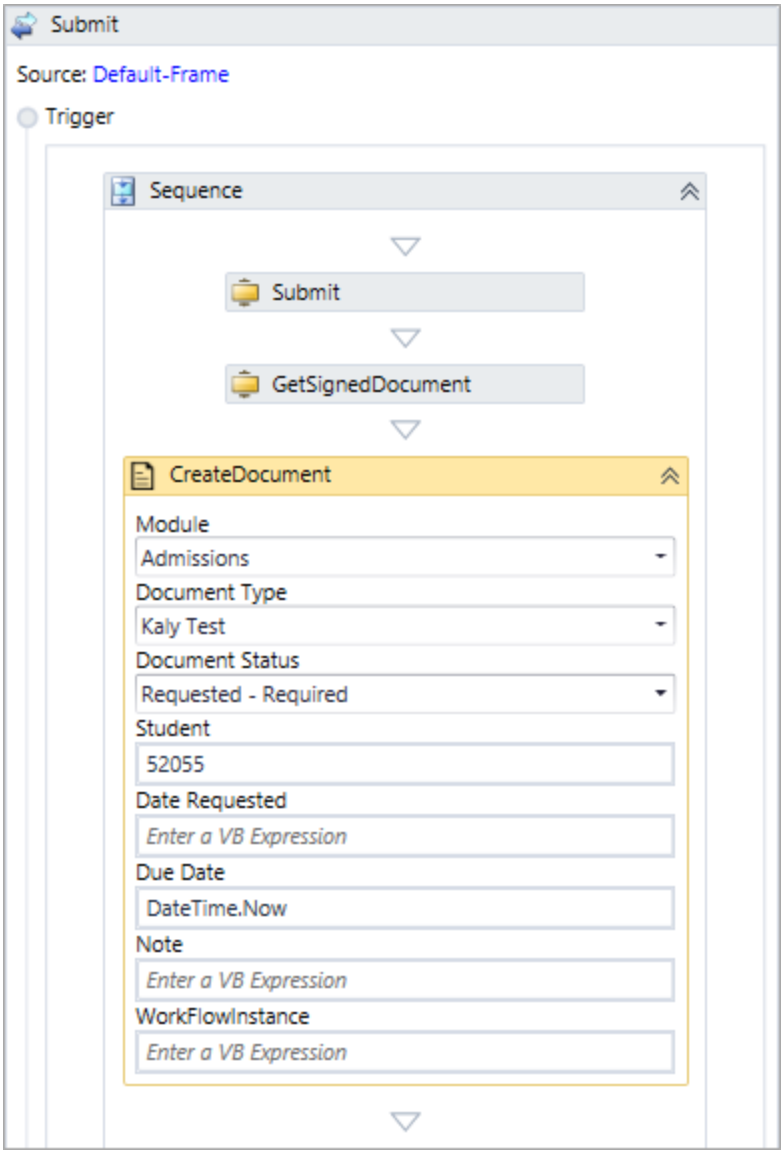
   In the Variable type field, select **Browse for Type**. In the "Browse and Select a .Net Type" window, scroll down to Cmc.Nexus.Crm.Contracts > Cmc.Nexus.Crm.Entities and select **DocumentEntity**.

2. Drop a **CreateDocument** activity below the GetSignedDocument activity.

   Specify the required properties for the activity as follows:

   - Module = Select a Module from the drop-down list.
   - Document Type = Select a Type (Template) from the drop-down list.

- Document Status = Select a Type from the drop-down list.
- Student = Specify a Student Id or use a variable.
- Due Date = Specify a date or use a variable, e.g., DateTime.Now
- Document (OutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
- Validation Messages = formInstance.ValidationMessages



3. Below the CreateDocument activity, drop an **Assign** activity for each row in the following table and type the indicated values:

"To" Field	Value
Doc.DocumentImage	SignedDocument.Content
Doc.OriginalFileName	"Signedpdf.Pdf"

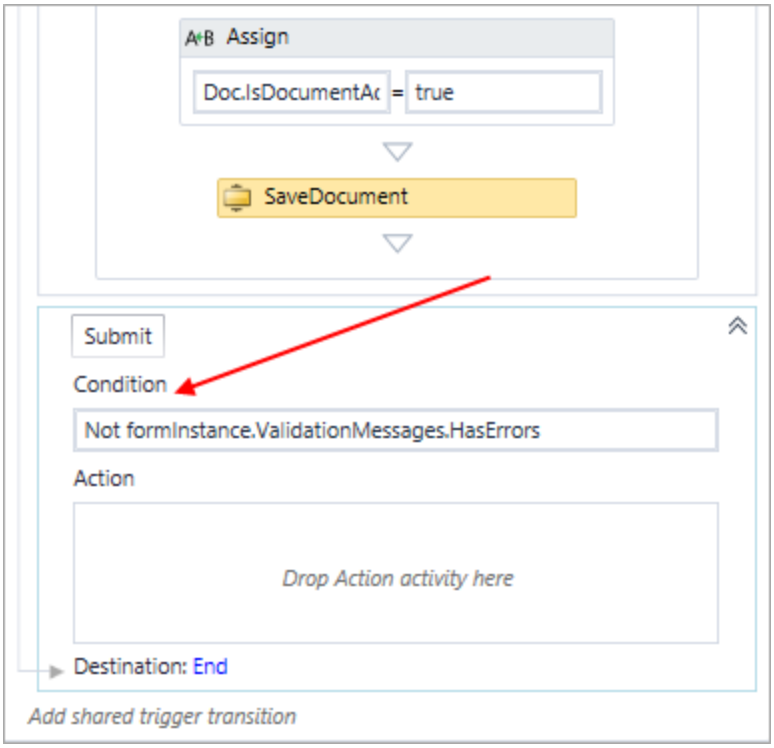
"To" Field	Value
Doc.ImageType	"Pdf"
Doc.IsDocumentAddedManually	true

4. Drop a **SaveDocument** activity below the CreateDocument activity.

   Specify the properties for the SaveDocument activity as follows:

   - Document (InOutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
   - Validation Messages = **formInstance.ValidationMessages**

5. In the Condition field of the Submit transition, specify the following (to prevent users from submitting a form that has validation errors): **Not formInstance.ValidationMessages.Has Errors**



6. **Save** the workflow locally and continue with the next set of steps.

Final Steps

1. Select **Publish**. The New Workflow Definition Version window is displayed.
2. If you want the workflow to be run as soon as the event occurs on the entity, select **Enable This Workflow Version?**, otherwise leave the check box cleared.
3. Select **Save**, then **Cancel** to close the publisher window.
4. In the **Sequence List**, select and complete the form that contains the SignNow fields.

5. Verify that the signed SignNow document is available at the return URL.
6. Verify that the signed document is added to the Anthology Student database.

While testing your workflow, make sure that you terminate the previous instance of the workflow before running an updated version of the same workflow. In Workflow Composer, select **Open Persisted Workflow**, select your workflow instance, and select **Terminate**.

Persisted workflow instances can be deleted from the Sequence Designer workspace. For more information, refer to Delete Persisted Workflow Instances.

SignNow Workflow Sample - Multiple Signers

The following procedure details a sample workflow and forms for a sequence that contains multiple signers.

Prerequisites

1. The SignNow properties are configured in Forms Builder. Refer to SignNow Settings.
2. A form sequence is created. In our example, the sequence includes the following forms:
    a. *Welcome*
    b. *TE_Personal_Info_Multi* — This is an admissions application form that collects the personal data of a student and contains four SignNow components (signature and date fields for two signers).

    The signing process for the primary signer is the same as described in SignNow Workflow Sample - Single Signer. The signing process for the primary signer is an embedded process within the form window. For the secondary signer, SignNow sends an email along with the document to be signed to the secondary signer. The signing process for primary and secondary signer takes place within the same SignNow request. SignNow returns events when each signer signs the document and when the document is complete. The workflow responds to these events.

    c. *SignNowCoSigner* — This form informs the student that a co-signer is needed and provides three text boxes for the co-signer's first name, last name, and email address.
    d. *Default-Frame* — This form contains an IFrame component.

    > ⚠ We recommend that you copy the original *Default-Frame* form, edit the copy, and use it in your sequences. Save a backup copy of your form.

    In your copy of the form, edit the Name property as shown below (case sensitive). Keep the URL property as {{vm.models.frameUrl}}.

    Name = *signNowFrame*

    e. *Default-DocuSignWait* — This form is provided with the Forms Builder installation. For this example, we have copied and renamed it to form to *SignNowWait*. It contains two HTML components.

    The first HTML component displays the following text. Edit the text in your copy of the form.

    "*Current user may close this browser tab. This page will auto submit when all the signatures are collected from DocuSign.*"

    The second HTML component contains JavaScript code that is not applicable to SignNow. Remove

that HTML component in your *SignNowWait* form.

> ⚠ The *Default-DocuSignWait* form will be overwritten when Forms Builder is upgraded. Copy the original form, rename it to *SignNowWait*, edit the copy, and use it in your sequences. Save a backup copy of your form.

This logic can be repeated as needed for additional signers.

To optimize the display of your e-signature forms on mobile devices, you may want to add the CSS described in this Note.

Test the Multiple Signer Feature

To test the multiple signature feature with SignNow, the test environment needs to meet specific requirements. SignNow will try to call APIs on Renderer when secondary signers complete their signing process via email. This feature is referred to as *Webhook*.

- If you are testing with a SignNow account in test mode:
  - Renderer must be hosted on port 80 or 443.
  - Port 80 or 443 must be open on the firewall depending on which port Renderer is using for SignNow to communicate. You can request an exception from your IT team for port 80 or 443 for an IP address range.
- If you are testing with a live SignNow account:
  - Renderer must be hosted on port 443 (https).
  - If hosting Renderer on port 443 is not possible, a SignNow representative for your account will have to make an exception.

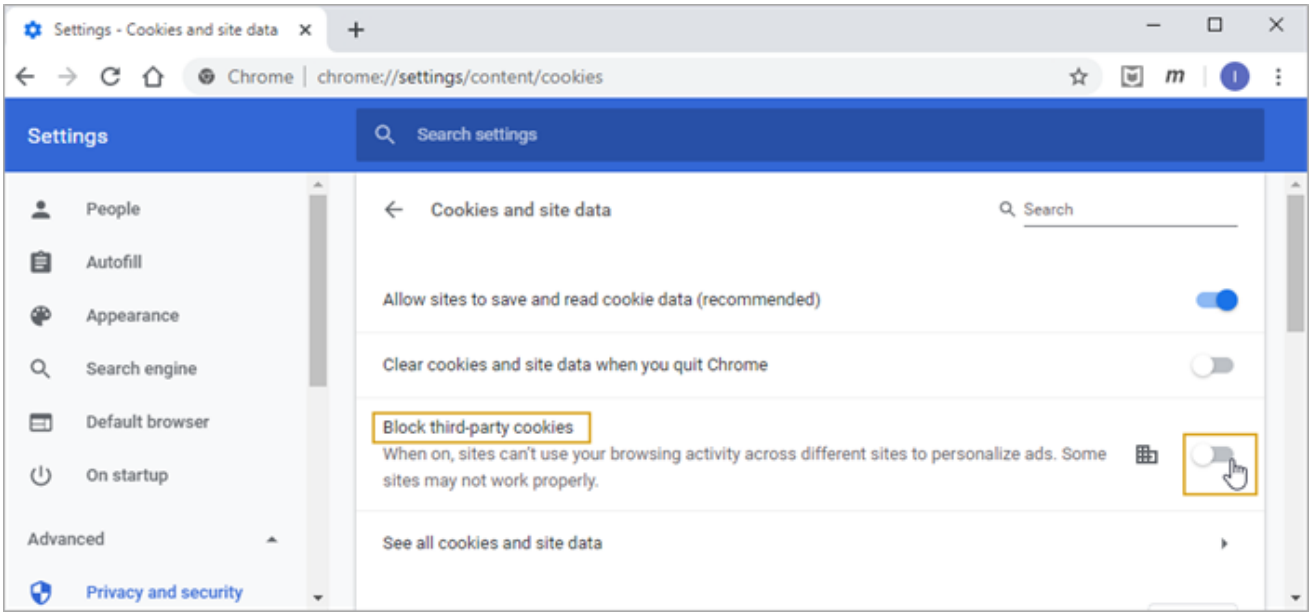
To confirm that multi-signature forms are completing the workflow process, please confirm that Webhook responses are received and displayed in the log. For more information, refer to Webhook Responses for Multiple Signer Sequences.

Chrome Browser Settings for Multiple Signers

The multiple signer feature for SignNow requires a specific setting in Google Chrome. If you experience issues with multiple signer forms, ensure that third-party cookies are allowed.

1. In Chrome, select **Settings**.
2. Select **Advanced** at the bottom of the page.
3. Expand **Site Settings**.
4. Expand **Cookies and site data**.

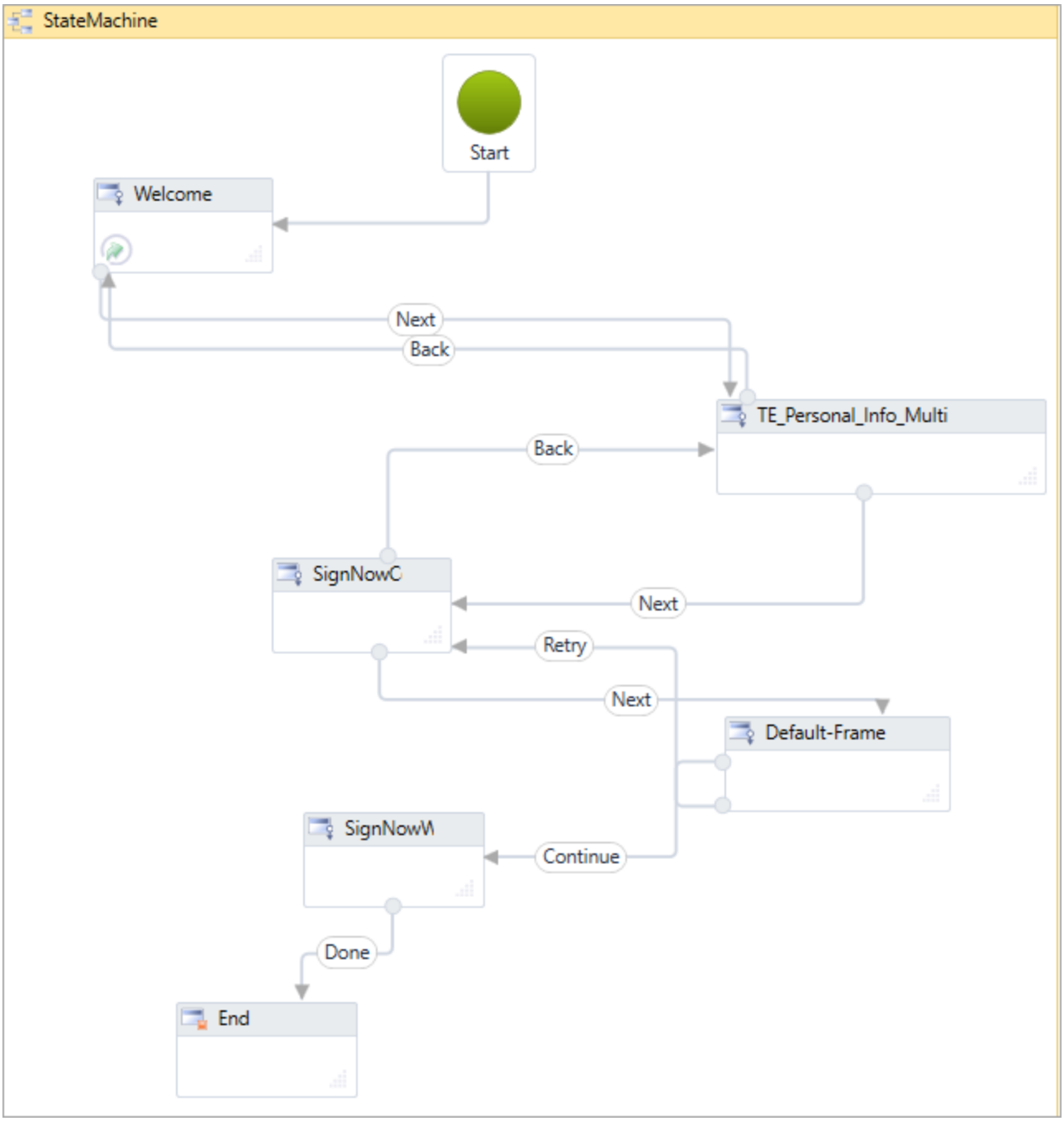
5. Disable **Block third-party cookies**.



Create the Workflow

Open the Workflow, Review Arguments, Variables, and Logging Requirements

1. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
2. In Workflow Composer, drag the state icons and transition lines to so that you can easily locate each item in the StateMachine workflow.



The workflow requires the following arguments and variables. You can create these arguments and variables before working on the activities, or you can add them when they are needed for a specific activity (as described below).

> ⚠ Keep in mind that arguments are passed in JSON format and that JSON elements are case sensitive. **Be sure to match the casing of argument names in Workflow Composer and Form Designer.**

Arguments

Name	Direction	Argument type	Default value
formInstance	In/Out	FormInstance	*Default value not supported*
entity	In/Out	VoidEntity	*Default value not supported*
event ⚠	In/Out	ConstructedEvent	*Default value not supported*
studentEntity	In/Out	StudentEntity	*Default value not supported*
frameUrl	In/Out	String	*Default value not supported*
signer2Name	In/Out	String	*Default value not supported*
signer2Email	In/Out	String	*Default value not supported*
signer2First	In/Out	String	*Default value not supported*
signer2Last	In/Out	String	*Default value not supported*

Variables

Name	Variable type	Scope	Default
renderedFormImage	String	StateMachine	*Enter a VB expression*
Pdf	Byte[]	StateMachine	*Enter a VB expression*
Doc	DocumentEntity	StateMachine	*Enter a VB expression*
studentId	Int32	StateMachine	*Enter a VB expression*
URL	String	StateMachine	*Enter a VB expression*
SignNowConfig	SignNowConfig	StateMachine	*Enter a VB expression*
SignNowDocument	SignNowDocument	StateMachine	new SignNowDocument
SignNowRecipient	SignNowRecipient	StateMachine	new SignNowRecipient
SignNowRecipients	SignNowRecipient[]	StateMachine	*Enter a VB expression*
SignNowRecipient2	SignNowRecipient	StateMachine	new SignNowRecipient
SignNowRequest	SignNowRequest	StateMachine	*Enter a VB expression*
SignNowSignedDocument	SignNowDocument	StateMachine	*Enter a VB expression*

LogLine Activities

While testing and troubleshooting the workflow, we recommend adding LogLine activities at critical stages within the workflow. The following expression will provide logging for specific objects (replace <object> with the object name):

Newtonsoft.Json.JsonConvert.SerializeObject(**<object>**, Newtonsoft.Json.Formatting.Indented)

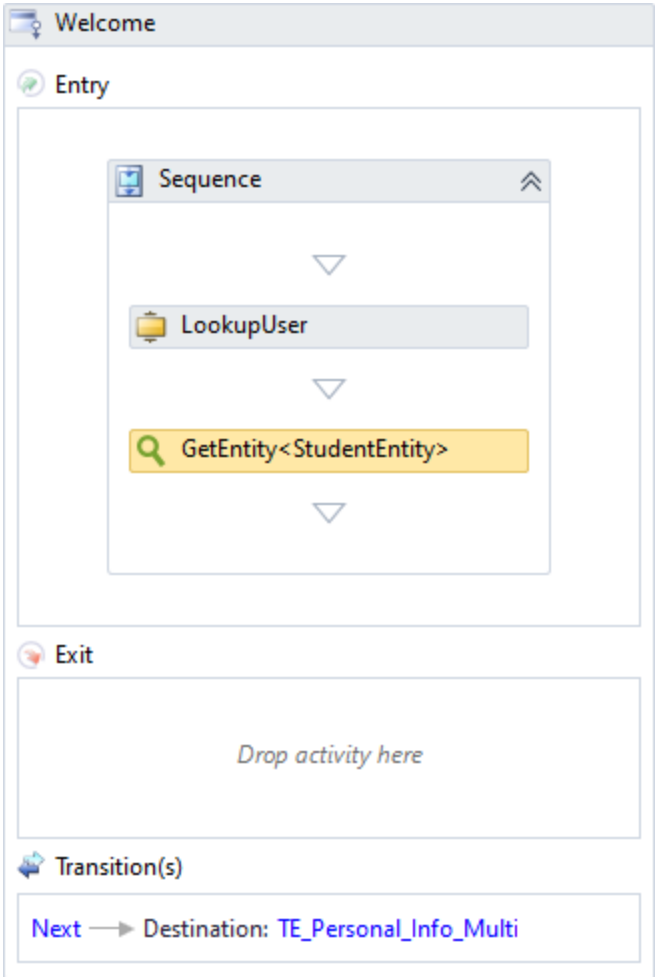
We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

LookupUser and GetEntity

If your sequence is non-authenticated (Anonymous=true), the LookupUser and GetEntity activities are not applicable, but a CreateEntity activity may be needed instead. Be sure that any anonymous sequence includes the user's email address. It is required as the SignNow recipient address.

Use the *Welcome* form to retrieve the Student Id and Student Entity.

1. Double-click the icon of the **Welcome** state.
2. Create a variable named **studentid** of Type **Int32**.
3. Drop a **LookupUser** activity into the Entry area of the Welcome state.
    - In the **UserId** property, specify the **studentid** variable.
    - In the **UserName** property, specify **formInstance.UserName**.
4. Drop a **GetEntity** activity below the LookupUser activity.
    - In the **EntityId** property, specify the **studentid** variable.
    - In the **Result** property, specify **studentEntity**.

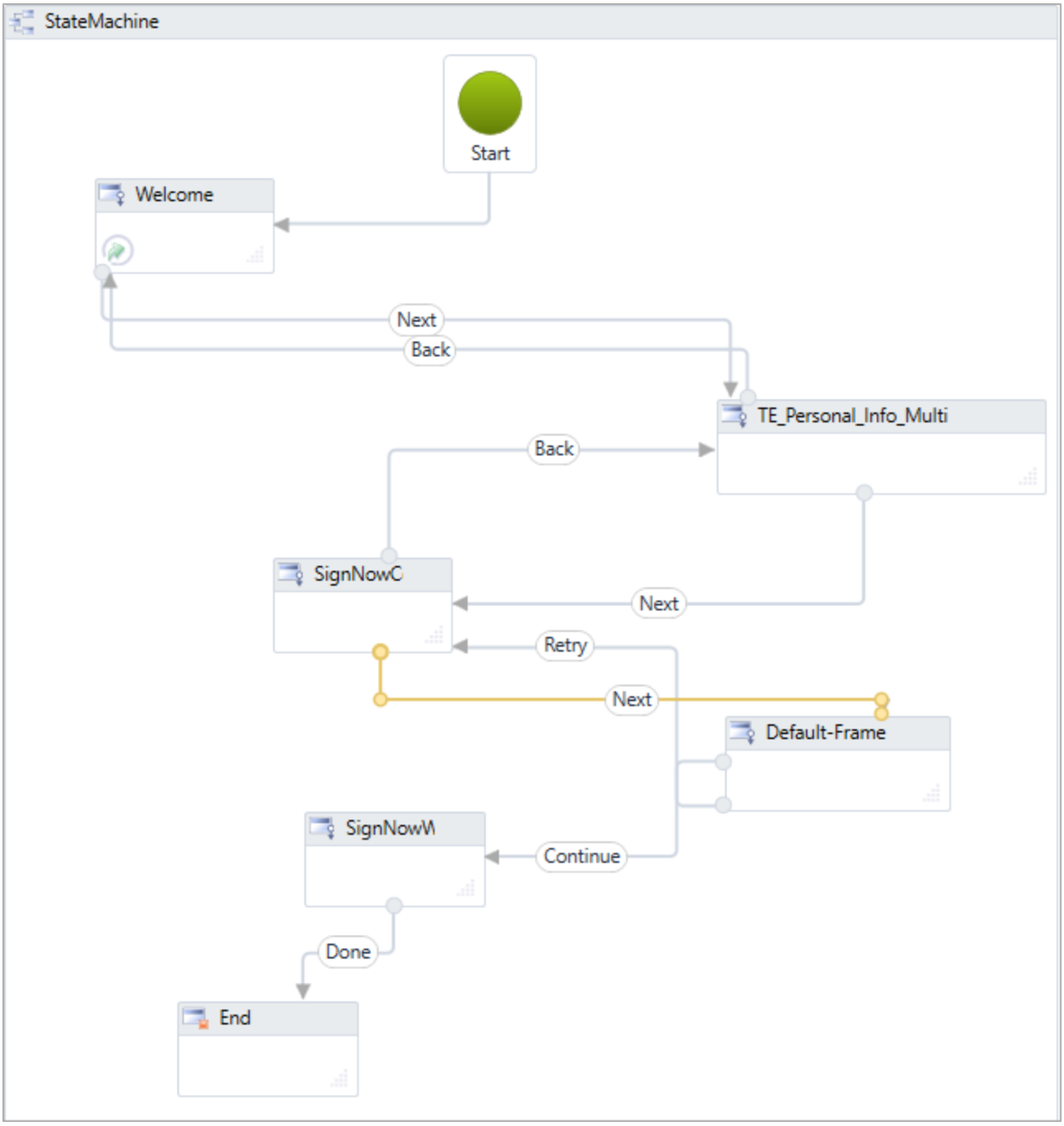


5. **Save** the workflow locally and continue with the next set of steps.

Persist the Workflow Instance After Collecting the Form Data

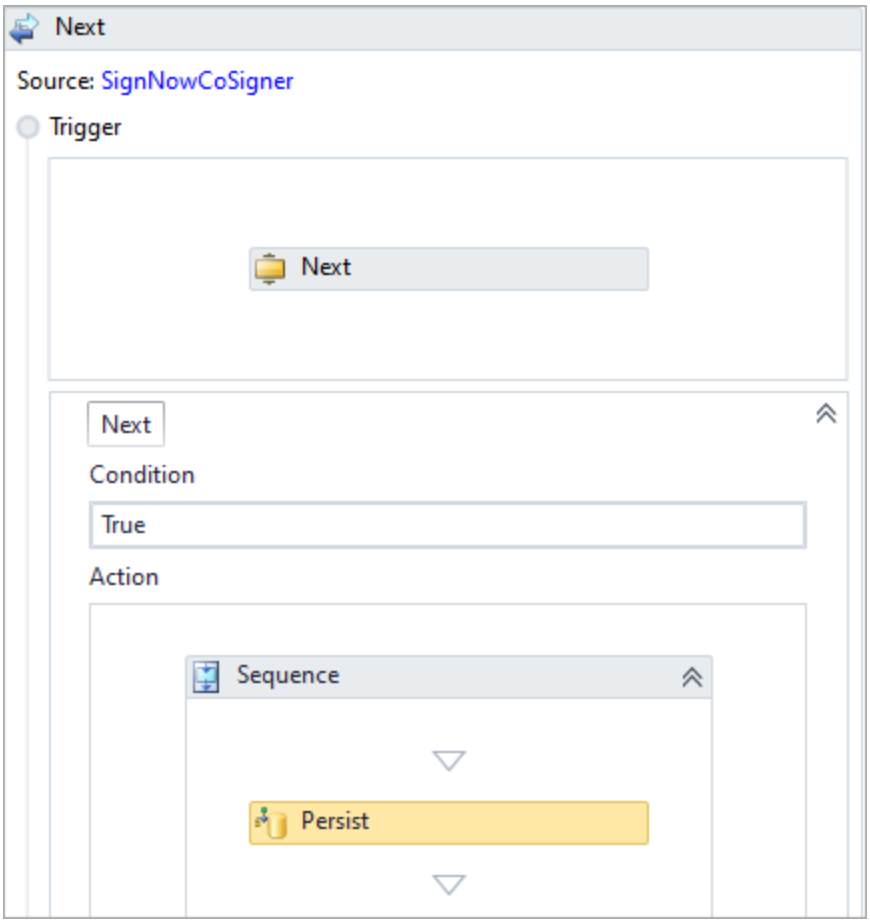
Since the workflow can be busy for a long time, we recommend that you persist the workflow instance before creating the PDF.

1. Double-click the **Next** transition between the *SignNowCoSigner* and *Default-Frame* states.



2. Set the Condition to **True**.
3. Drag a **Persist** activity into the Trigger area below the Next activity. The Persist activity does not require any properties to be specified.

   Workflow Composer automatically wraps the Next and Persist activities in a Sequence.



4. **Save** the workflow locally and continue with the next set of steps.

Create a PDF of the SignNow Form

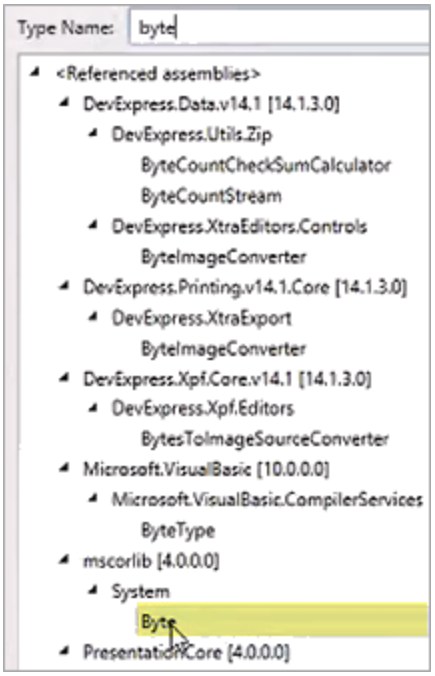
1. Create a variable named **URL**. Set the Variable type to **String**. Set the Scope to **StateMachine**. This variable will be assigned to the PDF created from the form.
2. Create a variable with a name like **Pdf**. Set the Scope to **StateMachine**. This variable will hold the document image created from the form.

   Make sure you use the same variable name when you reference this variable later in the workflow. (This applies to any other variable.)

   a. In the Variable type field, select **Array of [T]** and select **Browse for Type**.

   b. In the "Browse and Select a .Net Type" window, specify **byte**, select the System variable **Byte**, and

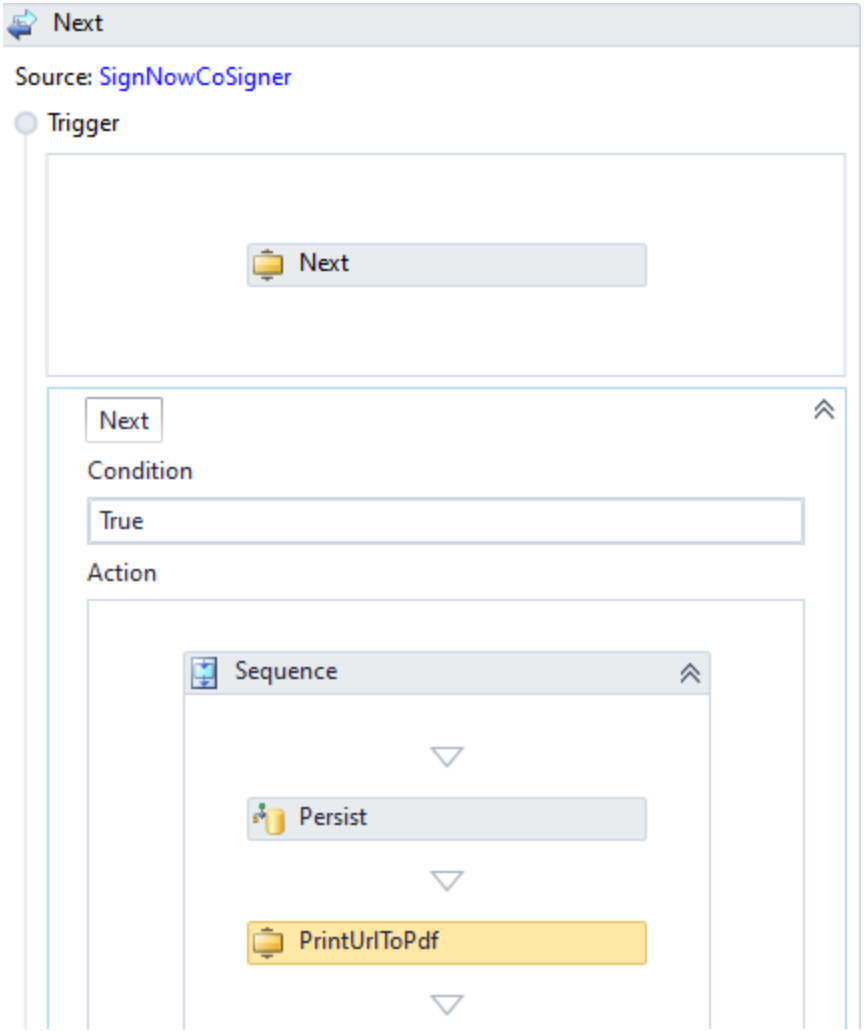
select **OK**.



3. Drop a **PrintUrlToPdf** activity below the Persist activity.

   Specify the properties for the activity as follows:

   - PdfDocument = **Pdf** (This is the name of the variable created above.)
   - Url = **URL** (This is the name of the variable created above.)
   - Validation Messages = **formInstance.ValidationMessages**



4. **Save** the workflow locally and continue with the next set of steps.

Get the SignNow Configuration and Pass the Recipient Information

In the following steps we will continue to add activities to the Next transition between the *SignNowCoSigner* and *Default-Frame* states.

1. Create a variable named **SignNowConfig**. Set the Scope to **StateMachine**.

   In the Variable type field, select **Browse for Type**. In the "Browse and Select a .Net Type" window, scroll down to Cmc.Nexus.FormsBuilder.Contracts > Cmc.Nexus.FormsBuilder.Entities and select **SignNowConfig**.

2. Drop a **GetSignNowConfig** activity below the PrintUrlToPdf activity.

   Specify the properties for the activity as follows:

- SignNowConfig = **SignNowConfig**
- Validation Messages = **formInstance.ValidationMessages**

The GetSignNowConfig activity retrieves your user credentials from the SignNow settings in Forms Builder. These values enable the workflow to log in to SignNow.

3. Create the following variables:

Variable Name	Type	Scope	Default
SignNowRecipient	Cmc.Nex-us.FormsBuilder.Entities.SignNowRecipient	StateMachine	new SignNowRe-cipient
SignNowRe-cipient2	Cmc.Nex-us.FormsBuilder.Entities.SignNowRecipient	StateMachine	new SignNowRe-cipient
SignNowDocument	Cmc.Nex-us.FormsBuilder.Entities.SignNowDocument	StateMachine	new SignNowDocu-ment
SignNowRequest	Cmc.Nex-us.FormsBuilder.Entities.SignNowRequest	StateMachine	N/A

4. Below the GetSignNowConfig activity, drop an **Assign** activity for each row in the following table and type the indicated values:

"To" Field	Value	Notes
SignNowConfig.EmailSubject	"CMC SignNow Test at " + DateTime.Now.ToString()	This is the email that the end user will receive after the signing pro-cess is done.
SignNowDocument.DocumentId	"1"	The SignerId should match the Signer property in the SignNow component on the form. Allowed values are "1" to "5".
SignNowDocument.Name	"NewMultiSignNow.pdf"	Name of the signed document.
SignNowDocument.Content	Pdf	Name of the variable that holds the document image (refer to Create a PDF of the Form). In our example the name of the variable is **Pdf**.
SignNowRecipient.FirstName	studentEntity.FirstName	First name of the user who sub-mitted the signed document.
SignNowRecipient.LastName	studentEntity.LastName	Last name of the user who sub-mitted the signed document.
SignNowRecipient.Email	SignNowConfig.SignerAccountEmail	Email address of the person who will receive the signed SignNow document.

"To" Field	Value	Notes
SignNowRecipient2.FirstName	signer2First	First name of the co-signer who submitted the signed document. This could be a parent name from the StudentRelationshipAddressEntity.
SignNowRecipient2.LastName	signer2Last	Last name of the co-signer who submitted the signed document. This could be a parent name from the StudentRelationshipAddressEntity.
SignNowRecipient2.Email	signer2Email	Email address of the co-signer who will receive the signed SignNow document. This could be the email address retrieved from the StudentRelationshipAddressEntity.
SignNowRecipient2.SignerId	"2"	The SignerId should match the Signer property in the SignNow component on the form. Allowed values are "1" to "5".

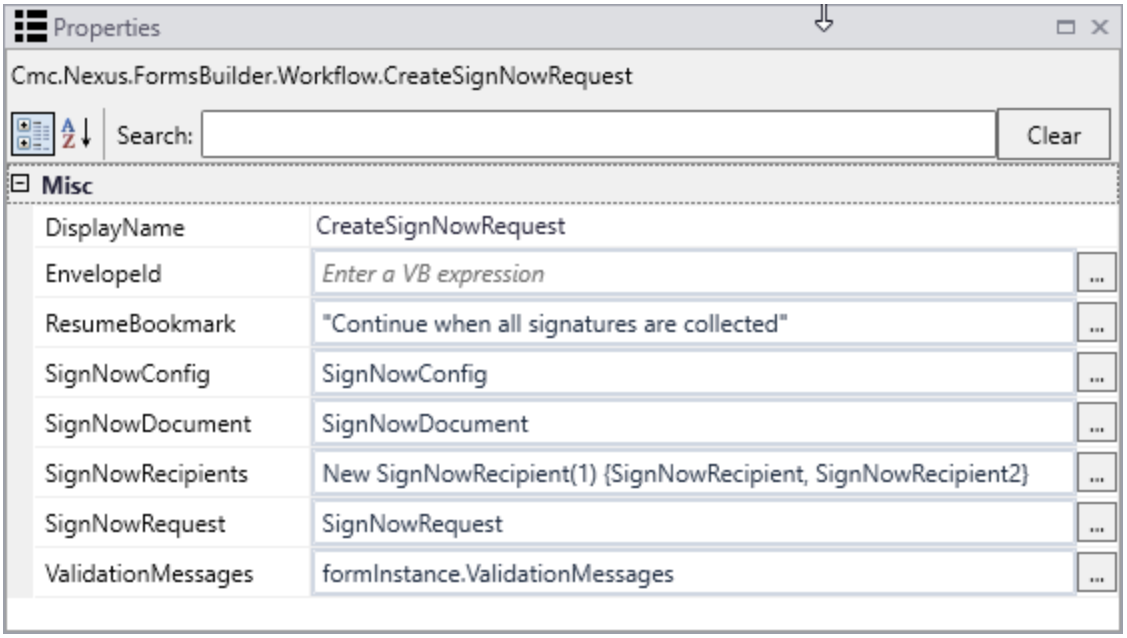
5. **Save** the workflow locally and continue with the next set of steps.

Create the SignNow Request and Specify the IFrame URL

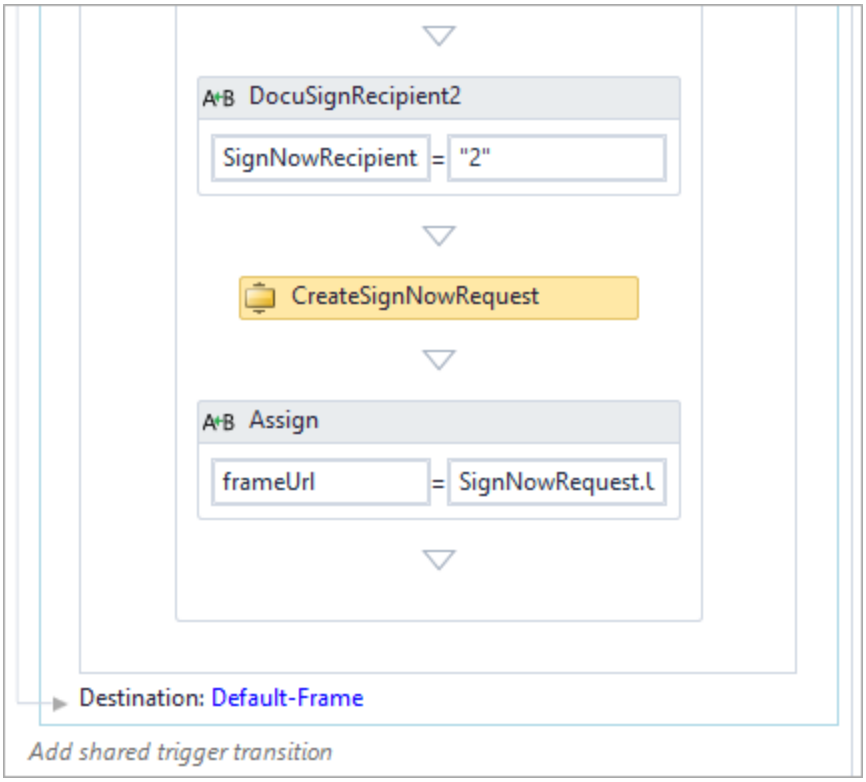
In the following steps we will continue to add activities to the Next transition between the *SignNowCoSigner* and *Default-Frame* states.

1. Drop the **CreateSignNowRequest** activity below the previous activity.

   Specify the properties using the names of the variables created above as follows:



- The expression for SignNowRecipients passes multiple items (the primary and secondary recipient) within an array.
- The ResumeBookmark value will be passed to SignNow as a custom field. When the signature request has been completed, SignNow will return this flag and any process that has to happen after the bookmark can resume automatically without any user intervention.
- The out argument SignNowRequest returns the URL of the signed SignNow document.



2. Drop an **Assign** activity below the CreateSignNowRequest activity.
    a. In the *To* field, specify **frameUrl**. (This is the argument associated with the IFrame form.)

    ⚠ Be sure to use the exact casing shown here.

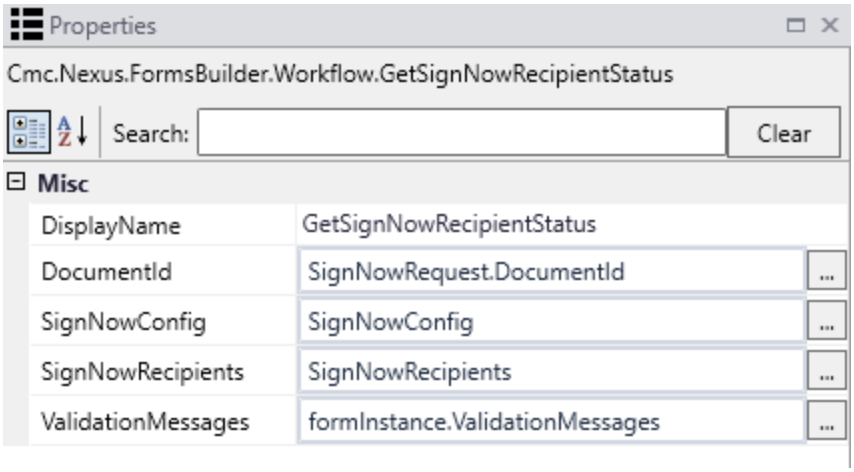
    b. In the *Value* field, specify **SignNowRequest.Url**.
3. **Save** the workflow locally and continue with the next set of steps.

Check Signature Status and Continue or Retry

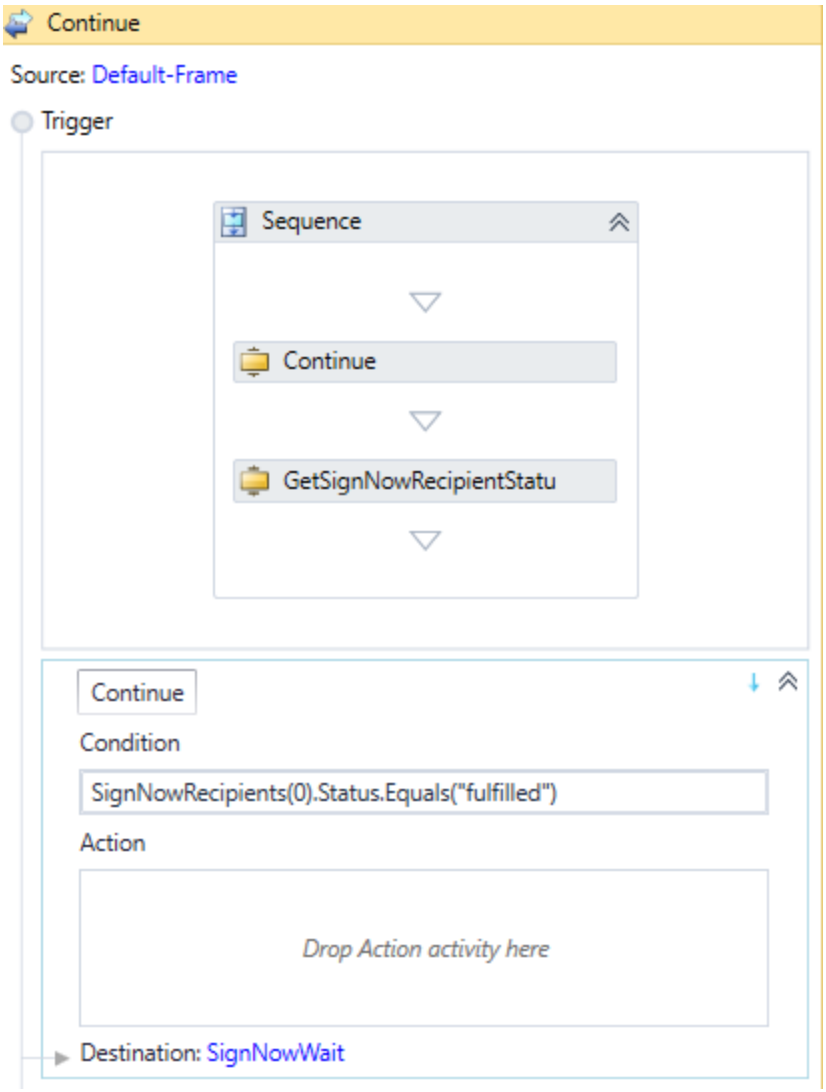
The IFrame form (*Default-Frame*) will receive the SignNow document with the first signature, but the signing process is not yet complete. We want the workflow to continue with the *SignNowWait* form upon receipt of the first signature. The workflow will pause on the *SignNowWait* form until the second signature is received. If user closes the form or logs out after submitting the first signature, upon return to the form, the workflow will display the *SignNowWait* form.

1. Double-click the transition from the *Default-Frame* form to the *SignNowWait* form.
2. In the Trigger area, name the WaitForFormBookmark as **Continue**.
3. Select the property **Transition Type = MoveForward** on the WaitForFormBookmark activity.
4. Drop a **GetSignNowRecipientStatus** activity below the Continue WaitForFormBookmark.

    Specify the properties using the names of the variables created above as follows:

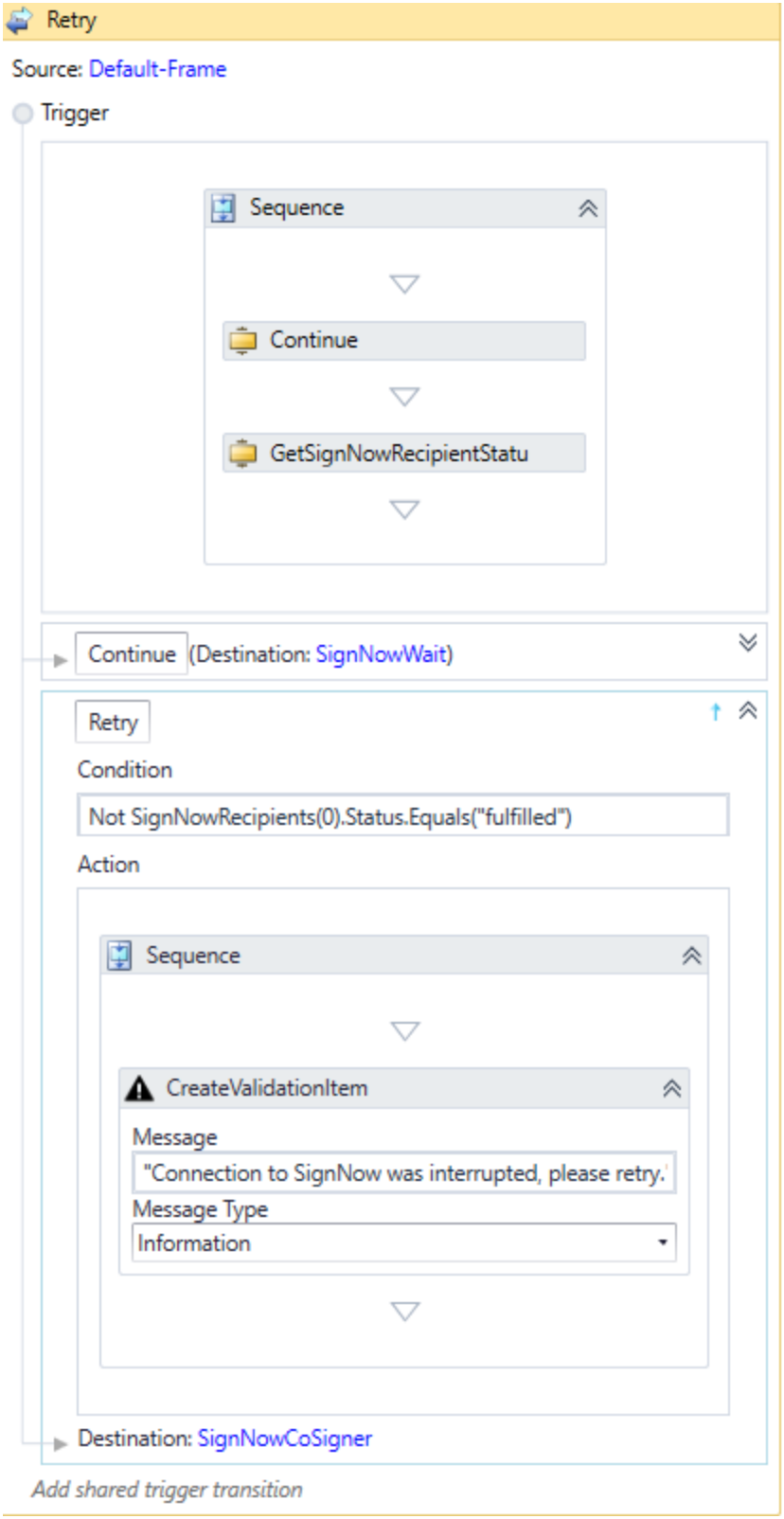


5. Specify the Condition as **SignNowRecipients(0).Status.Equals("fulfilled")**.

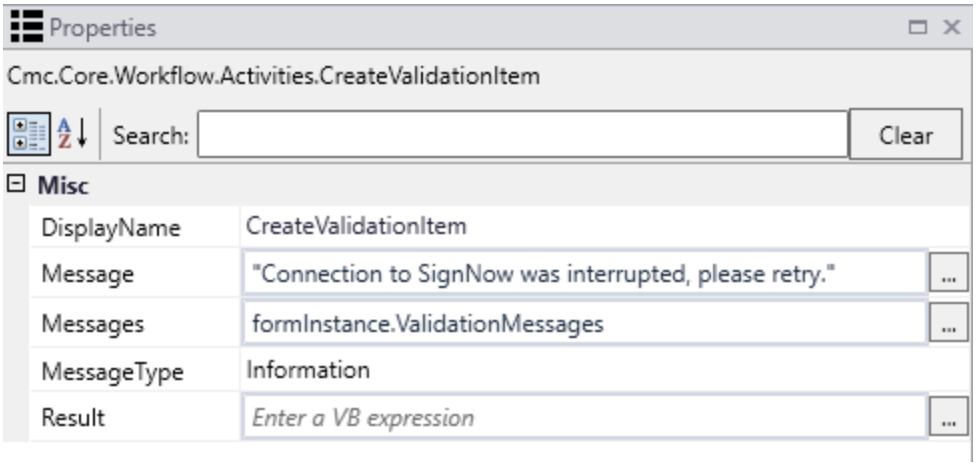


6. Select the *Default-Frame* state and drag a line from the *Default-Frame* state to the *SignNowCoSigner* state to create a new transition (T1).
7. Change the T1 label to **Retry** and add the following condition:

   Not SignNowRecipients(0).Status.Equals("fulfilled")



8. Drag a **CreateValidationItem** activity into the Action area of the Retry transition and specify the following properties:

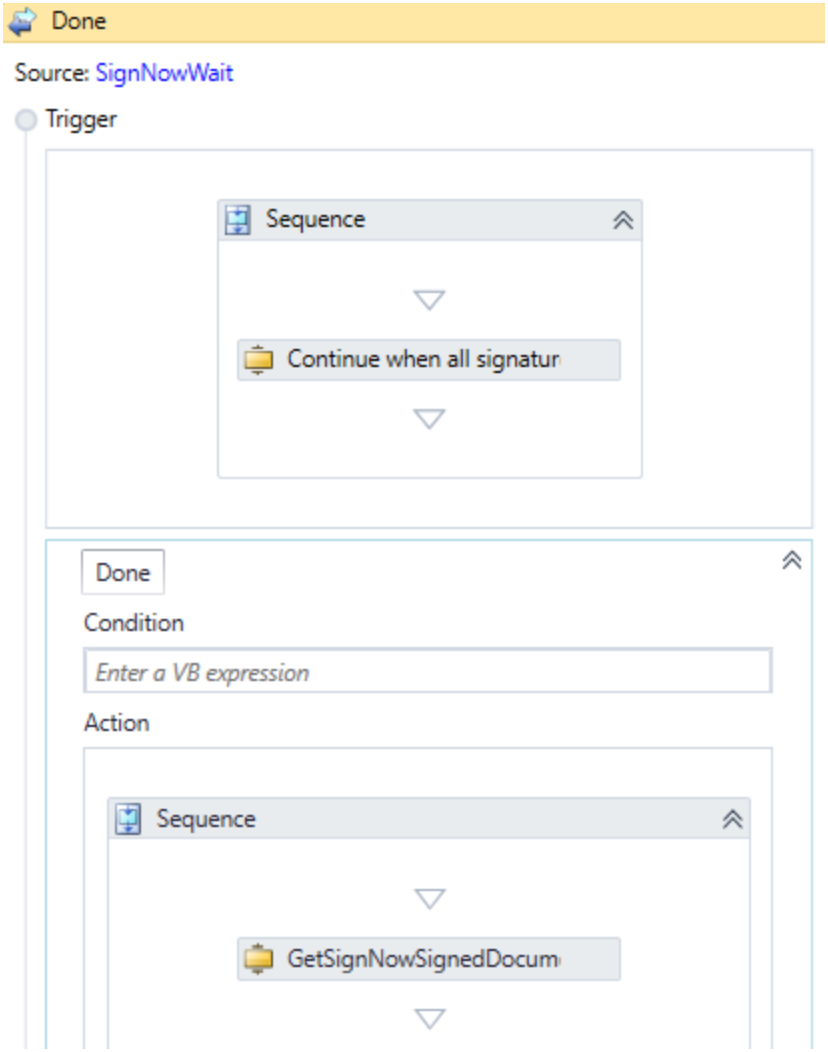


9. **Save** the workflow locally and continue with the next set of steps.

Receive the Signed SignNow Document

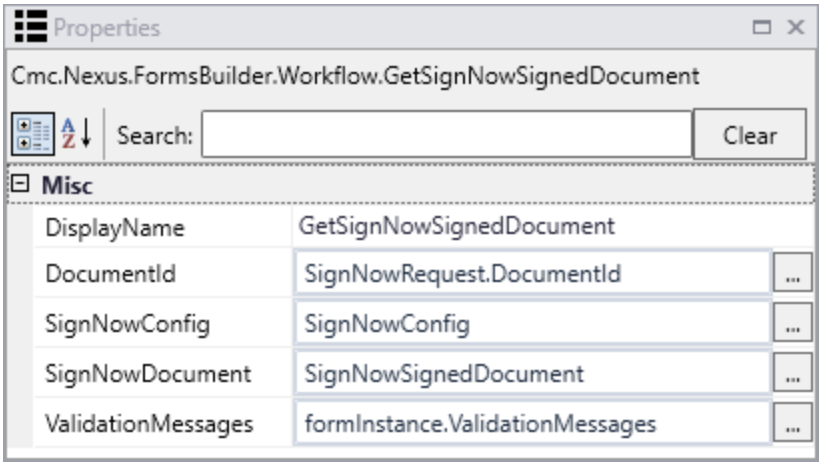
In the following steps, we will create activities in the Done transition before the End state. At this point in the workflow, the second signature was received and the SignNow document is complete.

1. Double-click the **Done** transition before the *End* state and rename the WaitForFormBookmark activity as **Continue when all signatures are collected**. SignNow returns the string "*Continue when all signatures are collected" in the custom field named "Bookmark*".



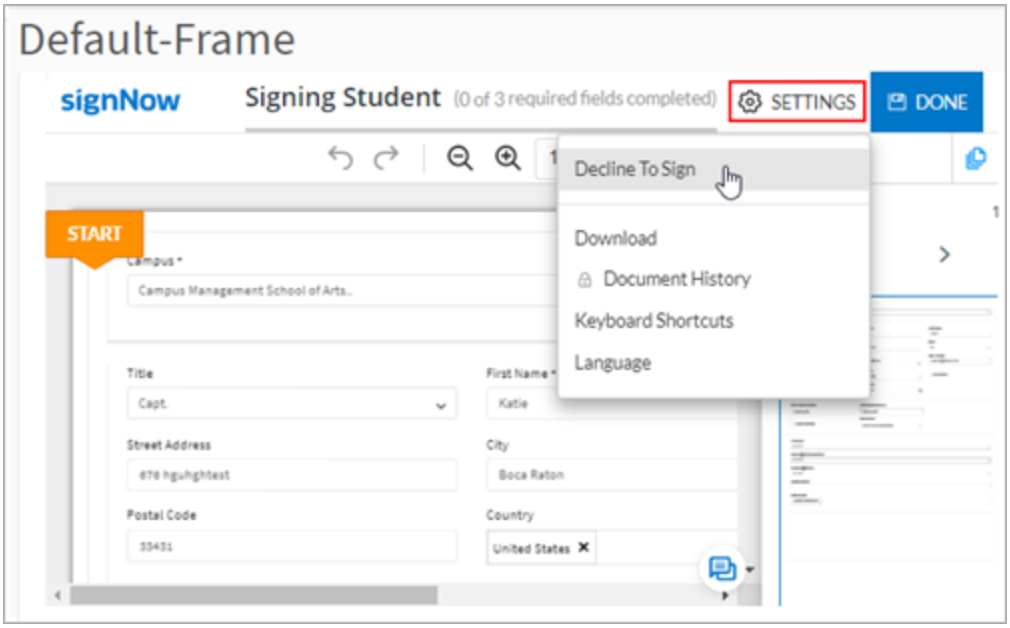
2. Create a variable named **SignNowSignedDocument**. In the Variable type field, select **SignNowDocument**. Set the Scope to **StateMachine**.
3. Drop a **GetSignNowSignedDocument** activity into the Action area.

   Specify the properties for the activity as follows:



4. **Save** the workflow locally and continue with the next set of steps.

SignNow provides signers the **Decline To Sign** option.



If the decline option is selected by the primary, secondary, or any additional signer:

- The signer will be presented with a popup stating "*You've declined to sign <PDF name>. <SignNow account email address> has been notified of your decision. You can always sign the document later if you change your mind*."
- SignNow will not return control of the sequence to Forms Builder and the workflow will not continue. The workflow will remain as a persisted instance. You will need to terminate persisted workflows periodically using either Sequence Designer (refer to Delete Persisted Workflow Instances) or Workflow Composer (refer to Persisted Workflows).

Create and Save the Document in Anthology Student

To convert a SignNow document to a DocumentEntity that can be attached to a record in Anthology Student, add CreateDocument and SaveDocument activities to the workflow. These activities will be placed below the GetSignNowSignedDocument activity.

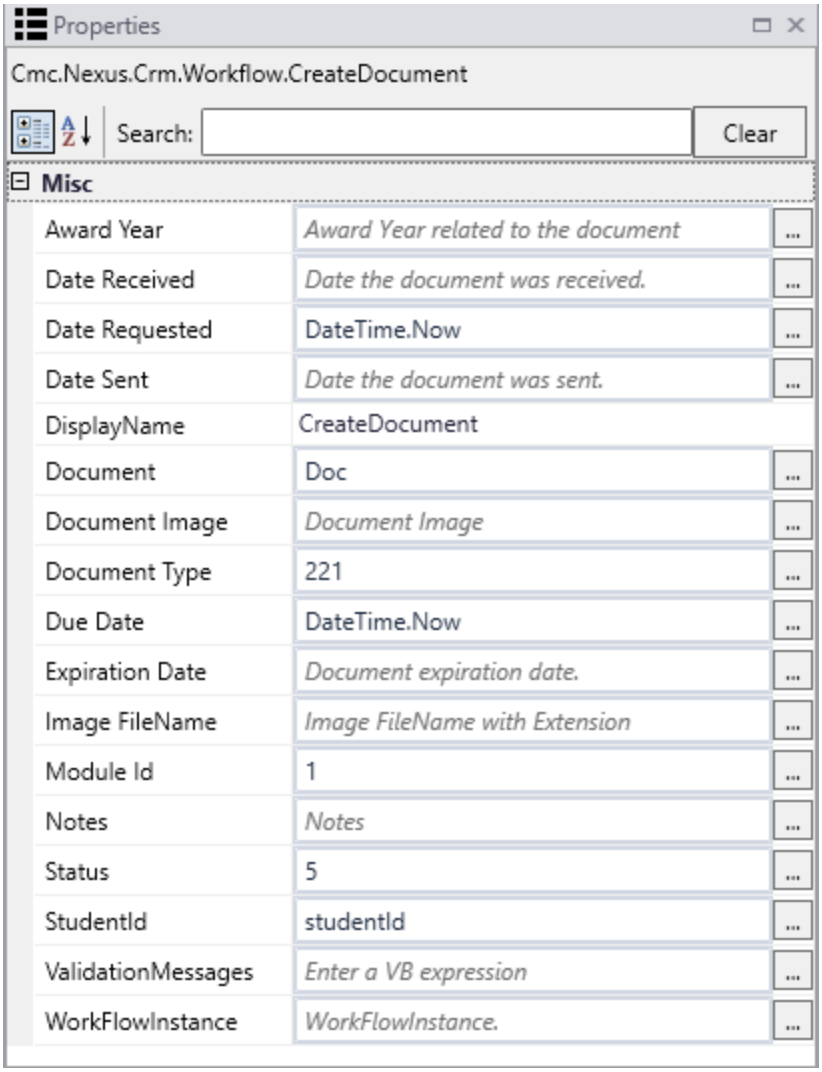
1. Create a variable named **Doc**. Set the Scope to **StateMachine**.

   In the Variable type field, select **Browse for Type**. In the "Browse and Select a .Net Type" window, scroll down to Cmc.Nexus.Crm.Contracts > Cmc.Nexus.Crm.Entities and select **DocumentEntity**.

2. Drop a **CreateDocument** activity below the GetSignNowSignedDocument activity.

   Specify the properties as shown below activity and use the drop-down field in the activity to select appropriate values:

   - Module = Select a Module from the drop-down list, e.g., Admissions.
   - Document Type = Select a Type (Template) from the drop-down list.
   - Document Status = Select a Type from the drop-down list.
   - Student = Specify a Student Id or use a variable.
   - Due Date = Specify a date or use a variable, e.g., DateTime.Now
   - Document (OutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
   - Validation Messages = formInstance.ValidationMessages



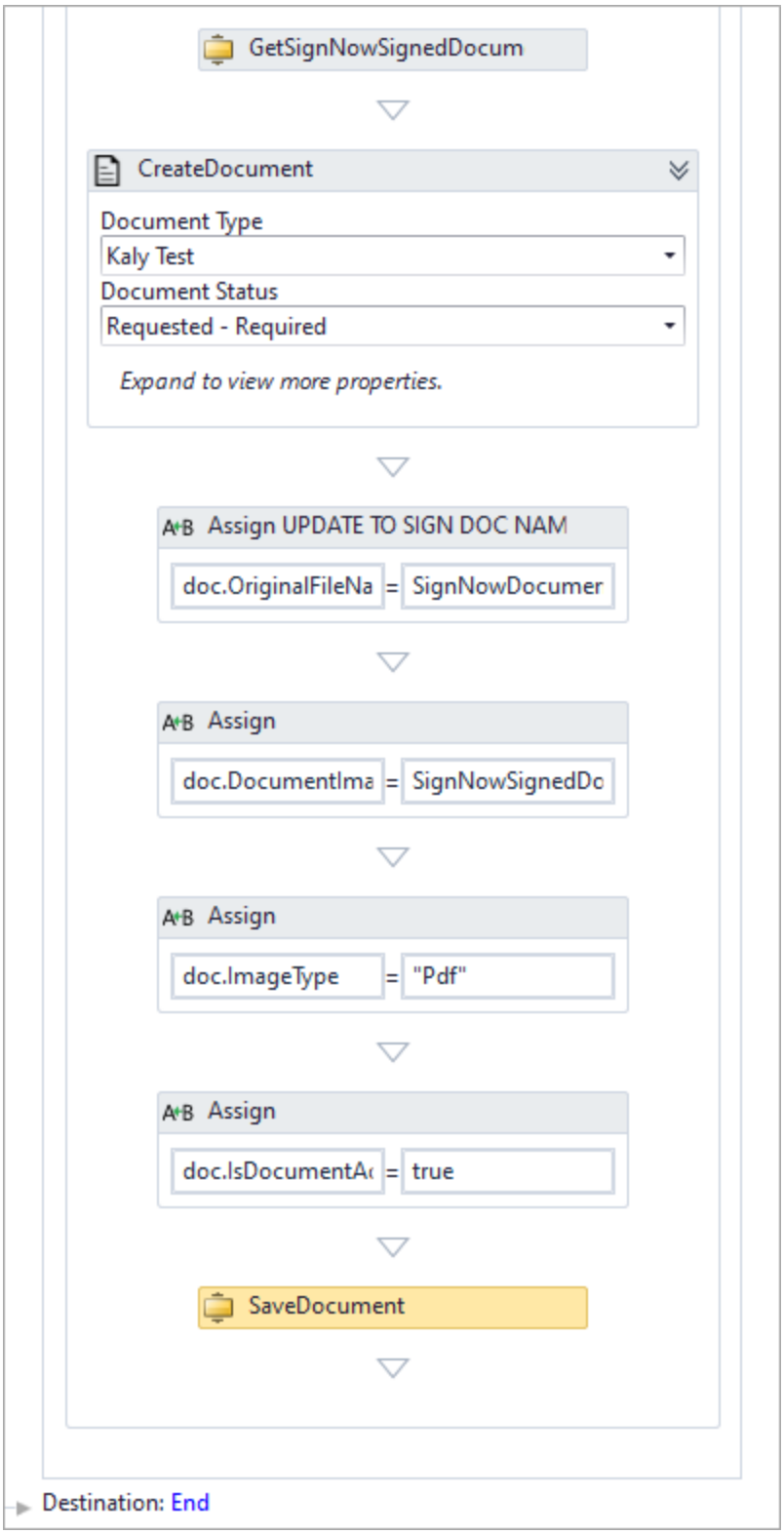
3. Below the CreateDocument activity, drop an **Assign** activity for each row in the following table and type the indicated values:

"To" Field	Value
doc.OriginalFileName	SignNowDocument.Name
doc.DocumentImage	SignNowSignedDocument.Content
doc.ImageType	"Pdf"
doc.IsDocumentAddedManually	true

4. Drop a **SaveDocument** activity below the last Assign activity.

   Specify the properties for the SaveDocument activity as follows:

   - Document (InOutArgument) = **Doc** (This is the variable created above for the DocumentEntity.)
   - Validation Messages = **formInstance.ValidationMessages**



5. **Save** the workflow locally and continue with the next set of steps.

Final Steps

1. Select **Publish**. The New Workflow Definition Version window is displayed.
2. If you want the workflow to be run as soon as the event occurs on the entity, select **Enable This Workflow Version?**, otherwise leave the check box cleared.
3. Select **Save**, then **Cancel** to close the publisher window.
4. In the **Sequence List**, select and complete the form that contains the SignNow fields.
5. Verify that the signed SignNow document is available at the return URL.

While testing your workflow, make sure that you terminate the previous instance of the workflow before running an updated version of the same workflow. In Workflow Composer, select **Open Persisted Workflow**, select your workflow instance, and select **Terminate**.

Persisted workflow instances can be also deleted from the Sequence Designer workspace. For more information, refer to Delete Persisted Workflow Instances.

FERPA Form

In this exercise, we will create a form for the student to complete for the Release of Information (FERPA). This is a simplified form that will insert the person information in the Address folder as type FERPA so the student's FERPA Directory Information can be further updated allowing the person access.

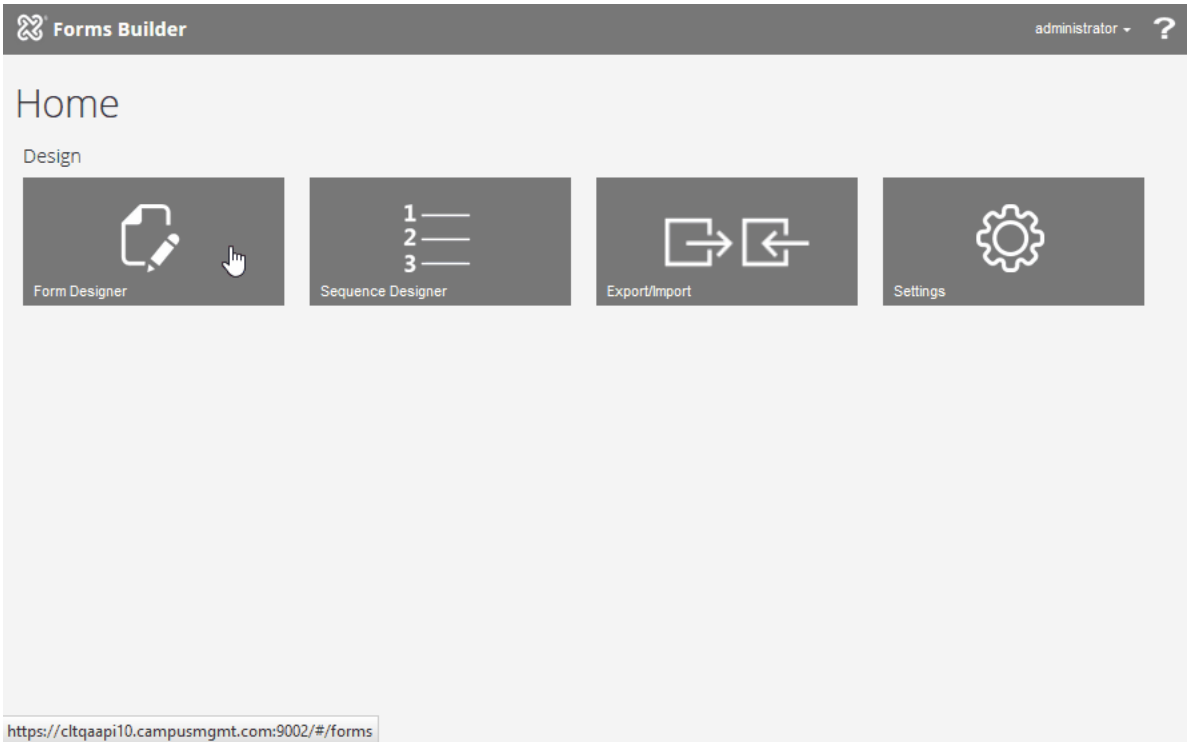
> ⚠ The screen captures in this topic show an earlier Forms Builder version. While the UI has been updated, the basic functionality is unchanged.

Build the Form

1. Open your browser and point to the Forms Builder URL.
2. Sign in using your user name and password.
3. On the Forms Builder home page, select the Form Designer tile. Products (if applicable), Entities and Forms are loaded into Forms Builder.
4. In the **Select Provider** drop-down list, select **Student**.

   This step is needed only if your Forms Builder installation uses the databases of both CampusNexus CRM and Anthology Student.
5. Set up the column layout for the form:
   a. In the Column drop-down list, select **2 Columns** and select **New**. A 2-column panel is added to the Layout pane.
   b. In the Column drop-down list, select **1 Column** and select  to add a new panel to the form layout.
   c. In the Column drop-down list, select **3 Columns** and select  to add a new panel to the form layout.
   d. In the Column drop-down list, select **2 Columns** and select  to add a new panel to the form layout.

   You should have a total of 4 panels in the Layout pane.



6. From the Components tab, drag a **Text Box** into the first column of the first panel. Specify the following properties:

    a. Label: **Full Name**

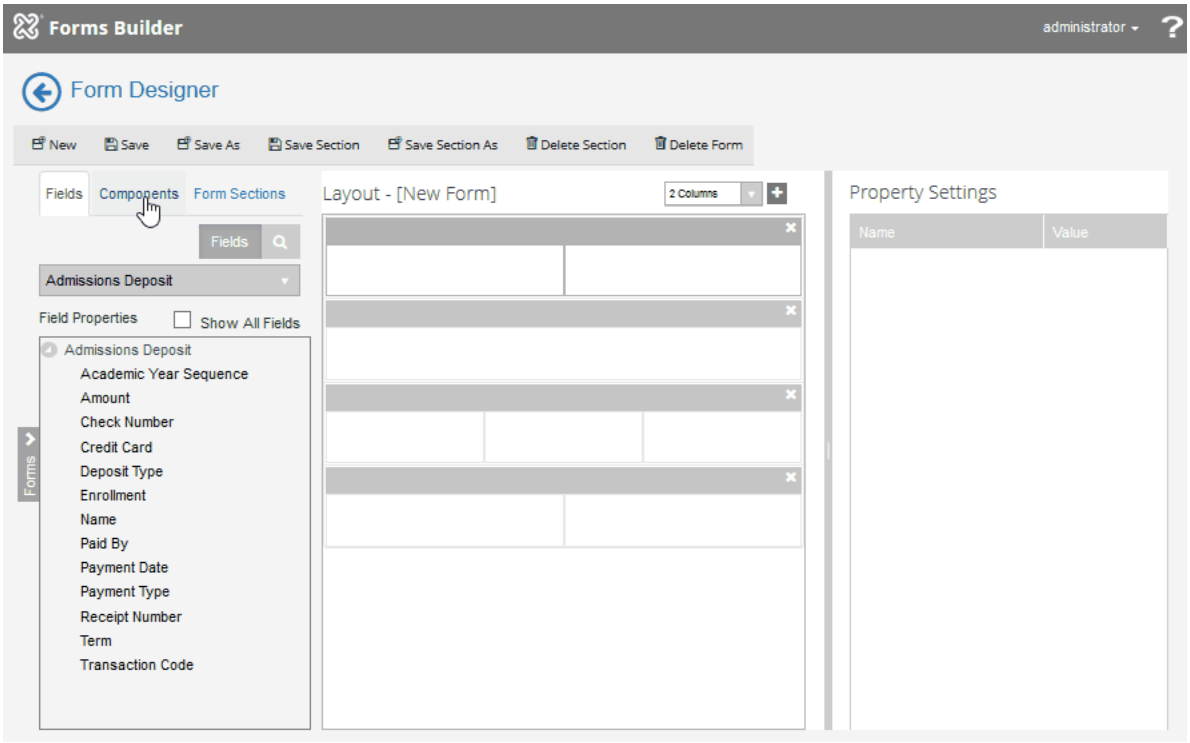
    b. Model: **vm.models.FullName**

    c. Read-Only: **true**

    We are going to bind this field to the studentEntity in workflow using an In/Out argument and populate the text box with the First Name and Last Name on the studentEntity.

7. On the Fields tab, select the select the **Student** entity.

8. Drag the **Student Number** into the second column of the first panel. The Read-Only property should be set to **true**.



9. From the Components tab, drag two **HTML** components into the second panel. Specify the following properties:

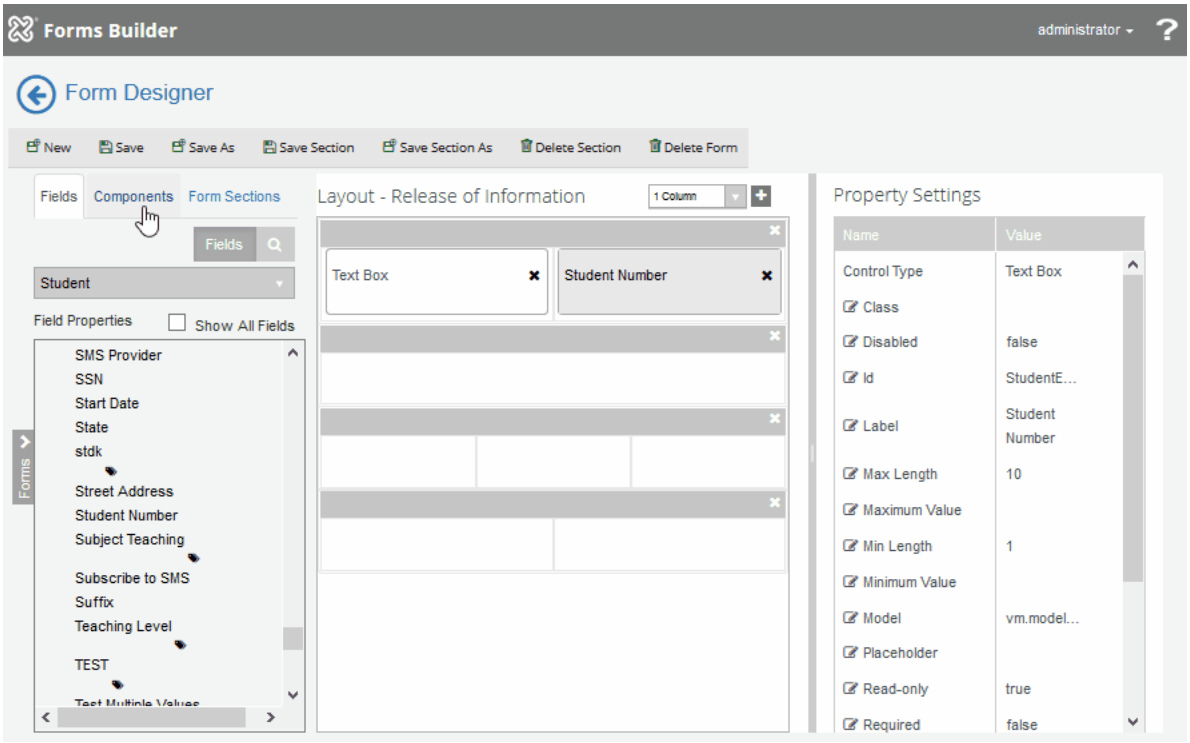
    a. First HTML component, HTML property:

    In compliance with the Department of Education's "Family Educational Rights and Privacy Act" (FERPA), information in your student record may not be released to a third-party (parents, guardians, spouse, sponsor, etc.) without your written permission except as provided by law (Refer to EC 76243, EC 76244).

    b. Second HTML component, HTML property:

    I grant permission to Campus University to release information about my educational record to the individuals listed below. This permission will remain in effect until revoked in writing. This permission does NOT cover financial records maintained in the Financial Aid Department.

10. On the Fields tab, select the select the **Student Relationship Address** entity.
11. Drag the **First Name**, **Last Name**, and **Relation to Student** fields into the third panel.
12. Change the Required property for all three fields to **true**.

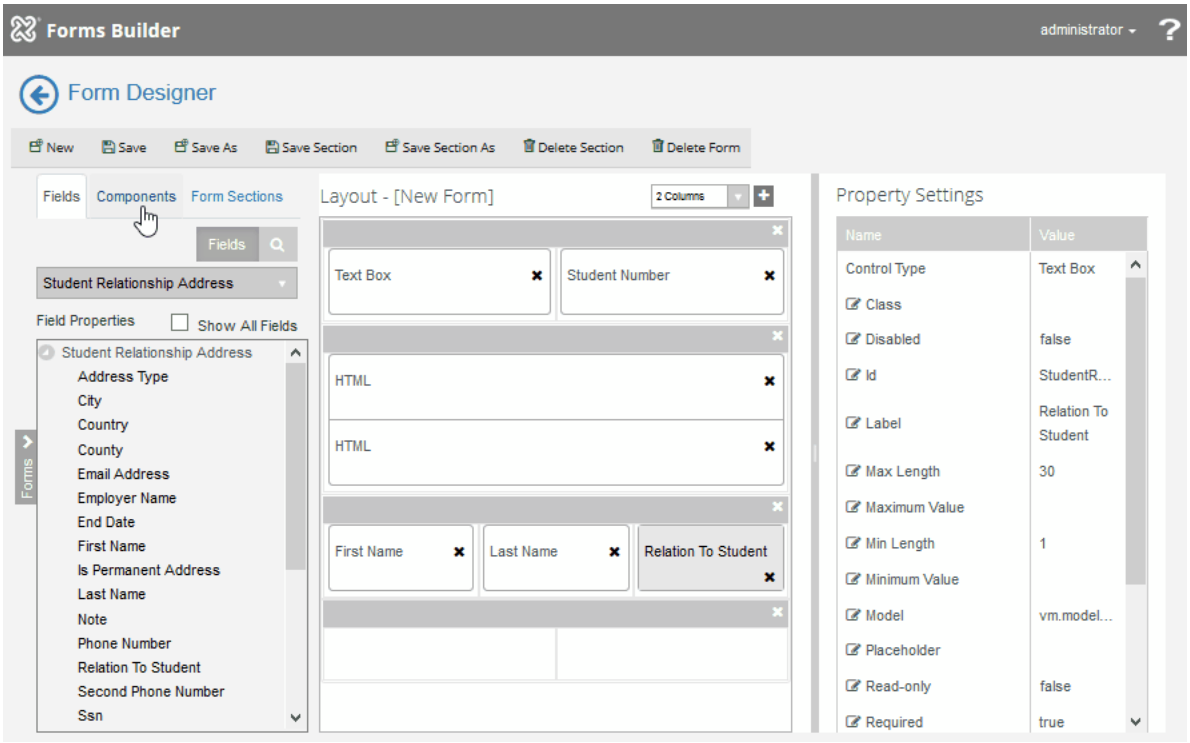


13. From the Components tab, drag a **Checkbox** into the first column of the fourth panel. Specify the following properties:

    a. Label: **I authorize the Release of Information to the person listed above.**

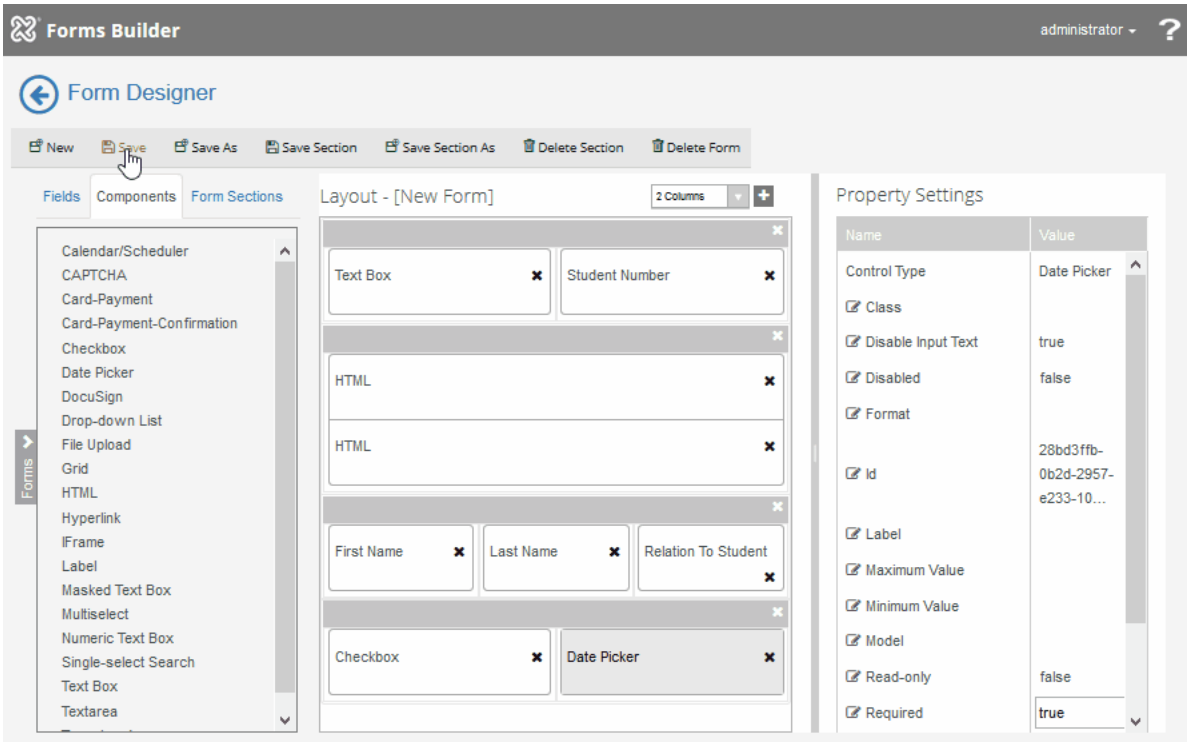
14. Drag the **Date Picker** component into the second column of the fourth panel. Specify the following properties:

    a. Required: **true**



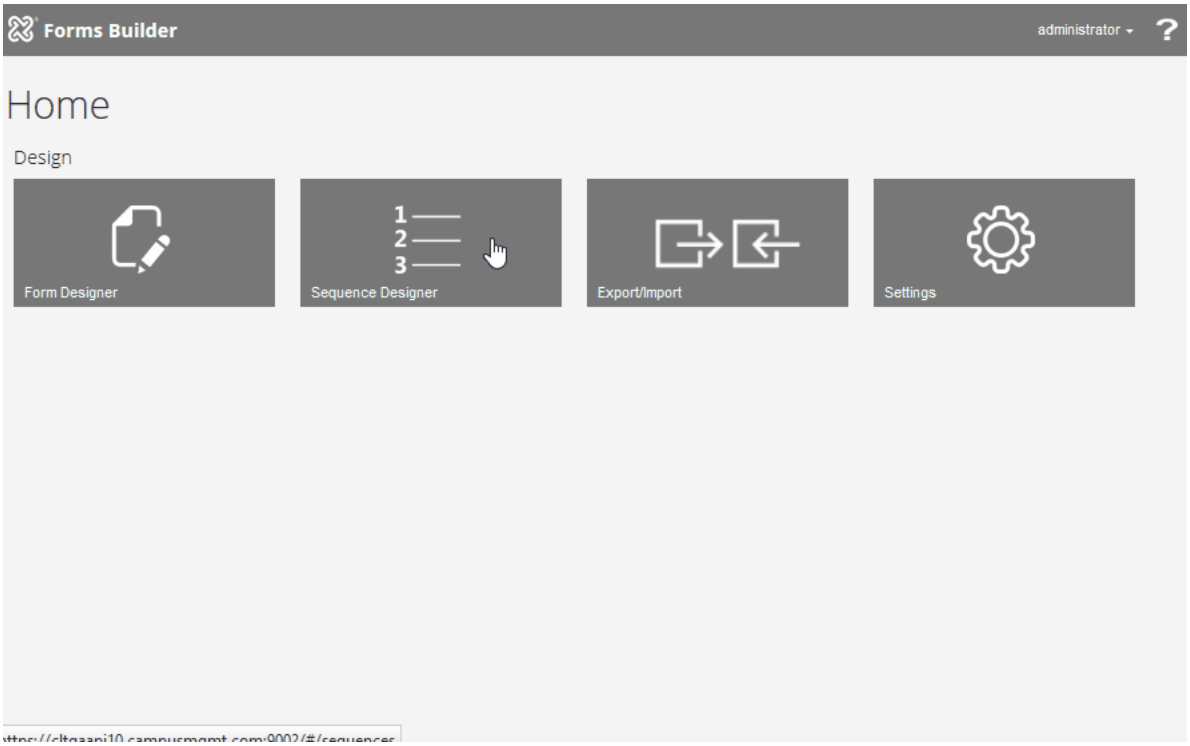
15. Select **Save** to save what you have done so far on your form. Specify the following properties:
    a. Form Name: **Release of Information**
    b. Title: **Release of Information**

    The title is optional. When a title is specified, it is displayed on the rendered form.
16. Select  to return to the Forms Builder home page.



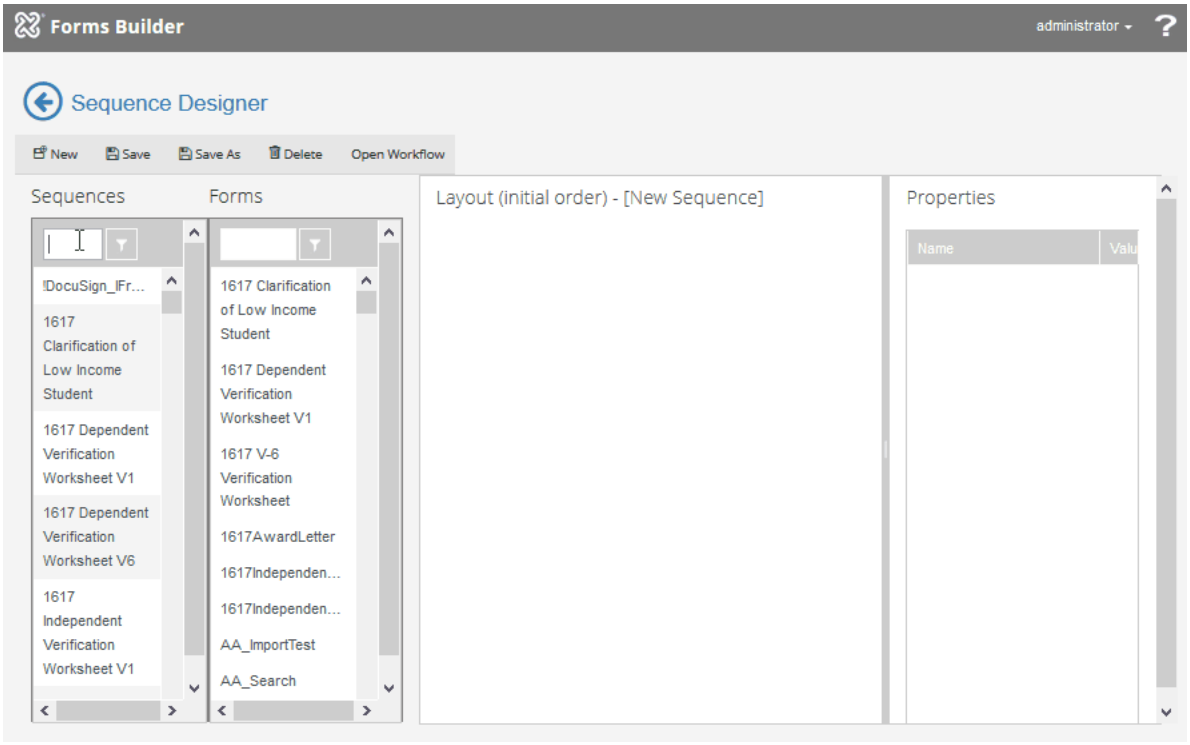
Create a Sequence

1. On the home page of Forms Builder, select the **Sequence Designer** tile.
2. Select **New**. A new panel is added to the Layout pane.
3. In the search bar of the Forms pane, search for the **Release of Information** form and drag it into the Layout pane.
4. **Save** the sequence. Specify the following properties:
    a. Sequence Name: **Release of Information**
    b. Title: **Release of Information**

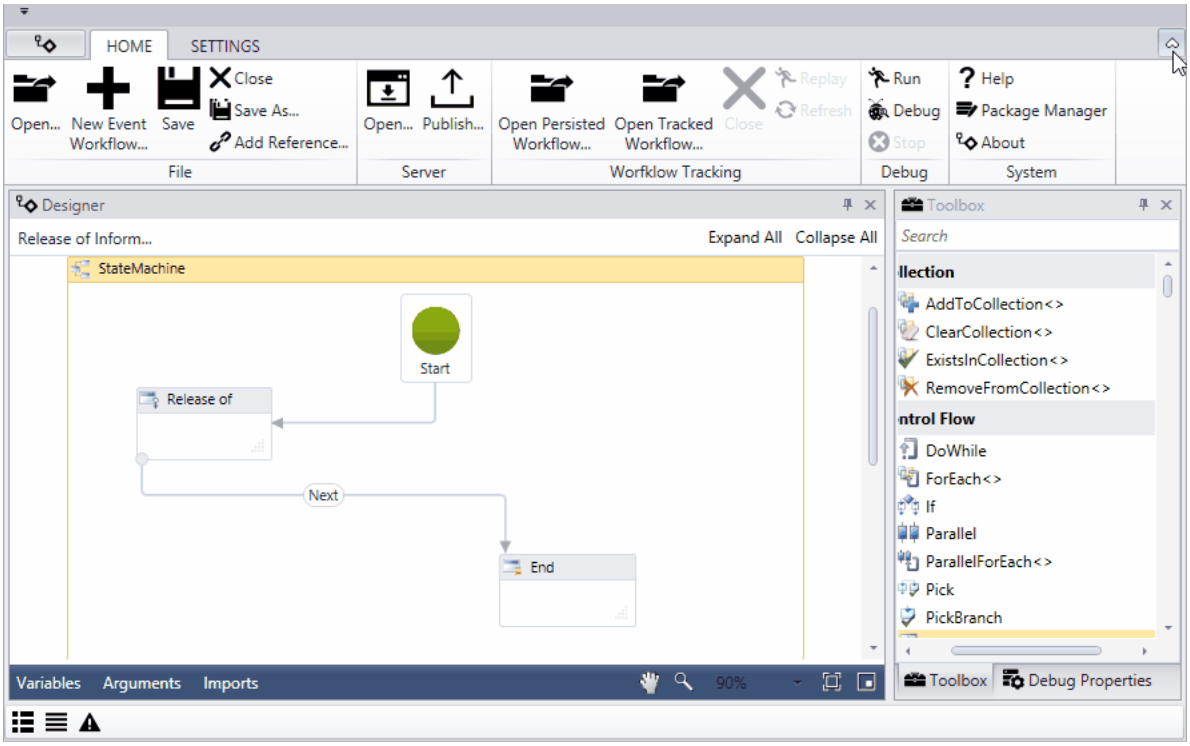


Edit the Workflow

1. In Sequence Designer, in the Sequences pane, locate and select the **Release of Information** sequence.
2. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
3. Double-click the StateMachine label and rename it to **Release of Information**.
4. Select the icon of the last state to give it the focus and press the **right arrow key** a few times to move the state to the right. This helps to organize the icons in the workflow to better navigate.



5. Hide the ribbon in Workflow Composer to maximize the Designer pane and double-click the icon on the first state in the workflow.
6. Drag a **Sequence** activity into the Entry section of the state. Rename the Sequence to **Find Student**.
7. Create a **Variable**. We want to find the student who is completing the form to display values back to our Release of Information form. Specify the following properties:
    a. Name: **studentid**
    b. Variable type: **Int32**
    c. Scope: **Release of Information**
8. Drag the **LookupUser** activity from the Toolbox into the Sequence. We use this activity to find the student completing the form. Specify the following properties:
    a. DisplayName: **Lookup Student**
    b. UserId: **studentid**
    c. UserName: **formInstance.UserName**
    d. ValidationMessages: **formInstance.ValidationMessages**



9. We want to get data from the studentEntity so we can populate some fields on our form. The GetEntity activity will accomplish this.

    a. Drag the **GetEntity** activity from the Toolbox into the Action section below the LookupUser activity. The "Select Types" dialog for the GetEntity activity is displayed.

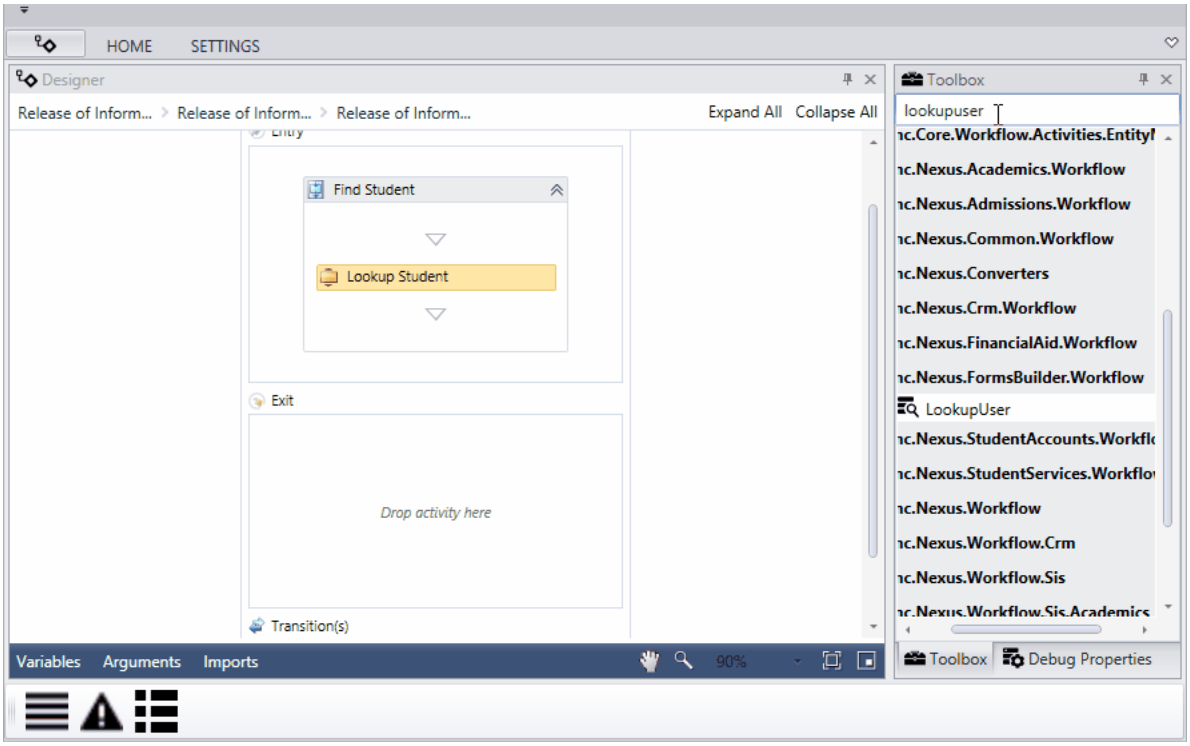
    b. Select the **drop-down** list in the Select Type window and select **Browse for Types**. The "Browse and Select a .Net Type" window is displayed.

    c. In the Type Name field, paste **StudentEntity** (or navigate to Cmc.Nexus.Common.Contracts > Cmc.Nexus.Common.Entities > StudentEntity) and select **OK**.

    d. Select **OK** in the "Select Types" dialog.

    e. Open the Properties pane for the GetEntity activity. Specify the following properties:

        - EntityId: **studentid** (This is the variable created above.)
        - Result: **studentEntity**



10. Create an **Argument** to bind the "Full Name" text box to the workflow. Specify the following properties:
    a. Name: **FullName**
    b. Direction: **In/Out**
    c. Argument type: **String**
11. Drag an **Assign** activity below the GetEntity activity. Specify the following properties:
    a. Display Name: **Assign Full Name**
    b. To field: **FullName**
    c. Value: **studentEntity.FirstName + " " + studentEntity.LastName**

    This will push the first and last name of the student to the "Full Name" text box on our form.
12. Drag a **LogLine** activity below the Assign activity. This LogLine activity writes the assigned student entity values to the log. Specify the following properties:
    a. Display Name: **Log Student Entity Values**
    b. Text: **Newtonsoft.Json.JsonConvert.SerializeObject(studentEntity, Newtonsoft.Json.Formatting.Indented)**
    c. Level: **Information**

You may need to import a reference to the Newtonsoft.Json if the LogLine activity shows an error after completing the steps above.

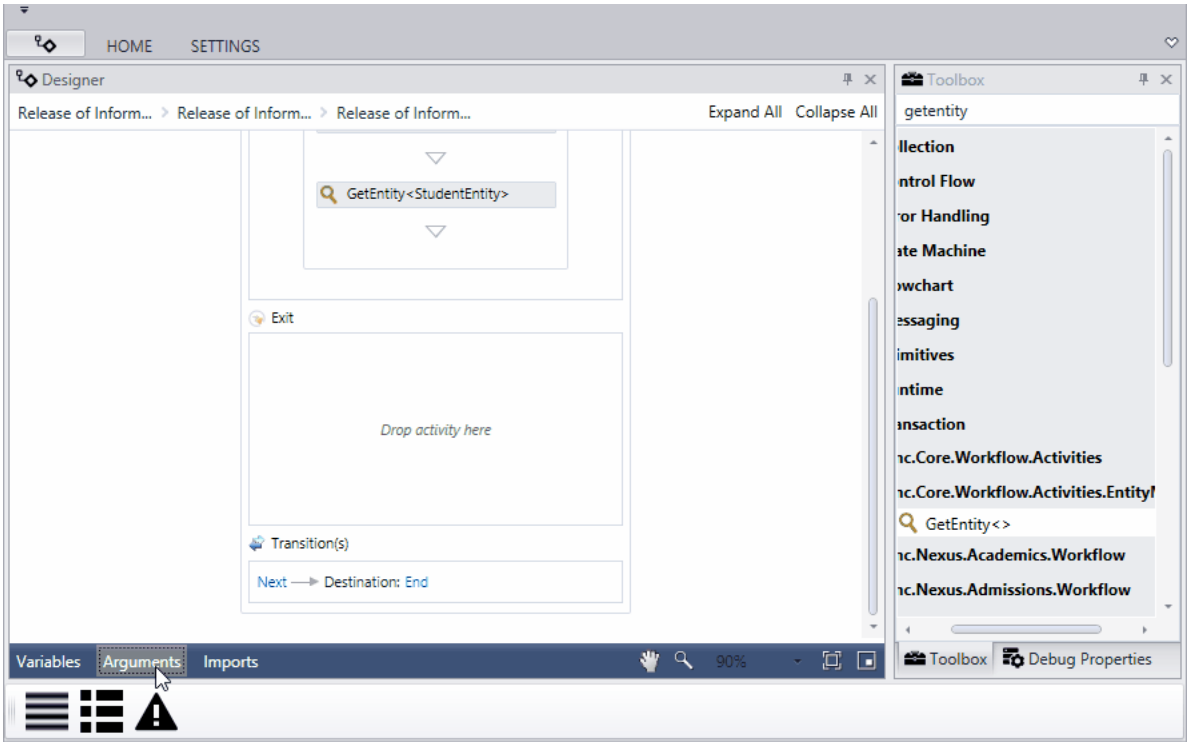
13. Drag the **CreateEntity** activity from the Toolbox below the LogLine activity. The "Select Types" dialog for the CreateEntity activity is displayed.

    a. Select the **drop-down** list in the Select Type window and select **Browse for Types**. The "Browse and Select a .Net Type" window is displayed.

    b. In the Type Name field, paste **StudentRelationshipAddressEntity** (or navigate to Cmc.Nexus.Common.Contracts > Cmc.Nexus.Common.Entities > StudentRelationshipAddressEntity) and select **OK**.

    c. Select **OK** in the "Select Types" dialog. The CreateEntity activity is added below the LogLine activity.

    d. Select the **CreateEntity** activity to give it the focus. Open the Properties pane and in the Result field, specify **studentRelationshipAddressEntity**.



14. Select the **Next** transition at the bottom of the sequence to transition to the form.

15. We want to change label on the transition in the workflow and the wording on the button on the form.

a. Change the label of the transition to **Submit**.

b. Change the label of the WaitForFormBookmark activity in the Trigger section of the transition to **Submit**. The Display Name of the WaitForFormBookmark activity will be the label of the button on the form.

c. Set the Condition to **True**.

16. Drag a **Sequence** activity in the Action section of the transition. Rename the sequence to **Create Student Relationship Address**.

17. Drag an **Assign** activity into the sequence. Specify the following properties:

    a. To field: **studentRelationshipAddressEntity.StudentId**

    b. Value: **studentEntity.Id**

18. Drag another **Assign** activity into the sequence. Specify the following properties:

    a. Display Name: **Assign Address Type**

    b. To field: **studentRelationshipAddressEntity.AddressTypeId**

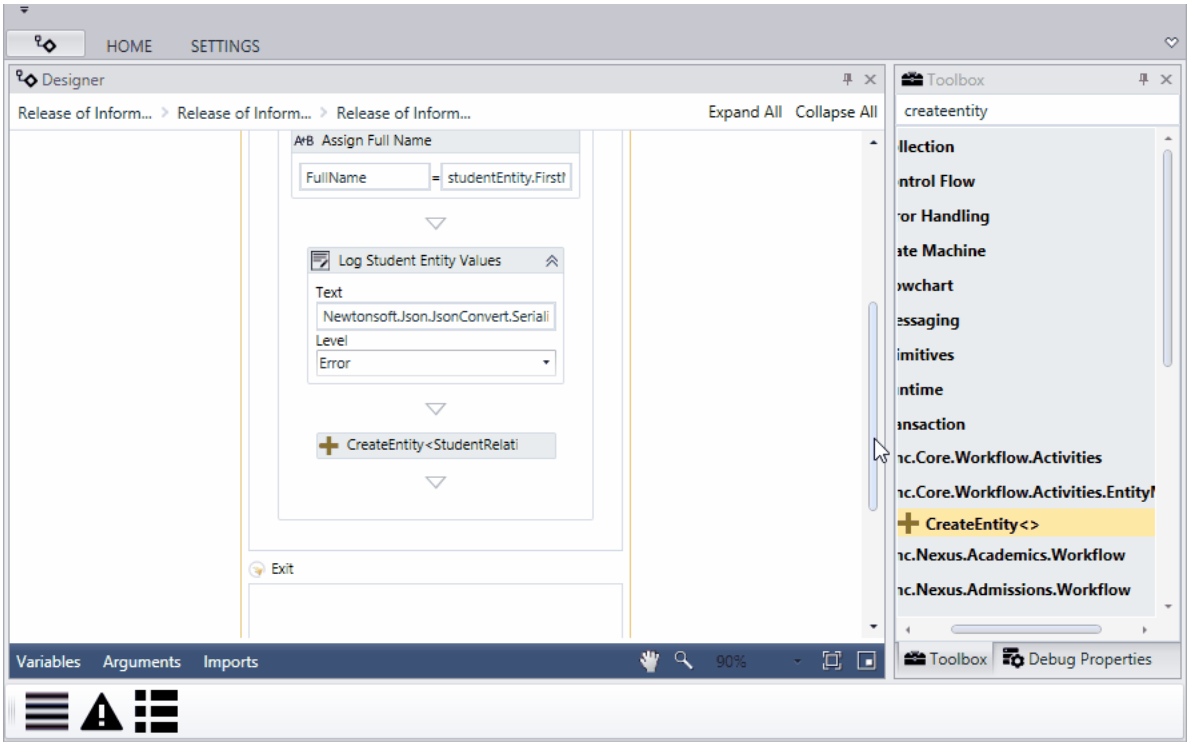
    c. Value: **25**

19. Drag a **LogLine** activity below the Assign activity. Specify the following properties:

    a. Display Name: **Log Student Relationship Values**

    b. Text: **Newtonsoft.Json.JsonConvert.SerializeObject(studentRelationshipAddressEntity, Newtonsoft.Json.Formatting.Indented)**

    c. Level: **Information**



20. Drag the **SaveEntity** activity below the LogLine activity.

    In the dialog box, select **Browse for Types**, select **Cmc.Nexus.Common.Entities.StudentRelationshipAddressEntity**, and select **OK**.

    In the Properties pane for the SaveEntity activity, Specify the following properties:

    a. Entity: **studentRelationshipAddressEntity**

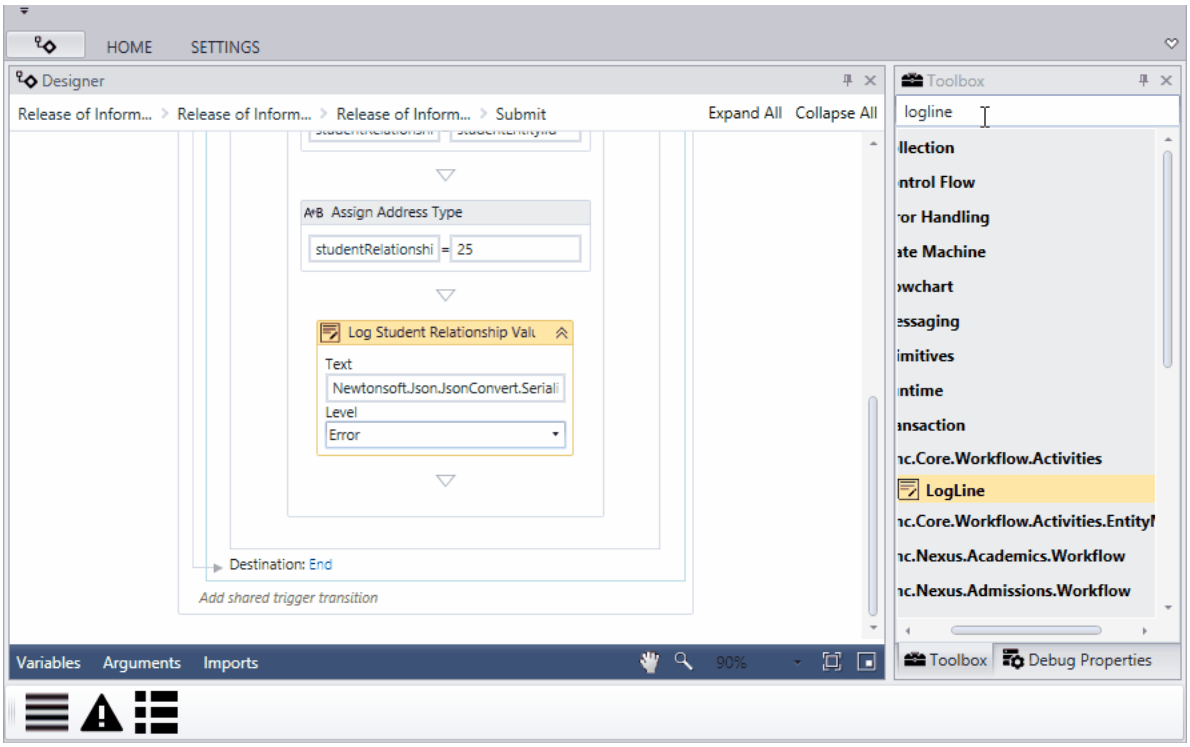
    b. ValidationMessage:**formInstance.ValidationMessages**

21. Select the cookie trail at the top left of the Designer pane to return to the top level of the workflow flowchart.

22. Confirm there are no errors in the **Error** panel.

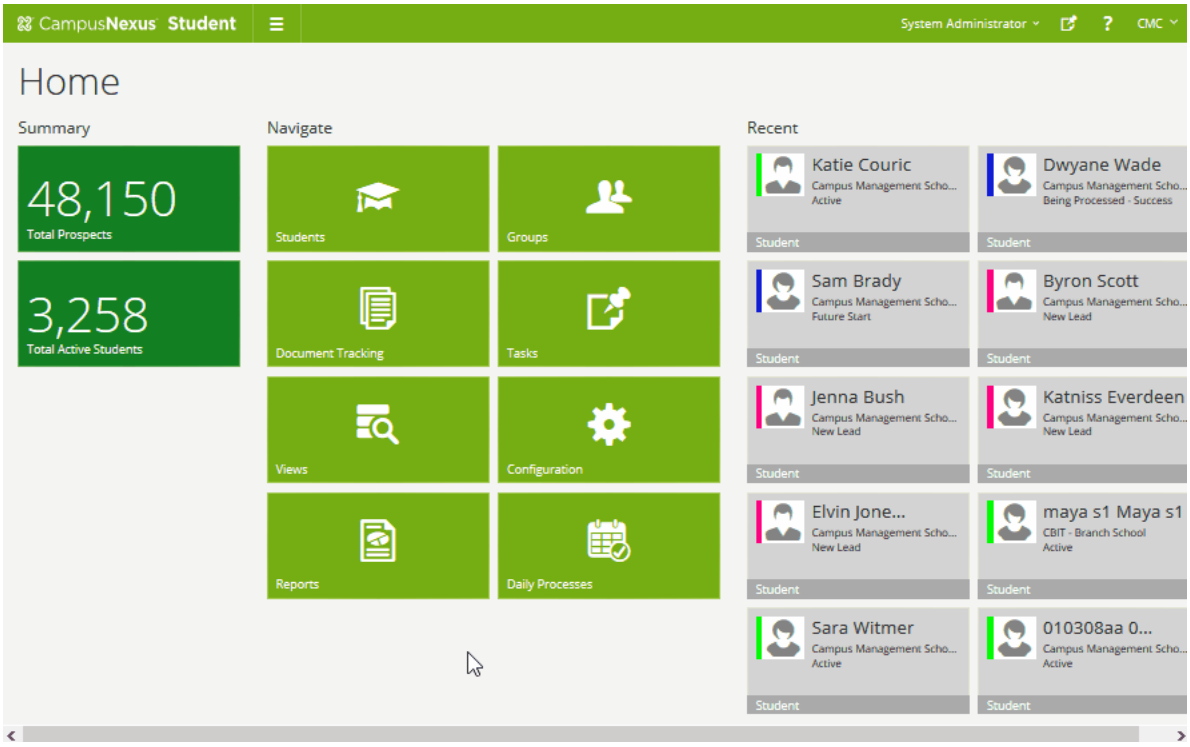
23. Restore the **ribbon** of Workflow Composer.

24. Select **Publish** and select **Enable This Workflow Version?** Select **OK** to confirm.



Validate the Data in the Web Client

1. Open the Web Client for Anthology Student in another browser.
2. Sign in with your user name and password.
3. Select the **Students** tile.
4. Search for a student. Select the student by selecting the name link.
5. Expand **Contact Manager** and select the **Related Addresses** tile.



Submit the Release of Information Form

1. Open a different browser and point to the URL for the **Sequence List**.

   (`http://<server>:<port>/#/sequencelist` or `https://<server>:<port>/#/sequencelist`)

2. Use the search box above the Sequence Name column to find the **Release of Information** sequence.

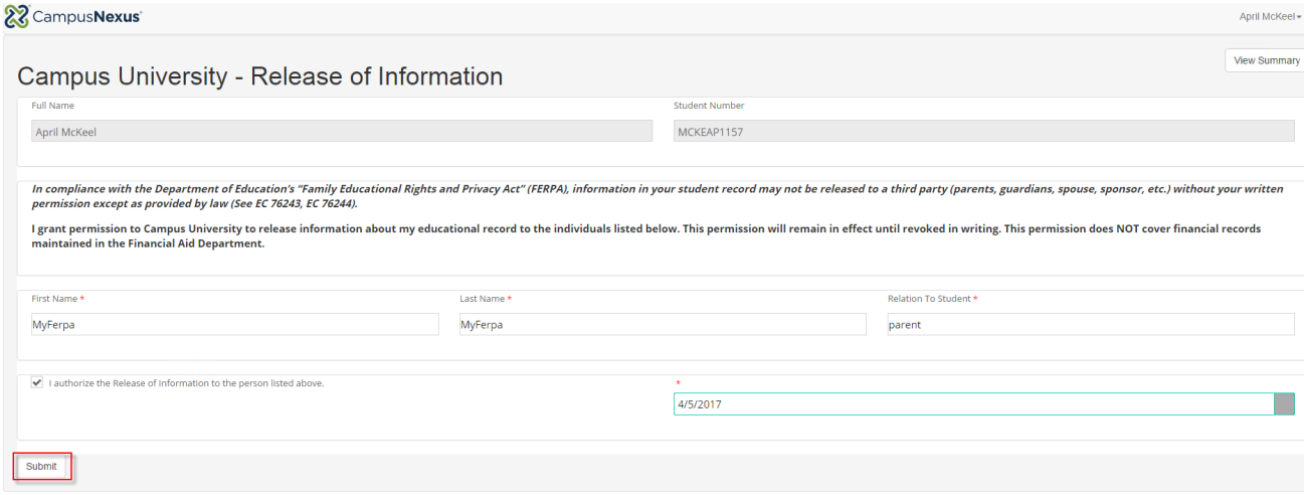
3. Scroll to the right and select  to view the sequence.

4. Login as the student:

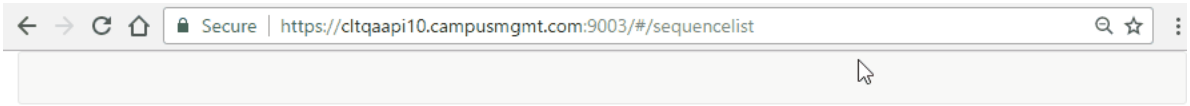
   Username: April.McKeel

   Password: nexus123

5. Complete the form as shown below and select **Submit**.



6. Confirm that you received a Confirmation Message.



Confirm the Updates in the Web Client

1. Open the Web Client for Anthology Student in another browser.
2. Sign in with your user name and password.
3. Select the **Students** tile.
4. Search for a student. Select the student by selecting the name link.

5. Expand **Contact Manager** and select the **Related Addresses** tile.
6. Confirm the Related Addresses was updated.
7. Confirm that if you try to add FERPA access information for a person that your person can be selected in the drop-down list.

Check the Renderer Log

1. Navigate to: **\\<server>\c$\logs** and open the FormsBuilderRenderer file that has today's date.
2. Navigate to the bottom of the file (Ctrl + End) and then scroll up until you find the log lines from your workflow.

Request for Information Form

In this section, we will build a Request for Information (RFI) form and a workflow that will create a student record in Anthology Student, create the student's Portal account, and notify the student.

> ⚠ The screen captures in this topic show an earlier Forms Builder version. While the UI has been updated, the basic functionality is unchanged.

Build the Form

1. Open your browser and point to the Forms Builder URL.
2. Sign in using your user name and password.
3. On the Forms Builder home page, select the Form Designer tile. Products (if applicable), Entities and Forms are loaded into Forms Builder.
4. In the **Select Provider** drop-down list, select **Student**.

   This step is needed only if your Forms Builder installation uses the databases of both CampusNexus CRM and Anthology Student.
5. Select **New**. A 1-column panel is added to the Layout pane.



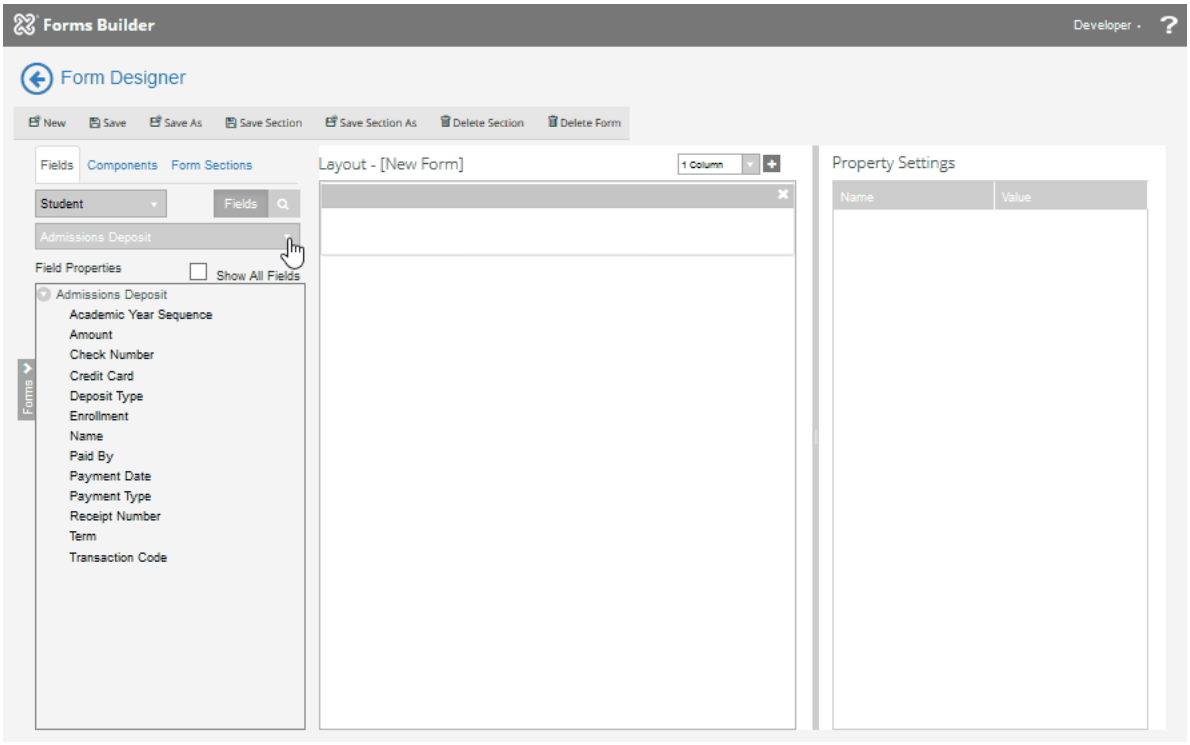
6. Select the **Prospect Inquiry** entity from the drop-down list.

   Any time a prospective student inquires about possible enrollment, a prospect inquiry record is created, and the information is tracked within Anthology Student. The Campus is a required attribute of the prospect inquiry record.

7. Drag the **Campus** field into the Layout pane.

   The Campus field is a drop-down control that is populated with values from the Anthology Student database. The binding to the database is accomplished by the Model value *vm.models.prospectInquiryEntity.CampusId*, which is automatically populated in the Property Settings. The values for the drop-down list are obtained by the Lookup Query *Campuses?$select=Code, Name, Id&$filter=IsActive eq true&$orderby=Name*, which is also populated automatically.

8. In the Property Settings panel, change the Option Label property to **<Select Campus>**.



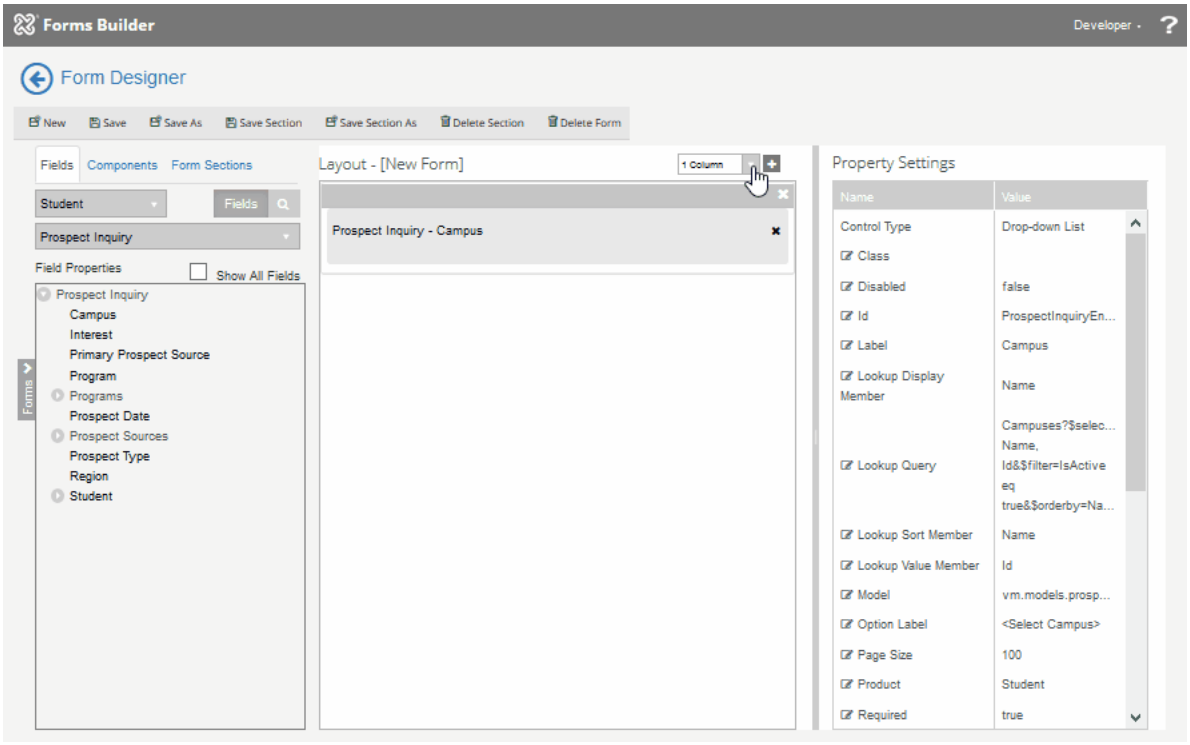
9. In the Column drop-down list, select **3 Columns** and select  to add a new panel to the form layout.

10. From the Prospect Inquiry entity, under Student, drag **First Name**, **Middle Name**, and **Last Name** into the Layout pane.

    The name fields are text box controls with a default type of *text*. The fields are bound to the database under the *prospectInquiryEntity.Student* model (*vm.models.prospectInquiryEntity.Student.LastName*,

vm.models.prospectInquiryEntity.Student.MiddleName, and *vm.-models.prospectInquiryEntity.Student.LastName*).

The Required property for these fields is set to true by default.



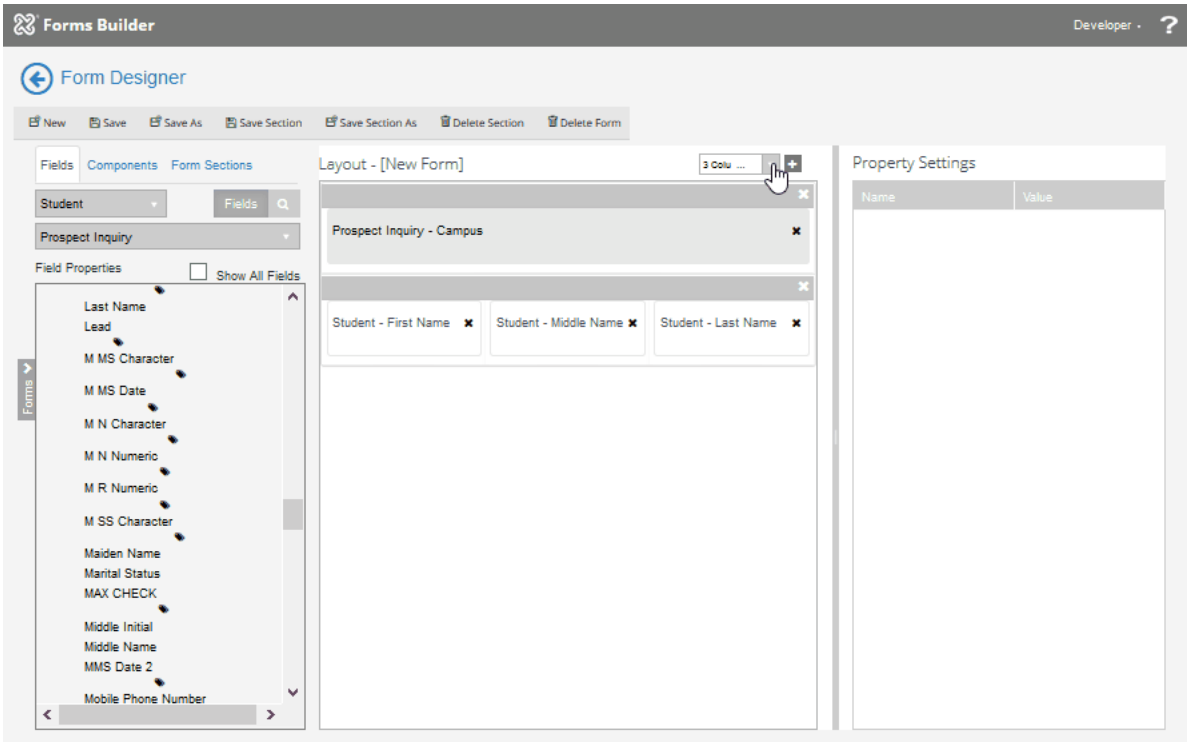
11. In the Column drop-down list, select **1 Column** and select  to add a new panel to the form layout.
12. From the Prospect Inquiry entity, below Student, drag **Street Address** into the new layout panel.

    You can use the Tooltip property on any field to provide additional information to the user, e.g., *If an address has more than 40 characters, the additional characters in the address line will not be extracted to be sent to COD but will be truncated.*
13. Change the Required property for Street Address to **true**.

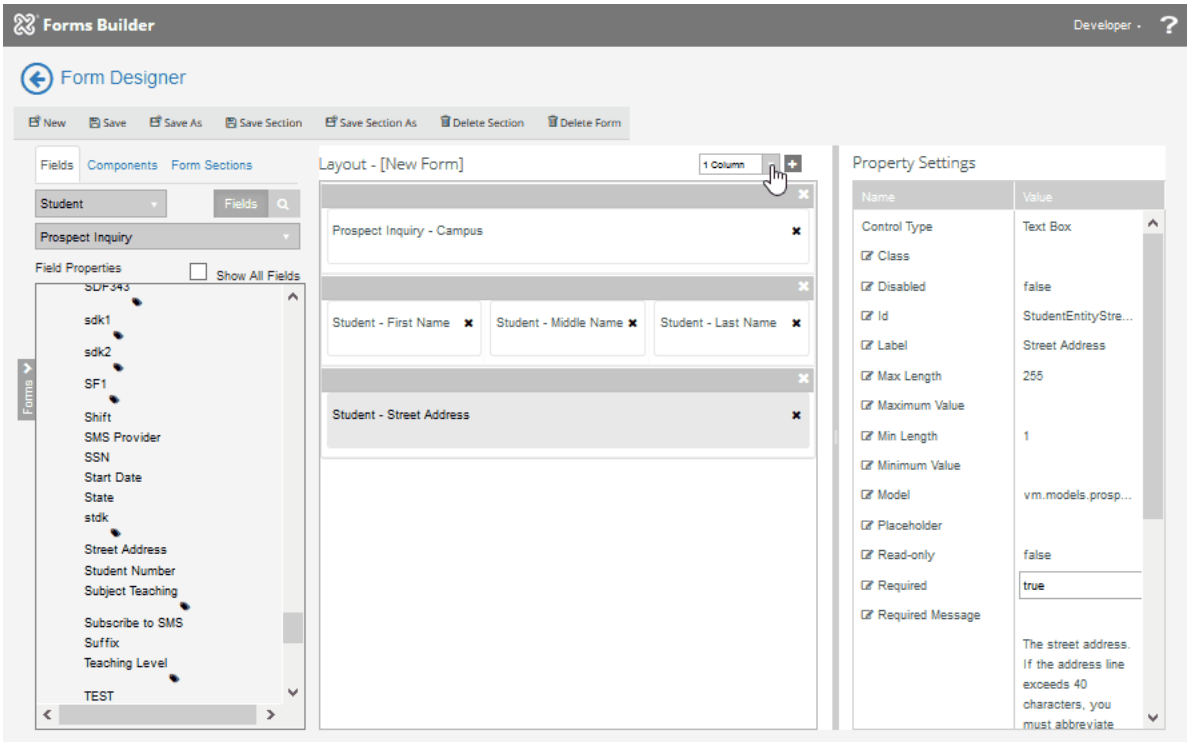
    Required makes the control required and will raise a validation error on the form. It is set to false by default. If input is required, set the property value to true. The rendered form will display a red asterisk (*) next to the component.

    - Can be bound to a workflow argument or another control's value.
    - A property array string index requires single quotes, e.g., vm.models.xxx.CustomProperties ['yyyyy'].

- An expression can be used that evaluates to true or false, e.g., vm.models.myvalue==7 (>,<, !=, ==, >=, <=).
- If comparing to a string, it must be in single quotes.
- (true and false must be all lowercase)

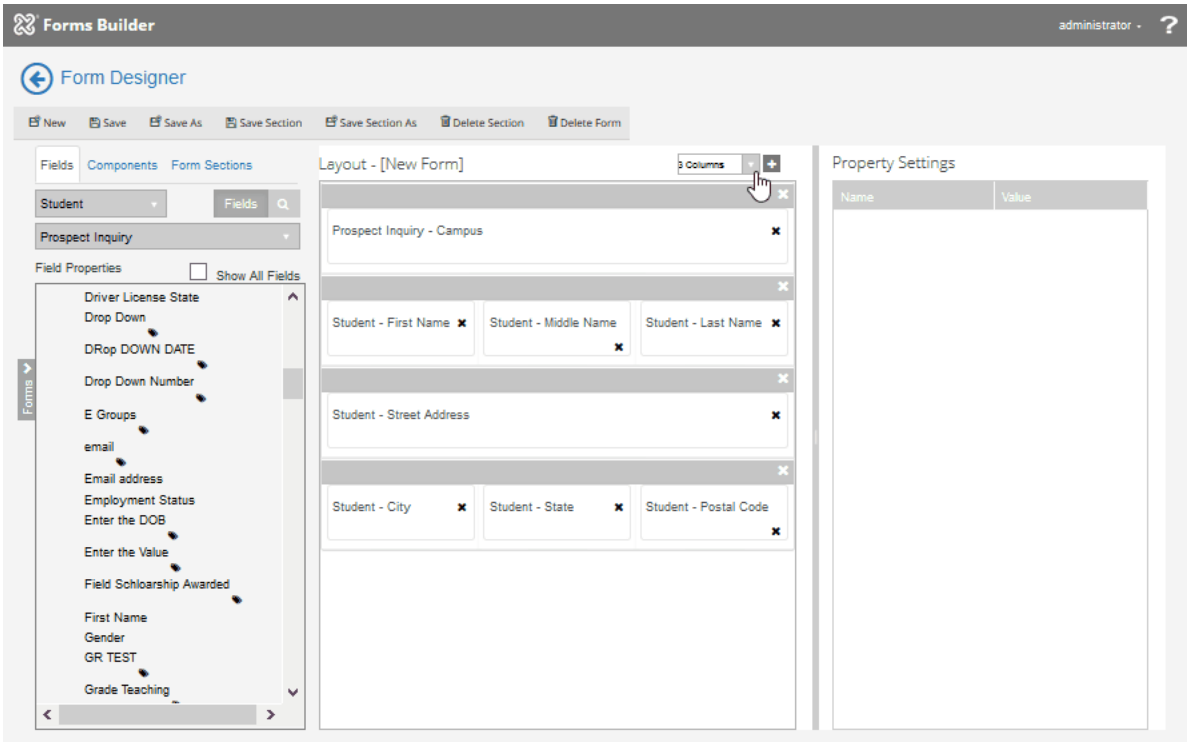


14. In the Column drop-down list, select **3 Columns** and select + to add a new panel to the form layout.
15. From the Prospect Inquiry entity, below Student, drag **City**, **State**, and **Postal Code** into the layout panel.
16. Change the Required property for all three fields to **true**.
17. Change the Option Label property for State to **<Select State>**.



18. In the Column drop-down list, select **3 Columns** and select  to add a new panel to the form layout.
19. From the Prospect Inquiry entity, below Student, drag **Email address** into the layout panel. Specify the following properties:
    a. Required: **true**
    b. Type: **email**

    The Email address field is a text box that is bound to *vm.models.prospectInquiryEntity.Student.EmailAddress*. This text box control does not use the default text box type of *text*, but instead uses the text box type of *email*. When the email type is selected, Forms Builder validates the entry in the text box for proper format.



20. From the Components tab, drag a **Masked Text Box** into the panel next to Email address. Specify the following properties:

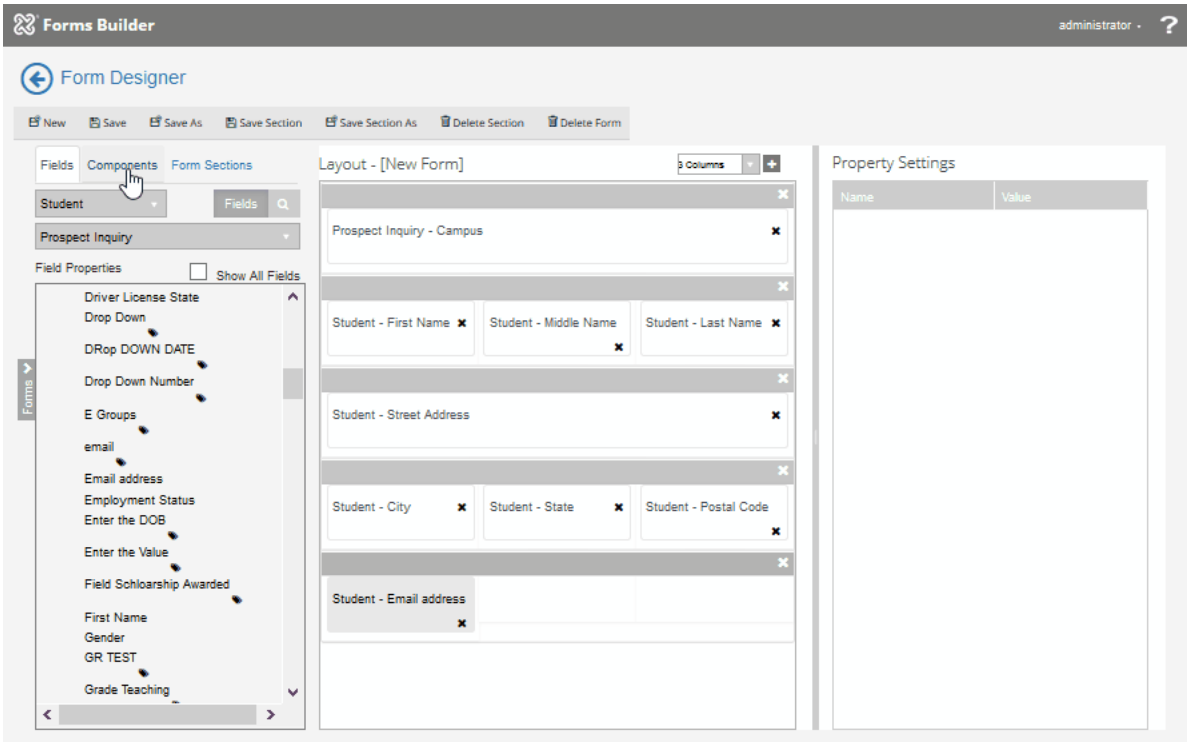
    a. Format: **(###)###-####**

    b. Label: **Phone Number**

    c. Model: **vm.models.prospectInquiryEntity.Student.PhoneNumber**

    Some of the controls listed on the Components tab provide a Model property, but unlike controls that are listed on the Fields tab, the Model property value is not automatically populated for components. If the Model value is not specified, the component will be displayed on the form, but any values the end user enters on the form cannot be used in the workflow. If you want the component to be bound to the workflow, specify a Model value. The Model value is always prefixed with *vm.models*. The string that is appended to *vm.models* can be added as an In/Out argument in the associated workflow.

    d. Required: **true**



21. From the Components tab, drag another **Masked Text Box** into the panel next to the Masked Text Box for the Phone Number. Specify the following properties:

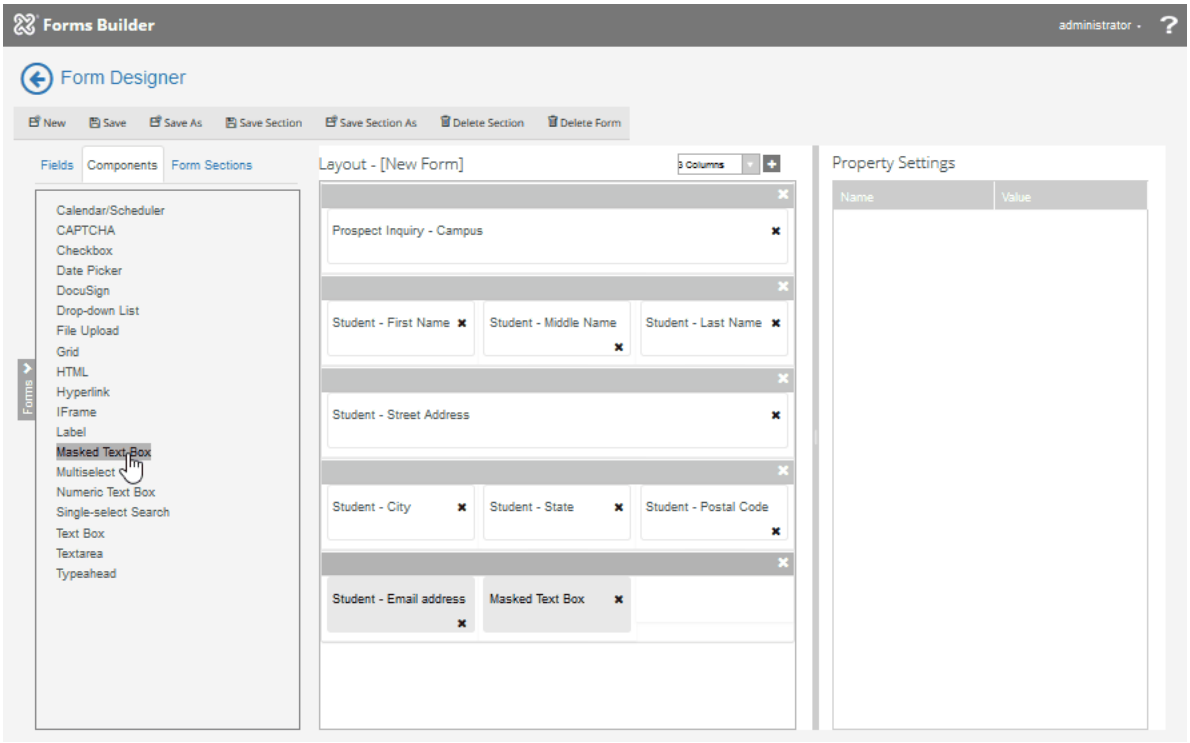
    a. Format: **(###)###-####**

    b. Label: **Mobile Phone Number**

    c. Model: **vm.models.prospectInquiryEntity.Student.MobilePhoneNumber**

    d. Required: **true**

The Model property binds the Masked Text Boxes for Phone Number and Mobile Phone Number the corresponding database fields. The masking must be (###)###-#### because this is the format of phone numbers in the Anthology Student database.



22. In the Column drop-down list, select **1 Column** and select  to add a new panel to the form layout.
23. From the Components tab, drag **Multiselect** into the new panel. Specify the following properties:
    a. Label: **Program**
    b. Lookup Display Member: **Name**
    c. Lookup Query: <refer to Note>

       Note: Program is a multiselect field in the Anthology Student user interface. The Multiselect component needs a Lookup Query to retrieve the values for the multiselect field from the database.

       To create a Lookup Query, you can either:

       - Copy the query from the Program ID field under ProspectInquiry > Student and paste it in the Lookup Query property for the Multiselect component (Programs?$select=Code,Name,Id&$filter=IsActive eq true).

         — OR —

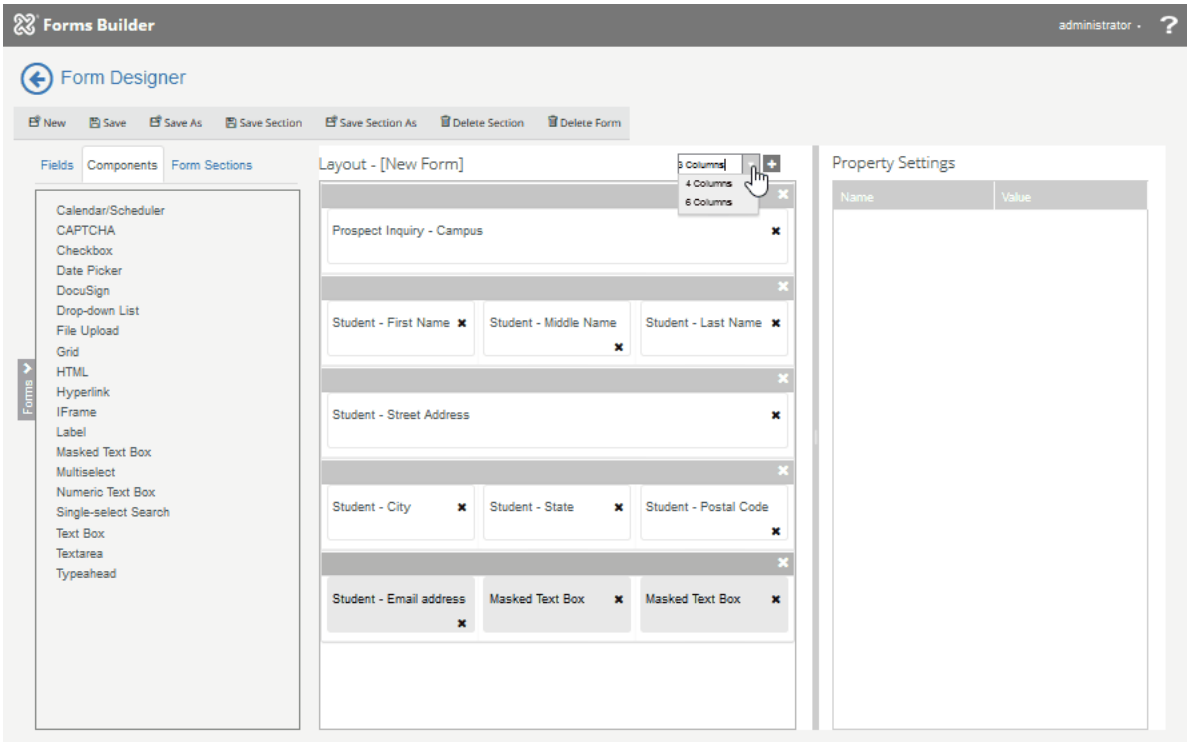
       - Query the data in the Web Client for Anthology Student as shown in the next section.

    d. Lookup Sort Member: **Name**
    e. Lookup Value Member: **Id**
    f. Model: **vm.models.prospectInquiryEntity.Student.ProgramsList**

The Multiselect component needs to be bound to the programs list in the CampusNexusentity model to retrieve values from the database.

g. Option Label: **<Select your Program of interest>**

h. Required: **true**

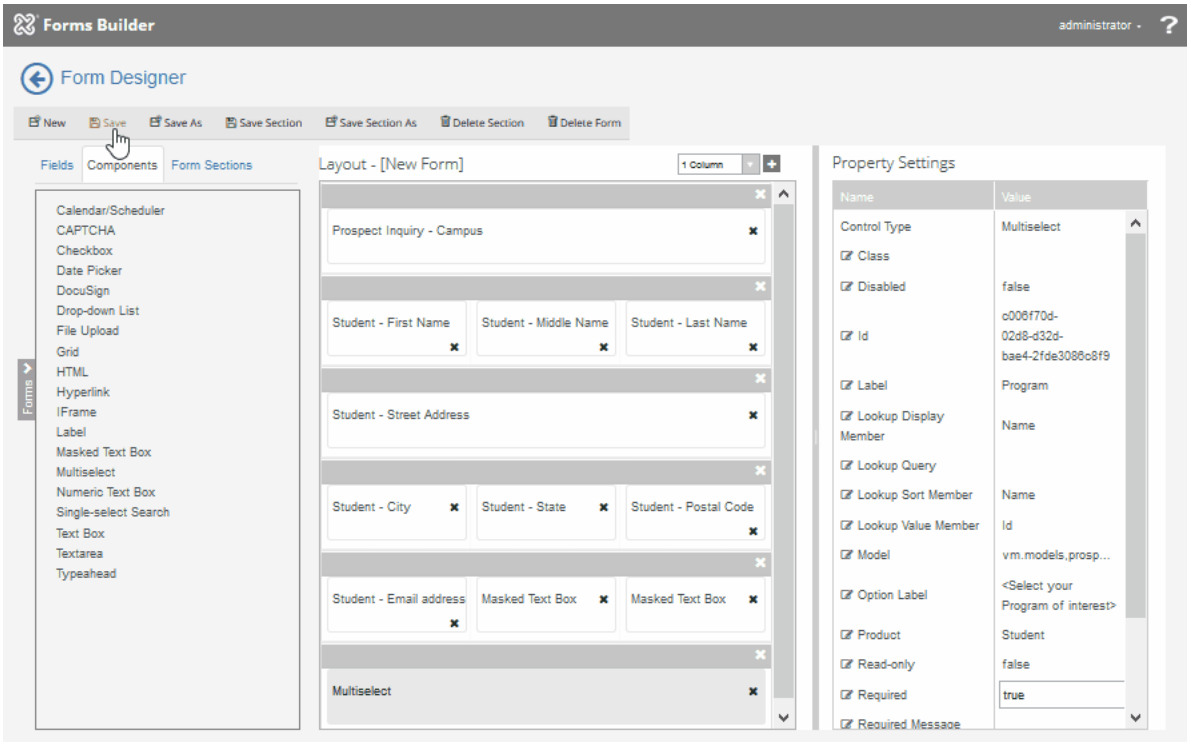


24. Select **Save** to save what you have done so far on your form. Specify the following properties:

    a. Form Name: **Campus University - Request for Information**

    b. Title: **Campus University - Request for Information**

    The title is optional. When a title is specified, it is displayed on the rendered form.

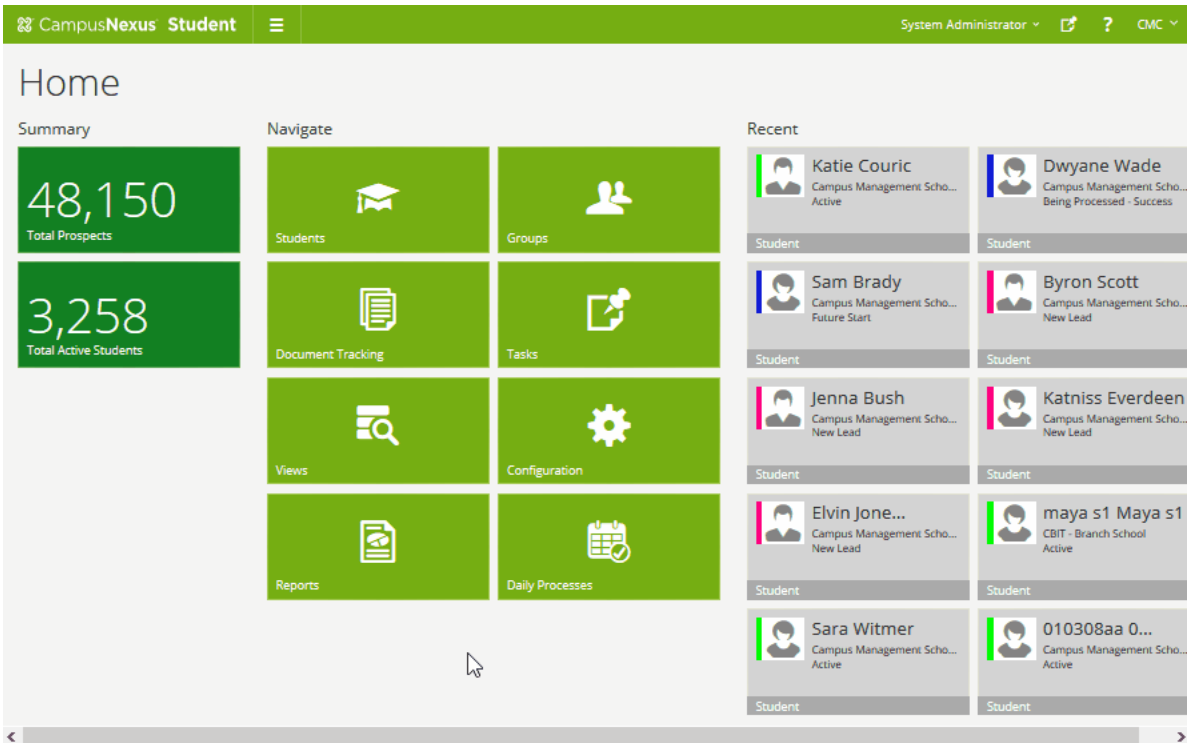


Create a Query in the Web Client

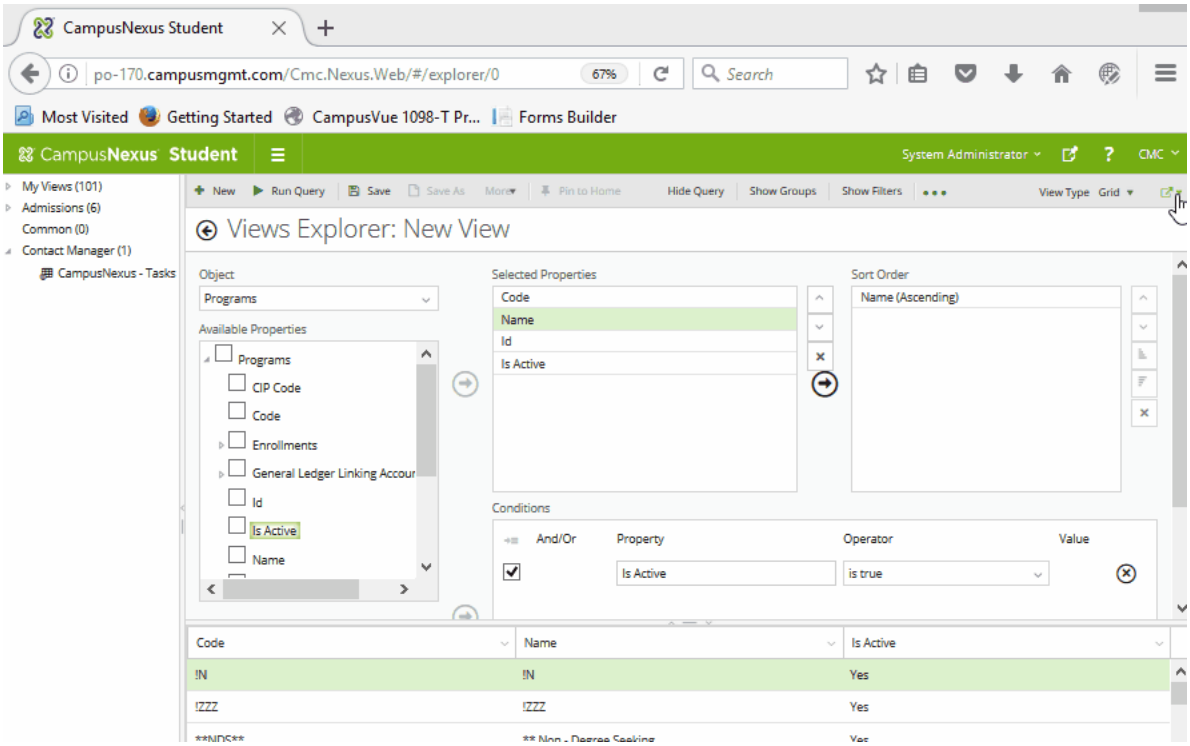
1. Open the Web Client for Anthology Student in another browser.
2. Sign in with your user name and password.
3. Select the **Views** tile.
4. Select 

to create a new view.
5. In the Object drop-down list, under Academics, select **Programs**.
6. The Selected Properties pane lists the default query properties. For this query, we only want the Code, Id, and Name.
    a. Select **CIP Code** and select ✖ to delete the property.
    b. Select **Is Active** and select ✖ to delete the property.
7. Select **Id** and select  to move Id below Name.
8. Select **Name** and select  to move Name to the Sort Order pane. By default, the sort order is ascending, and this is what we want.

9. Select **Is Active** in the Available Properties pane and select ➔ to move the property to the Conditions pane. This will only filter active programs.
10. Select **Run Query**. The records returned by your query are displayed.



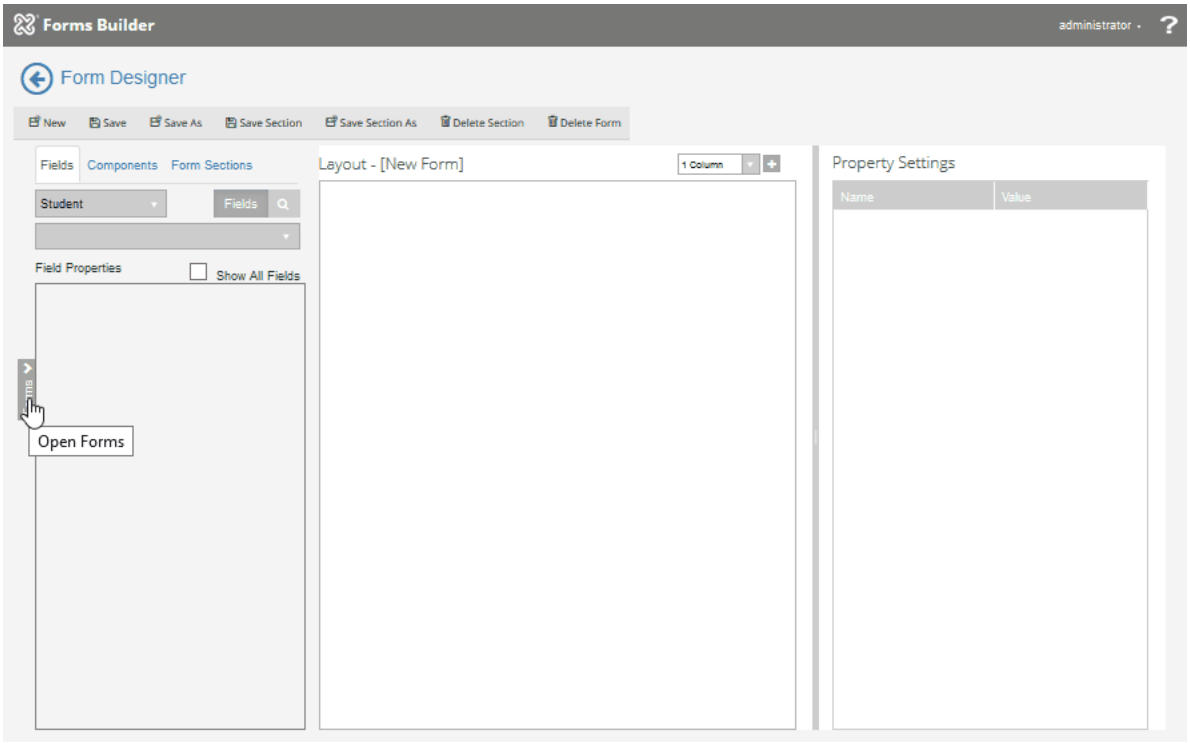
11. Select the drop-down list on the far right of the toolbar and select **Query URL**. The "Copy the URL to the clipboard" popup is displayed.
12. Press **Ctrl + C** to copy the URL and select **OK**.
13. Open a new tab in your browser, press **Ctrl + V** to paste the URL into the address bar, and press **Enter**. The browser returns the metadata of the query.
14. Select the end portion of the URL, which is the query. Select the string from "Programs..." forward and copy it to the clipboard (**Ctrl + C**).



Add the Query to the Form

1. In Form Designer, return to the "Campus University - Request for Information" form.
2. Select the **Multiselect** (Program) field in the Layout pane.
3. Paste (**Ctrl + V**) your query into **Lookup Query** property. It should look exactly like this:

   Programs?$select=Code,Name,Id&$filter=IsActive eq true
4. **Save** your form.
5. Select  to return to the home page of Forms Builder.



Create a Sequence

1. On the home page of Forms Builder, select the **Sequence Designer** tile.
2. Select **New**. A new panel is added to the Layout pane.
3. In the search bar of the Forms pane:
    a. Type **welcome** and drag the **Welcome** form into the Layout pane.

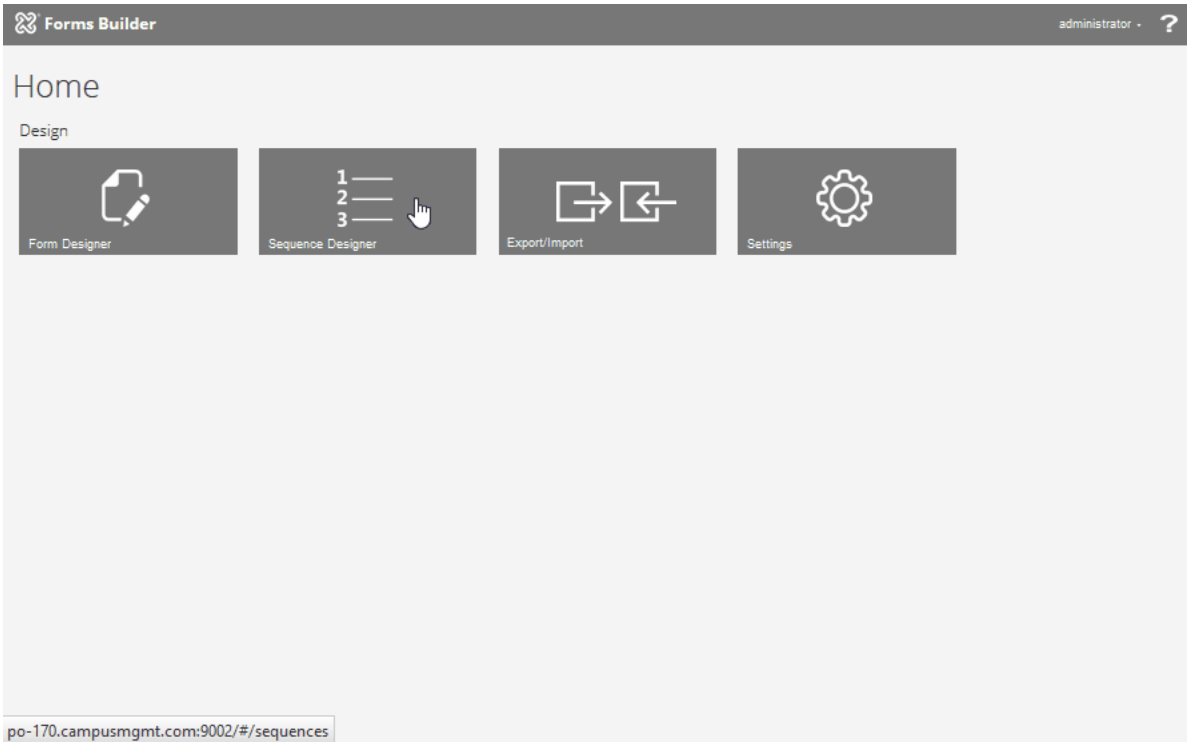
    The Welcome form is an out-of-the-box form that can be modified and added to each sequence (refer to Welcome and Confirmation Forms). The prospective student selects Next on this form and then proceeds through the sequence.

    b. Type **campus** and drag the **Campus University – Request for Information** form below the Welcome form.
4. In the Properties pane:
    a. In Authentication field, select the check box to set authentication to **true**.
    b. In the Authentication Product field, select **Student**.

    This step is needed only if your Forms Builder installation uses the databases of both CampusNexus CRM and Anthology Student.
5. **Save** the sequence. Specify the following properties:

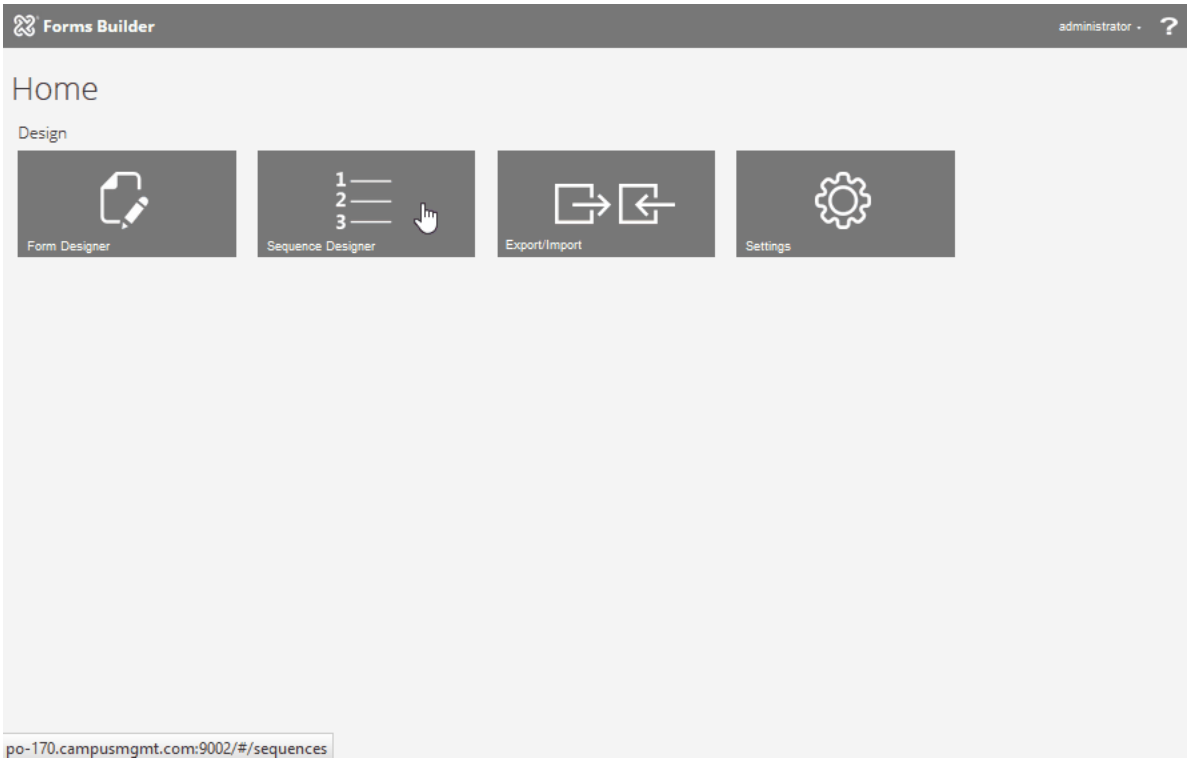
a. Sequence Name: **Campus University – Request for Information**

b. Title: **Campus University – Request for Information**



Edit the Workflow

1. Select the **Sequence Designer** tile on the Forms Builder home page.
2. Use the search box above the Sequences pane to find the **Campus University – Request for Information** sequence.
3. Launch Workflow Composer and open the workflow. Refer to Open the Workflow for a Sequence.
4. Double-click the StateMachine label and rename it to **Campus University – RFI**.
5. Select the icon of the second state to give it the focus and press the **right arrow key** a few times to move the state to the right. This helps to organize the icons in the workflow to better navigate.



6. When a prospective student completes the RFI, we want to capture the data from the form and update the Anthology Studentdatabase. Since our form is built with fields from the Prospect Inquiry entity, the first step is to create a Prospect Inquiry entity record. We will create the entity when the student selects Next on the Welcome form.

   Double-click the icon on the Welcome state (first state in the workflow) and select the **Next** link at the bottom of the transition. Specify the following properties:

   a. Enter **true** in the Condition field of the Next transition.

      When the Condition evaluates to true (i.e., the student selects Next), the Action of the transition is executed.

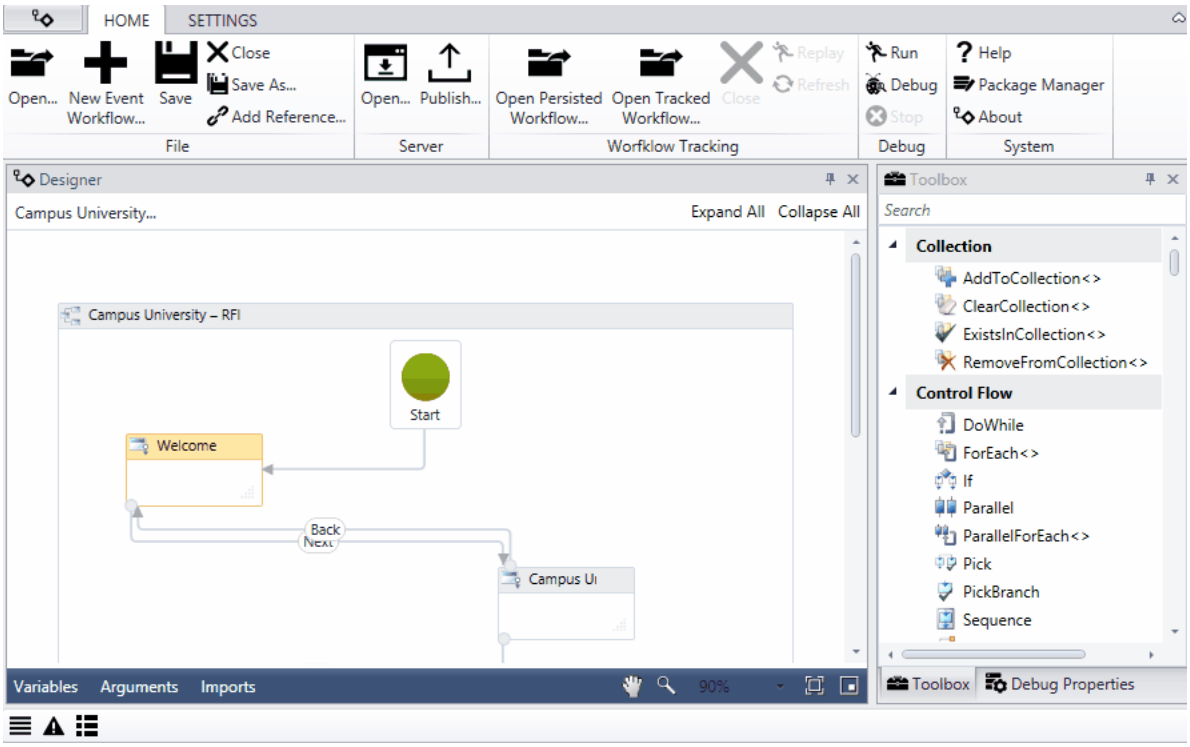
   b. Drag the **CreateEntity** activity from the Toolbox into the Action section below the Condition field. The "Select Types" dialog for the CreateEntity activity is displayed.

   c. Select the **drop-down** list in the Select Type window and select **Browse for Types**. The "Browse and Select a .Net Type" window is displayed.

   d. In the Type Name field, paste **ProspectInquiryEntity** (or navigate to Cmc.Nexus.Admissions.Contracts > Cmc.Nexus.Admissions.Entities > ProspectInquiryEntity) and select **OK**.

   e. Select **OK** in the "Select Types" dialog. The CreateEntity activity is added to the Action section of the Next transition.

f. Select the **CreateEntity** activity to give it the focus. Open the Properties pane and in the Result field, specify **prospectInquiryEntity**.



7. When the prospective student selects Next on the Welcome form, the next form is the "Campus University – Request for Information" form. When this form is completed and submitted, the workflow updates data in Anthology Student and automates other tasks, such as creating a Portal account and sending emails.

    a. Select the Destination link for the **Campus University – Request for Information** state.

    b. Select the **Next** link at the bottom of state.

    c. Change the label of the WaitForFormBookmark activity in the Trigger section of the transition to **Start your Journey!** The Display Name of the WaitForFormBookmark activity will be the label of the button on the form.

    d. Set the Condition to **True**.

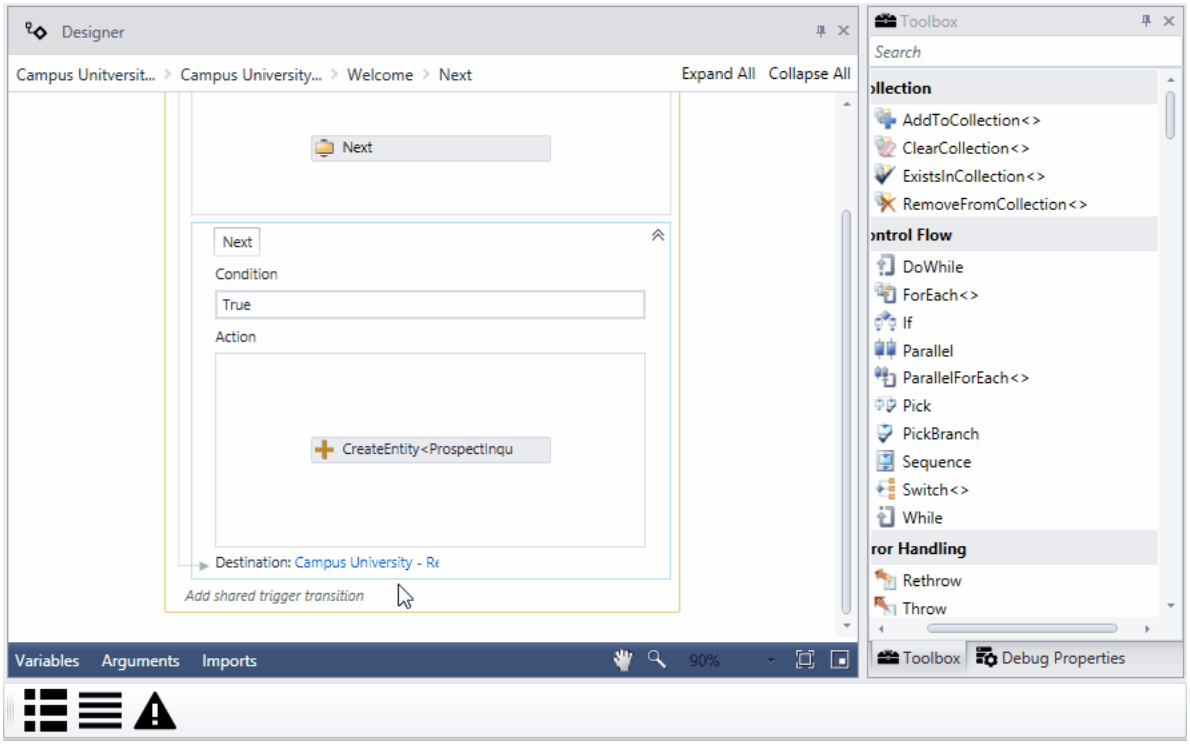
8. In the Action section below the Condition field, we want to accomplish the following:

    - Update values for the prospect and save the values to Anthology Student.
    - Create a Portal account and send an email with the credentials to the prospective student.
    - Send an email to the prospective student detailing the next steps in the application process.

    The workflow activities for these tasks will be placed in a Sequence activity.

Drag a **Sequence** activity from the Toolbox into the Action section. Specify the Display Name: **Assign Values**

9. Drag 7 **Assign** activities into the "Assign Values" sequence.



Specify the following properties: (each row in the table below represents an Assign activity)

In this example, we hard-coded the assignments to simplify the workflow definition. In a live environment, you would use variables and the LookupReferenceItem activity.

These assignments provide values for required fields in prospect inquiry records. Query your Anthology Student database to obtain appropriate values for your environment. Refer to Anthology Student entities reference Prospect Inquiry and Student.

Assign	Display Name	"To" Field	Value
1	Assign School Status	prospectInquiryEntity.Student.SchoolStatusId	88
2	Assign Lead Type	prospectInquiryEntity.LeadTypeId	11
3	Assign Lead Source	prospectInquiryEntity.LeadSourceId	711
4	Assign Admin Rep	prospectInquiryEntity.AssignedAdmissionsRepId	28
5	Assign Country ID	prospectInquiryEntity.Student.CountryId	10

Assign	Display Name	"To" Field	Value
6	Assign Lead Date	prospectInquiryEntity.LeadDate	DateTime.Now
7	Set as Current Inquiry	prospectInquiryEntity.IsCurrentInquiry	True

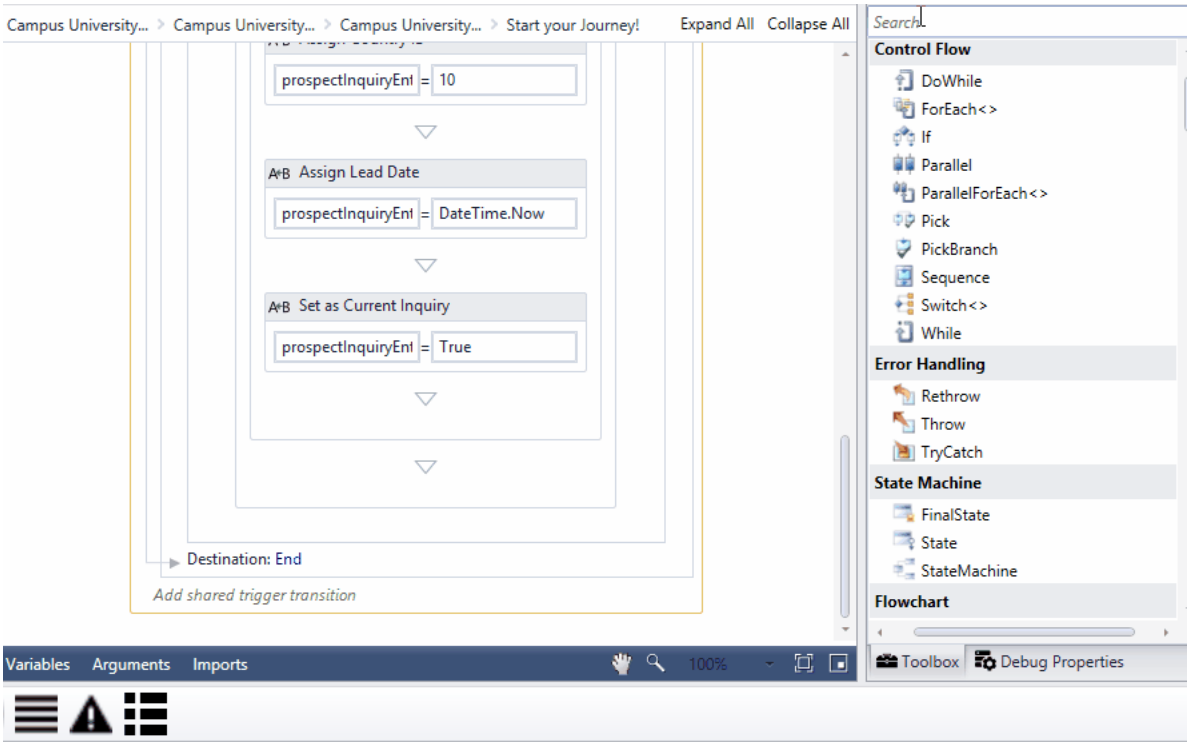
10. Drag a **LogLine** activity below the "Assign Values" sequence. This activity writes the assigned prospect inquiry values to the log. Specify the following properties:

    a. Display Name: **Log Prospect Inquiry Values**

    b. Text: **Newtonsoft.Json.JsonConvert.SerializeObject(prospectInquiryEntity, Newtonsoft.Json.Formatting.Indented)**

    c. Level: **Information**

    You may need to import a reference to the Newtonsoft.Json if the LogLine activity shows an error after completing the steps above.



11. Drag the **SaveEntity** activity below the LogLine activity.

    In the dialog box, select **Browse for Types**, select **Cmc.Nexus.Admissions.Entities.ProspectInquiryEntity**, and select **OK**.

    In the Properties pane for the SaveEntity activity, Specify the following properties:

a. Entity: **prospectInquiryEntity**

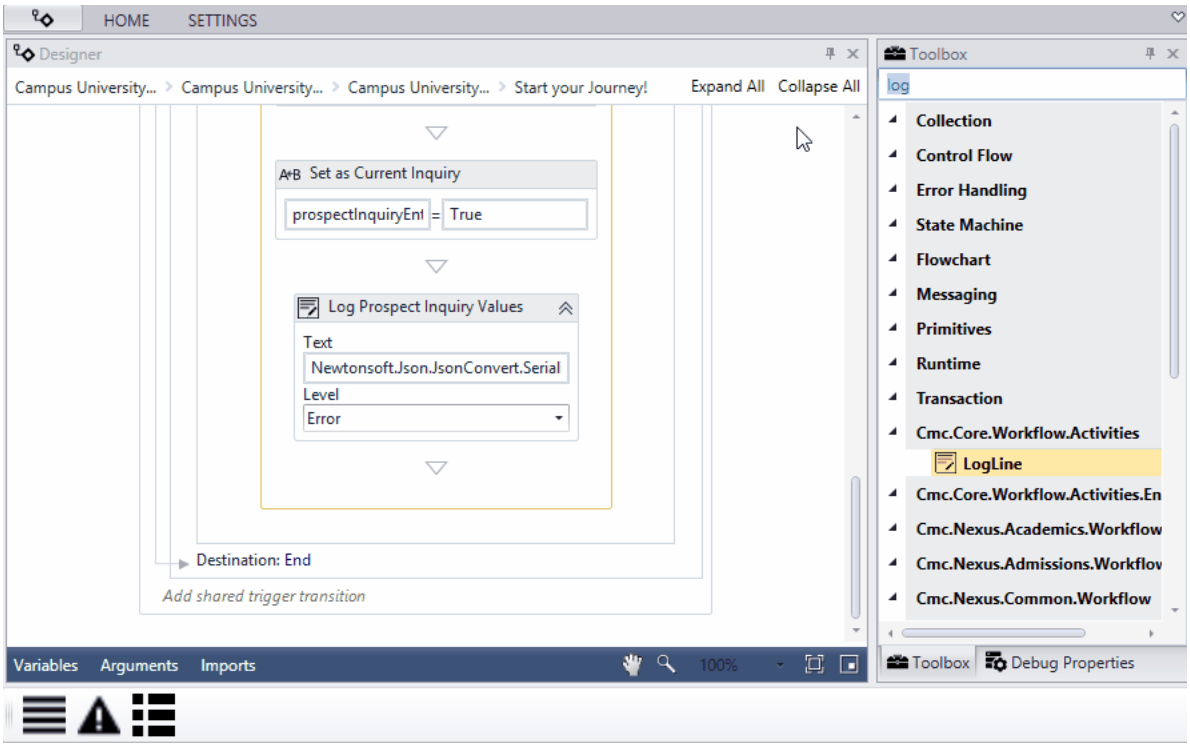
b. ValidationMessage:**formInstance.ValidationMessages**

12. Drag a **LogLine** activity below the SaveEntity activity. This LogLine activity writes the student ID values to the log. Specify the following properties:

    a. Display Name: **Log Student ID**

    b. Text: **environment.NewLine + Environment.NewLine + "The student ID is: " + prospectInquiryEntity.StudentId.ToString + Environment.NewLine + Environment.NewLine**

    c. Level: **Information**

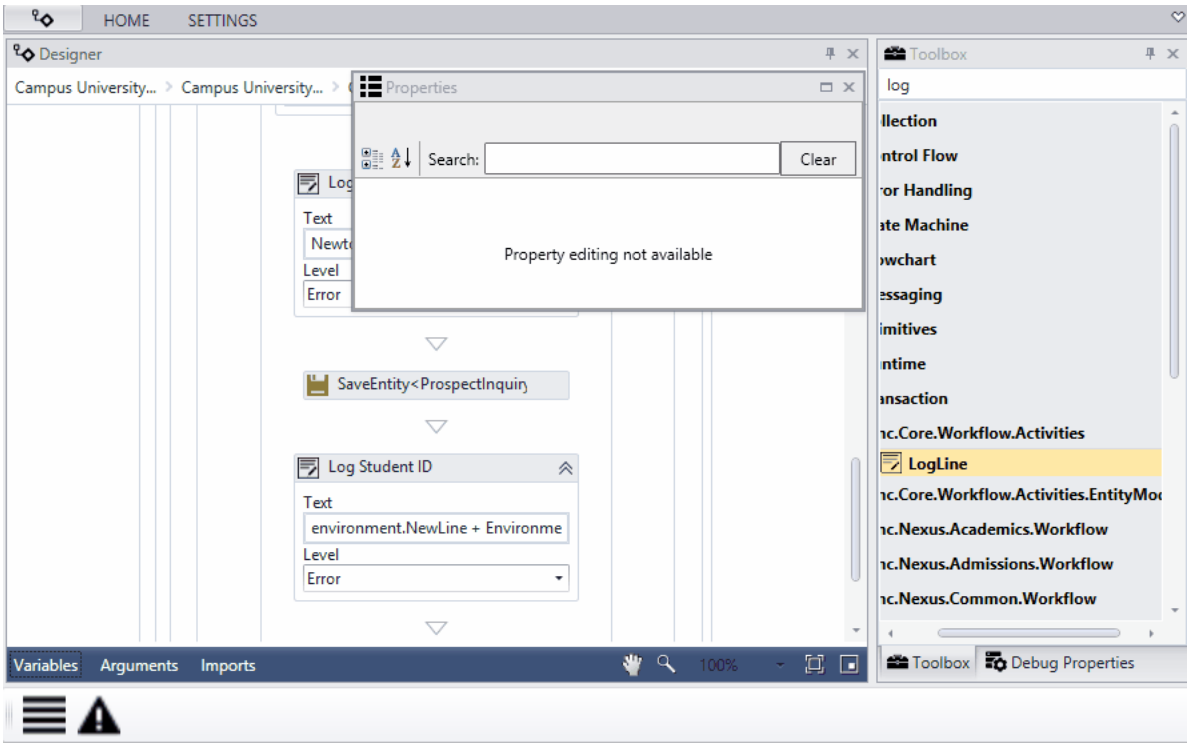


13. The next section of the workflow creates a Portal account for the prospective student. The workflow activities used to create the Portal account require several variables.

    Select the **Variables** tab below the Designer pane and create variables. Specify the following properties:

Name	Variable type	Scope	Default
UserID	String	Campus University RFI	
Role	String	Campus University RFI	“STUD”

Name	Variable type	Scope	Default
Culture	String	Campus University RFI	“en-US”
Key	String	Campus University RFI	"JkL0sPc="
URL	Int32	Campus University RFI	10
WebRoleID	Int32	Campus University RFI	1
Password	String	Campus University RFI	"DAA4496B6AC1B0C8FEEE6BE825854F05E78765F0"



14. Drag a **Sequence** activity below the "Log Student ID" LogLine activity. Rename the sequence to **Create Portal Account**.
15. Drag the following activities into the "Create Portal Account" sequence.

    Assign Activity

a. Display Name: **Assign ID**

b. To field: **UserID**

c. Value: **prospectInquiryEntity.Student.FirstName + "." + prospectInquiryEntity.Student.LastName**

ExecuteQuery Activity

a. Display Name: **Create Account**

b. Connection string name: **"<dbConnections> "** (Enter the connection string displayed in your About Forms Builder window.)

c. Command:**"exec sproc_School_Configuration_Admin_AddUser @LastName = '"& prospectInquiryEntity.Student.LastName & "', @FirstName = '" & prospectInquiryEntity.Student.FirstName &"', @eMail = '" & prospectInquiryEntity.Student.EmailAddress & "', @UserCode = '" & UserID & "', @Pwd = '" & Password & "', @Key = '" & Key & "', @DefaultCulture = '" & Culture & "', @wpURLID = " & URL & ", @CampusID = " & prospectInquiryEntity.CampusId & ", @C2kID = " & prospectInquiryEntity.StudentId & ", @Role = '" & Role & "', @wpWebRoleID = " & WebRoleID & ""**

SendMail Activity

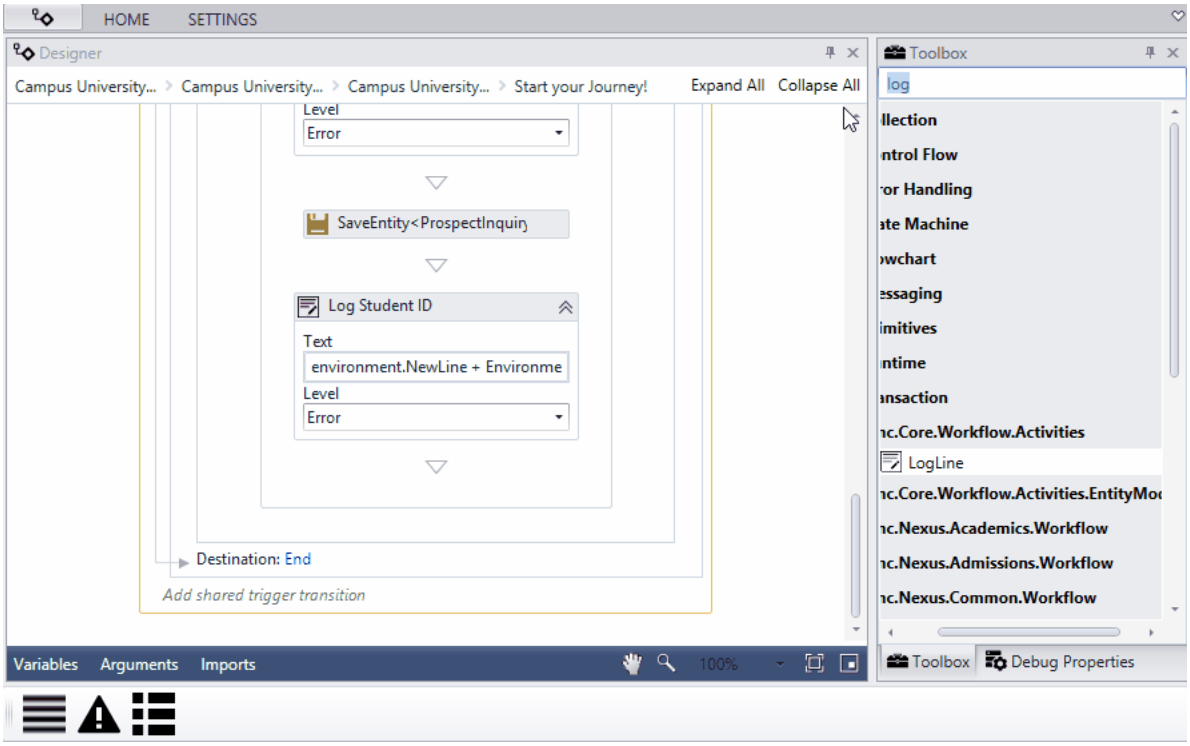
a. Display Name: **SendMail to Student with Portal Credentials**

b. From: **"<campusuniversity@campusmgmt.com>"** (Enter the email address of your institution.)

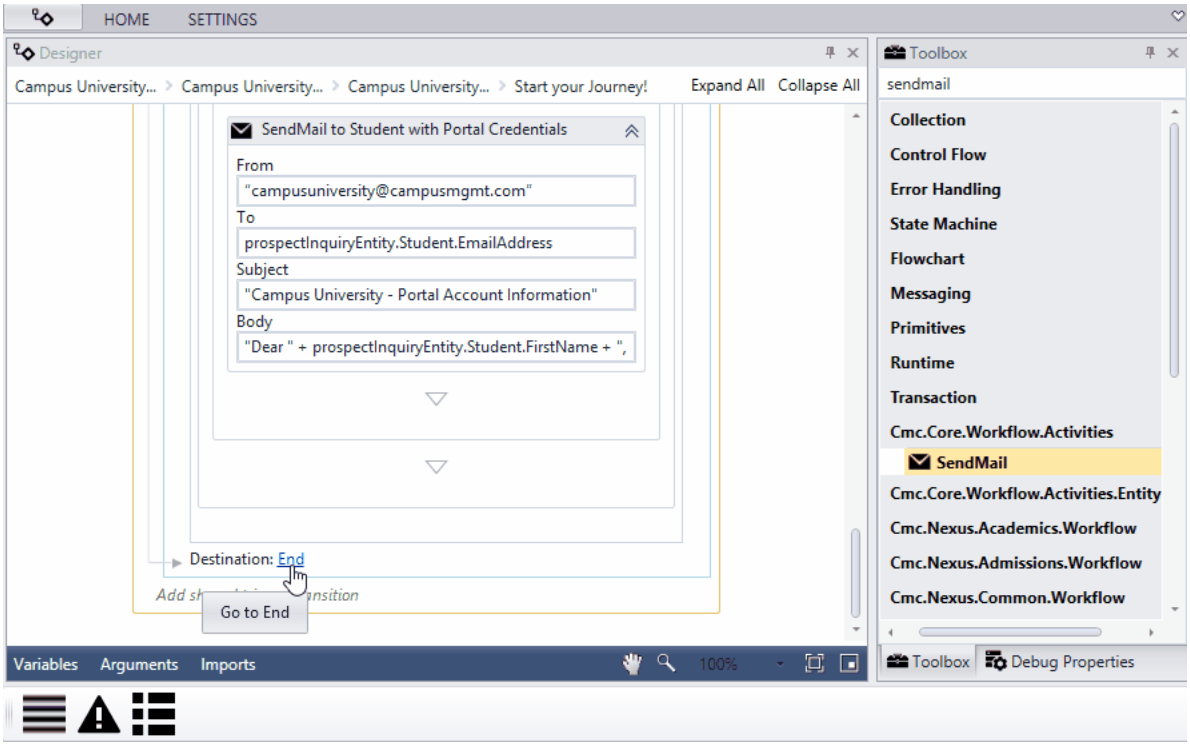
c. To: **prospectInquiryEntity.Student.EmailAddress**

d. Subject: **"Campus University - Portal Account Information"**

e. Body: **"Dear " + prospectInquiryEntity.Student.FirstName + ", " + Environment.NewLine + Environment.NewLine + "Below are your login credentials for the Campus University Portal. You can change your password after you log in." + Environment.NewLine + Environment.NewLine + "Your login Username is: " + UserID + Environment.NewLine + Environment.NewLine + "Your Password is: nexus123" + Environment.NewLine + Environment.NewLine + "Thank you for starting your journey with Campus University!"**



16. Select the **End** link at the bottom of the transition to complete the triggers to occur when the form is submitted.
17. Drag the **SendMail** activity into the Entry section of the End state. Specify the following properties:
    a. Display Name: **SendMail with link to Online Application**
    b. From: **“<campusuniversity@campusmgmt.com>”** (Enter the email address of your institution.)
    c. To:**prospectInquiryEntity.Student.EmailAddress**
    d. Subject: **"Welcome to Campus University"**
    e. Body: **"Dear " + prospectInquiryEntity.Student.FirstName + "," +Environment.NewLine + Environment.NewLine + "Thank you for your interest in Campus University. If you are ready to begin your education journey, go to: http://localhost:<port>/#/renderer/227 to complete the Online Application process." + Environment.NewLine + Environment.NewLine + "Thank you, " + Environment.NewLine + "The Admissions Team"**
18. Confirm there are no errors in the **Error** panel.
19. Select **Publish** and select **Enable This Workflow Version?** Select **OK** to confirm.



Submit the RFI Form

1. Open a different browser and point to the URL for the **Sequence List**.

   (`http://<server>:<port>/#/sequencelist` or `https://<server>:<port>/#/sequencelist`)

2. Use the search box above the Sequence Name column to find the **Campus University - Request for Information** sequence.

3. Scroll to the right and select  to view the sequence.

4. Select **Next** on the Welcome Screen.

5. Complete the RFI form using your student data.

   For the student information, you might want to use data obtained from a random name generator.

   Select these values for the following fields:

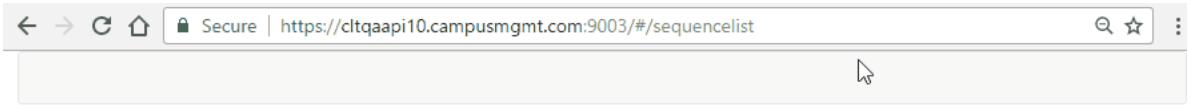
   a. Campus: **<Select a Campus>**

   b. Email: **<A valid email for you>**

   c. Phone Number/Mobile Number: **<Enter the same number for both**

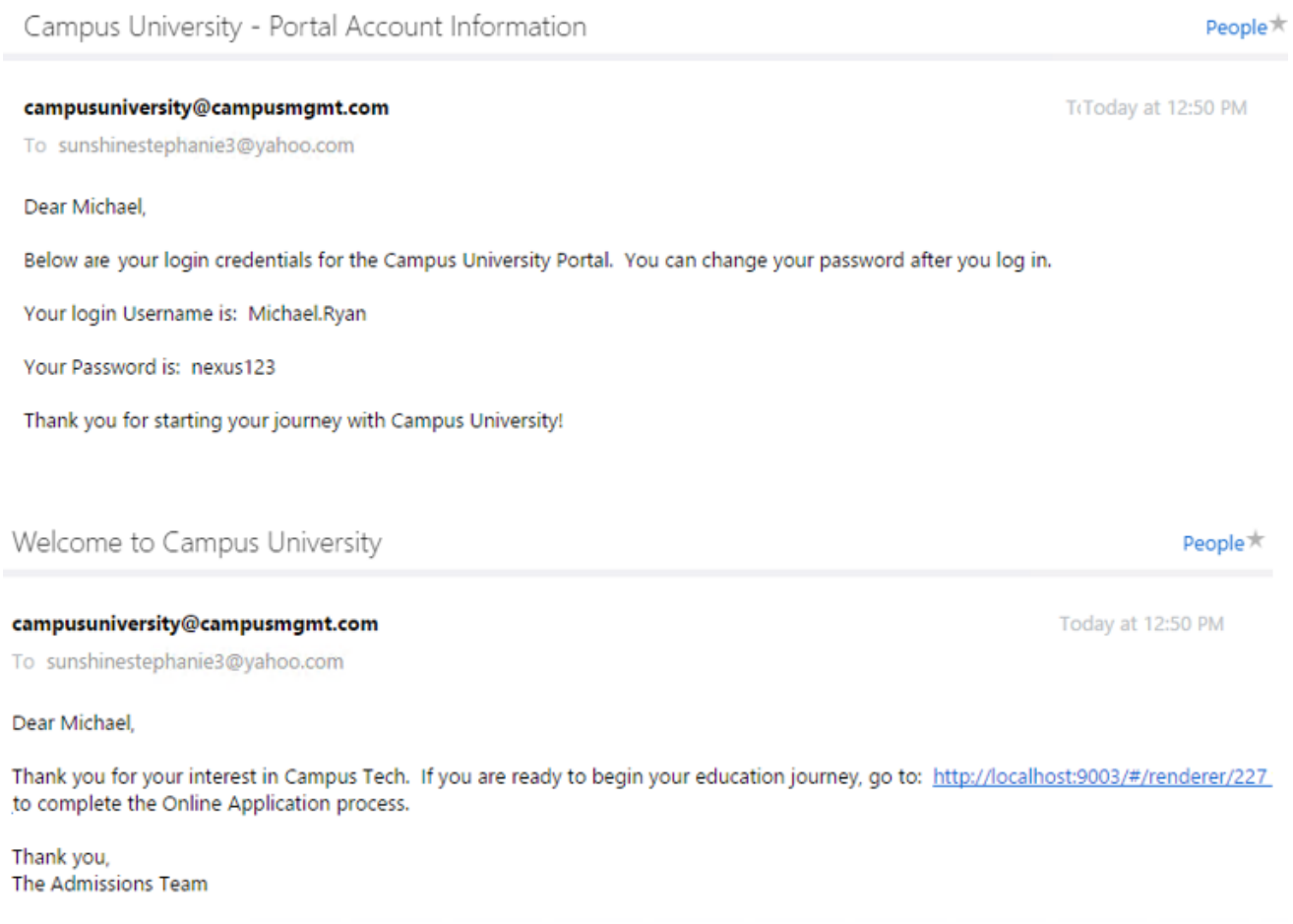
   d. Program: **<Select a program>**

6. Select **Start Your Journey!**
7. Confirm that you received a Confirmation Message.



Validate the Data in the Web Client

1. Open the Web Client for Anthology Student in another browser.
2. Sign in using your user name and password.
3. Select **Students** from the Home Page.
4. In the **Search** bar, search for your student's last name.
5. Select on the **Name** link for your student to access the student's information.
6. Validate that you received the following emails.



Campus University - Portal Account Information

People

campusuniversity@campusmgmt.com

Today at 12:50 PM

To sunshinestephanie3@yahoo.com

Dear Michael,

Below are your login credentials for the Campus University Portal. You can change your password after you log in.

Your login Username is: Michael.Ryan

Your Password is: nexus123

Thank you for starting your journey with Campus University!

Welcome to Campus University

People

campusuniversity@campusmgmt.com

Today at 12:50 PM

To sunshinestephanie3@yahoo.com

Dear Michael,

Thank you for your interest in Campus Tech. If you are ready to begin your education journey, go to: http://localhost:9003/#/renderer/227 to complete the Online Application process.

Thank you,
The Admissions Team

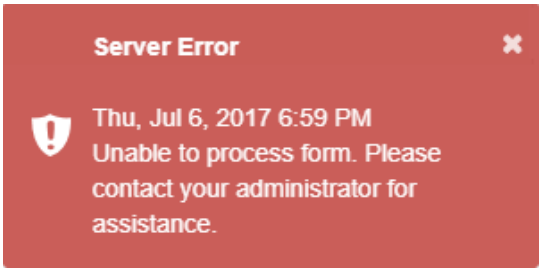
Check the Renderer Log

1. Navigate to: **\\<server>\c$\logs** and open the FormsBuilderRenderer file that has today's date.
2. Navigate to the bottom of the file (Ctrl + End) and then scroll up until you find the log lines from your workflow.

Troubleshooting

Since multiple applications and processes are involved in building forms, it is necessary to apply systematic troubleshooting techniques. This section contains general guidelines and reference materials that may help you to isolate and correct errors.

When a server-side error occurs during the processing of a rendered sequence that cannot be corrected via form resubmission, an error message is displayed. The default message text is *"Unable to process form. Please contact your administrator for assistance."*



You can edit the message text on the Settings tile in Form Designer and save your custom message. You can use HTML markup to encode a URL or email address if desired.

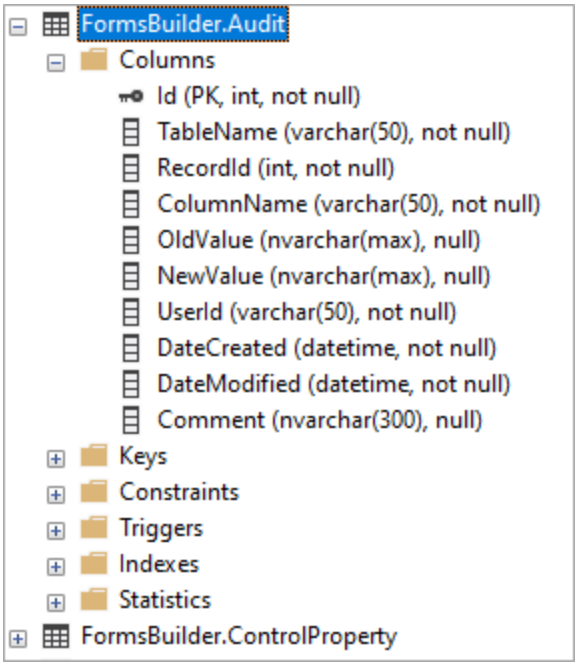
By design, the error details will only be captured in the log files. You need to check the log files to troubleshoot the error. For more information, refer to Log Files.

Audits

The database for Forms Builder 3.7 and later provides a **FormsBuilder.Audit** table that logs Form Designer changes related to:

- Forms
- Sequences
- Settings
- Imports

The FormsBuilder.Audit table has the following columns:



To query the audit table in MS SQL Server Management Studio, you can use a Select statements such as:

select * from formsbuilder.audit order by datemodified desc

select * from formsbuilder.audit where tablename='form' order by datemodified desc

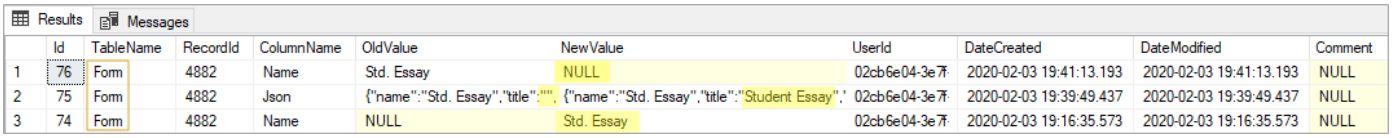
select * from formsbuilder.audit where tablename='sequence' order by datemodified desc

select * from formsbuilder.audit where tablename='setting' order by datemodified desc

select * from formsbuilder.audit where tablename='translation' order by datemodified desc

Forms - Add, Edit, Delete

Records with *TableName = Form* contain details about changes to forms.



	Id	TableName	RecordId	ColumnName	OldValue	NewValue	UserId	DateCreated	DateModified	Comment
1	76	Form	4882	Name	Std. Essay	NULL	02cb6e04-3e7f-	2020-02-03 19:41:13.193	2020-02-03 19:41:13.193	NULL
2	75	Form	4882	Json	{"name":"Std. Essay","title":"",	{"name":"Std. Essay","title":"Student Essay","'	02cb6e04-3e7f-	2020-02-03 19:39:49.437	2020-02-03 19:39:49.437	NULL
3	74	Form	4882	Name	NULL	Std. Essay	02cb6e04-3e7f-	2020-02-03 19:16:35.573	2020-02-03 19:16:35.573	NULL

In row 3, a form named "Std. Essay" is added (OldValue is NULL).

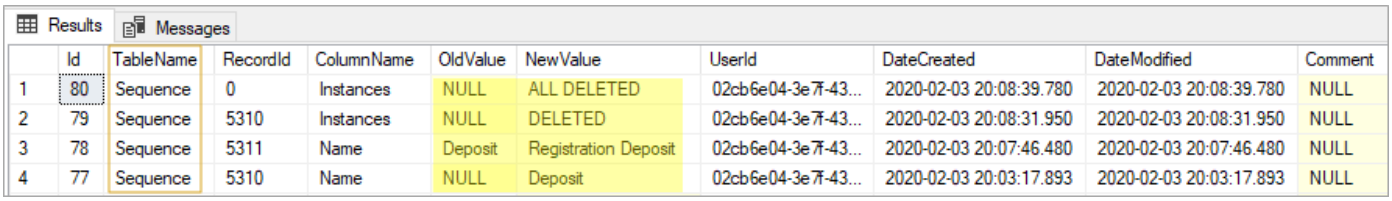
In row 2, the title is changed to "Student Essay".

In row 1, the form is deleted (NewValue is NULL).

Changes to Form Sections are not audited.

Sequences - Add, Edit, Delete

Records with *TableName* = *Sequence* contain details about changes to sequences.



	Id	TableName	RecordId	ColumnName	OldValue	NewValue	UserId	DateCreated	DateModified	Comment
1	80	Sequence	0	Instances	NULL	ALL DELETED	02cb6e04-3e7f-43...	2020-02-03 20:08:39.780	2020-02-03 20:08:39.780	NULL
2	79	Sequence	5310	Instances	NULL	DELETED	02cb6e04-3e7f-43...	2020-02-03 20:08:31.950	2020-02-03 20:08:31.950	NULL
3	78	Sequence	5311	Name	Deposit	Registration Deposit	02cb6e04-3e7f-43...	2020-02-03 20:07:46.480	2020-02-03 20:07:46.480	NULL
4	77	Sequence	5310	Name	NULL	Deposit	02cb6e04-3e7f-43...	2020-02-03 20:03:17.893	2020-02-03 20:03:17.893	NULL

In row 4, a sequence named "Deposit" is added.

In row 3, the sequence is renamed "Registration Deposit".

In row 2, the persisted instances of the sequence are deleted.

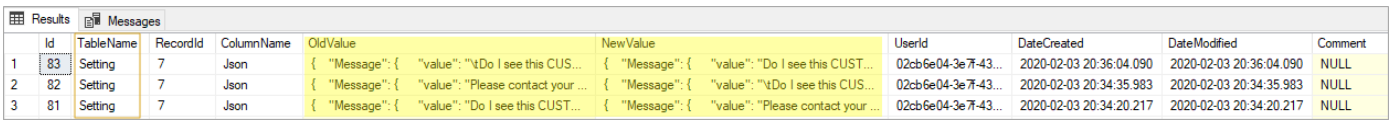
In row 1, all instances of the sequence are deleted.

Changes to the following sequence properties are **not** audited:

- Analytics Tracker
- Footer
- Header
- Nav Button Position
- Themes
- Title

Settings - Edit

Records with *TableName* = *Setting* contain details about changes to the Forms Builder settings.



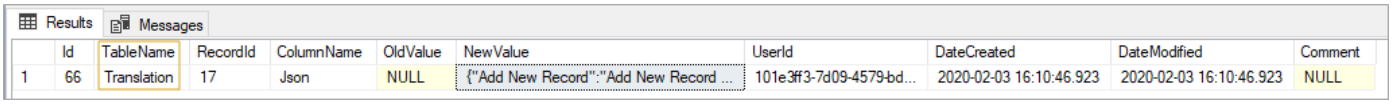
	Id	TableName	RecordId	ColumnName	OldValue	NewValue	UserId	DateCreated	DateModified	Comment
1	83	Setting	7	Json	{ "Message": { "value": "\tDo I see this CUS...	{ "Message": { "value": "Do I see this CUST...	02cb6e04-3e7f-43...	2020-02-03 20:36:04.090	2020-02-03 20:36:04.090	NULL
2	82	Setting	7	Json	{ "Message": { "value": "Please contact your ...	{ "Message": { "value": "\tDo I see this CUS...	02cb6e04-3e7f-43...	2020-02-03 20:34:35.983	2020-02-03 20:34:35.983	NULL
3	81	Setting	7	Json	{ "Message": { "value": "Do I see this CUST...	{ "Message": { "value": "Please contact your ...	02cb6e04-3e7f-43...	2020-02-03 20:34:20.217	2020-02-03 20:34:20.217	NULL

In rows 1, 2, and 3 a Setting value is modified.

All changes to values in the Forms Builder Settings workspace are audited. Users cannot add or removed settings.

Imports

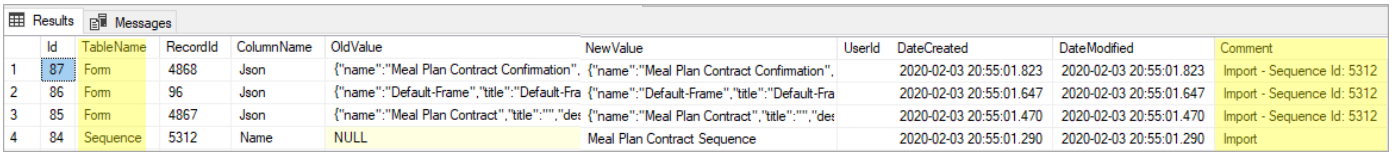
Records with *TableName = Translation* contain details about imported PO files.



Results | Messages

	Id	TableName	RecordId	ColumnName	OldValue	NewValue	UserId	DateCreated	DateModified	Comment
1	66	Translation	17	Json	NULL	{"Add New Record":"Add New Record ...	101e3ff3-7d09-4579-bd...	2020-02-03 16:10:46.923	2020-02-03 16:10:46.923	NULL

Records with *Comment = Import* and *Comment = Import - Sequence* contain details about an imported sequence and associated forms.



Results | Messages

	Id	TableName	RecordId	ColumnName	OldValue	NewValue	UserId	DateCreated	DateModified	Comment
1	87	Form	4868	Json	{"name":"Meal Plan Contract Confirmation",	{"name":"Meal Plan Contract Confirmation",		2020-02-03 20:55:01.823	2020-02-03 20:55:01.823	Import - Sequence Id: 5312
2	86	Form	96	Json	{"name":"Default-Frame","title":"Default-Fra	{"name":"Default-Frame","title":"Default-Fra		2020-02-03 20:55:01.647	2020-02-03 20:55:01.647	Import - Sequence Id: 5312
3	85	Form	4867	Json	{"name":"Meal Plan Contract","title":"","des	{"name":"Meal Plan Contract","title":"","des		2020-02-03 20:55:01.470	2020-02-03 20:55:01.470	Import - Sequence Id: 5312
4	84	Sequence	5312	Name	NULL	Meal Plan Contract Sequence		2020-02-03 20:55:01.290	2020-02-03 20:55:01.290	Import

Azure Blob Storage

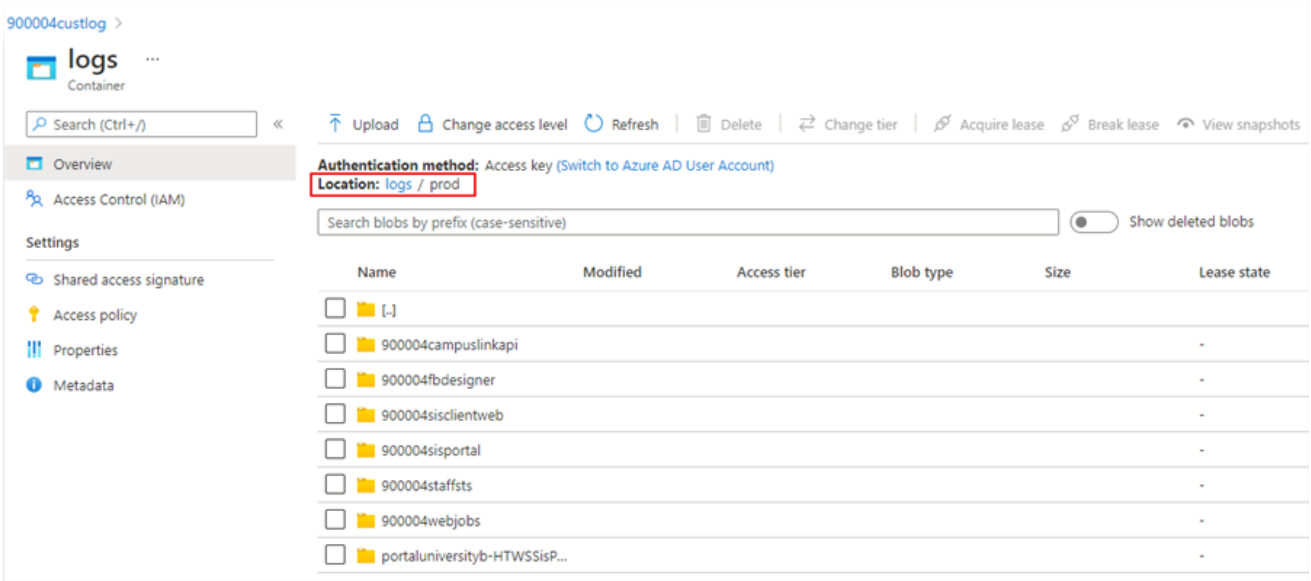
Forms Builder 3.7 adds logging capabilities for Designer and Renderer log files to the Azure blob storage. The Azure blob storage container also provides logs for other products included in your subscription, e.g., Anthology Student, Portal, Workflow Events, STS, CampusLink APIs, etc.

Azure blob storage is Microsoft's object storage solution for the cloud. Blob storage is optimized for storing massive amounts of unstructured data. Unstructured data is data that doesn't adhere to a particular data model or definition, such as text or binary data. Refer to https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction.

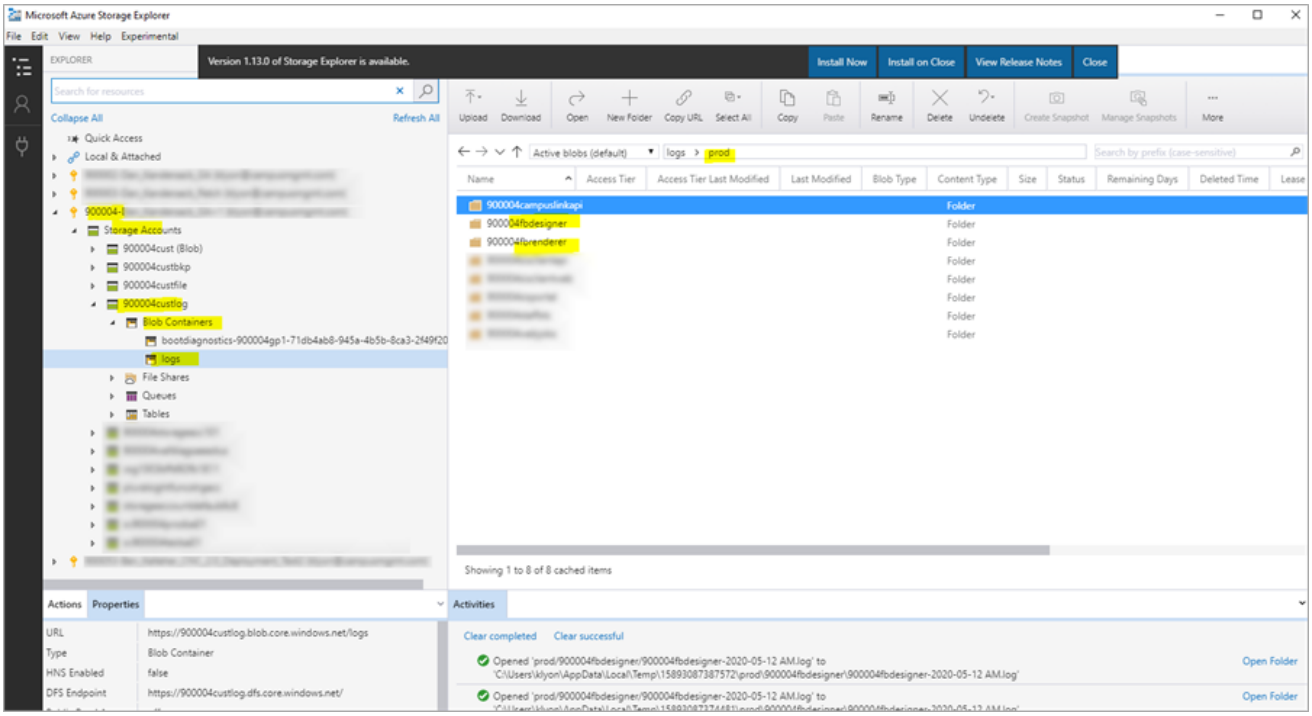
To find Azure blob storage logs for your cloud subscription:

1. Log in to the **Microsoft Azure Storage Explorer**.
2. Select your **subscription** (900004 in our example).
3. In the panel on the left side, navigate to **Storage Accounts** > **<your subscription>custlog** > **Blob Containers** > **logs**.
4. In the panel on the right side, open the **prod** directory.

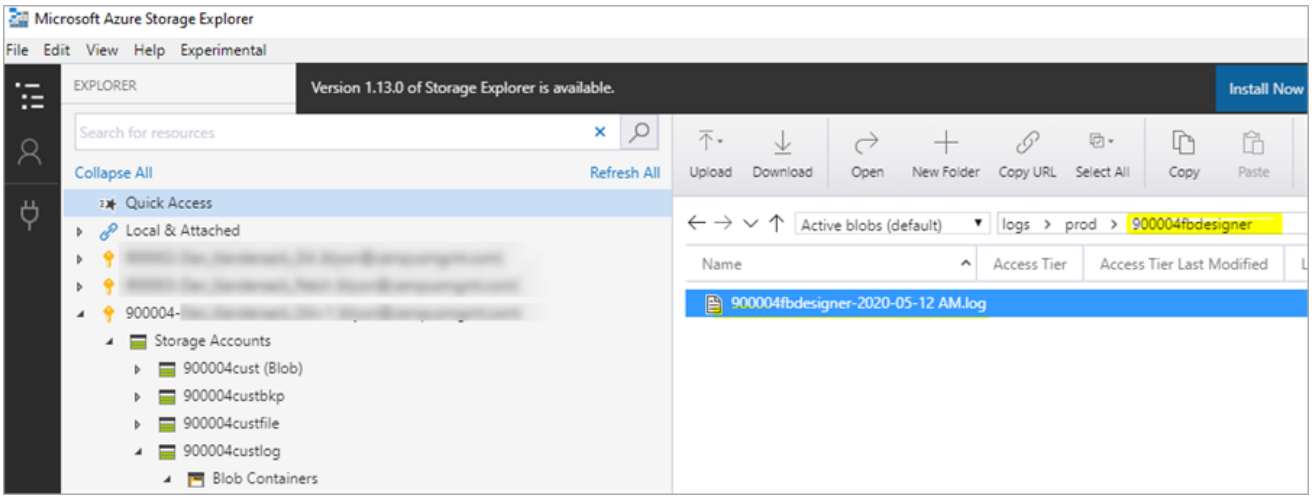
   Depending on your subscription, you will have folders for multiple products, e.g., CampusLinkAPI, Anthology Student (sisclientweb), Portal (sisportal), Form Designer, Forms Renderer, Staff STS, Web Jobs, etc. Select the folder for a product and navigate to the log you want to review. These folders contain the last 2 weeks of logs.



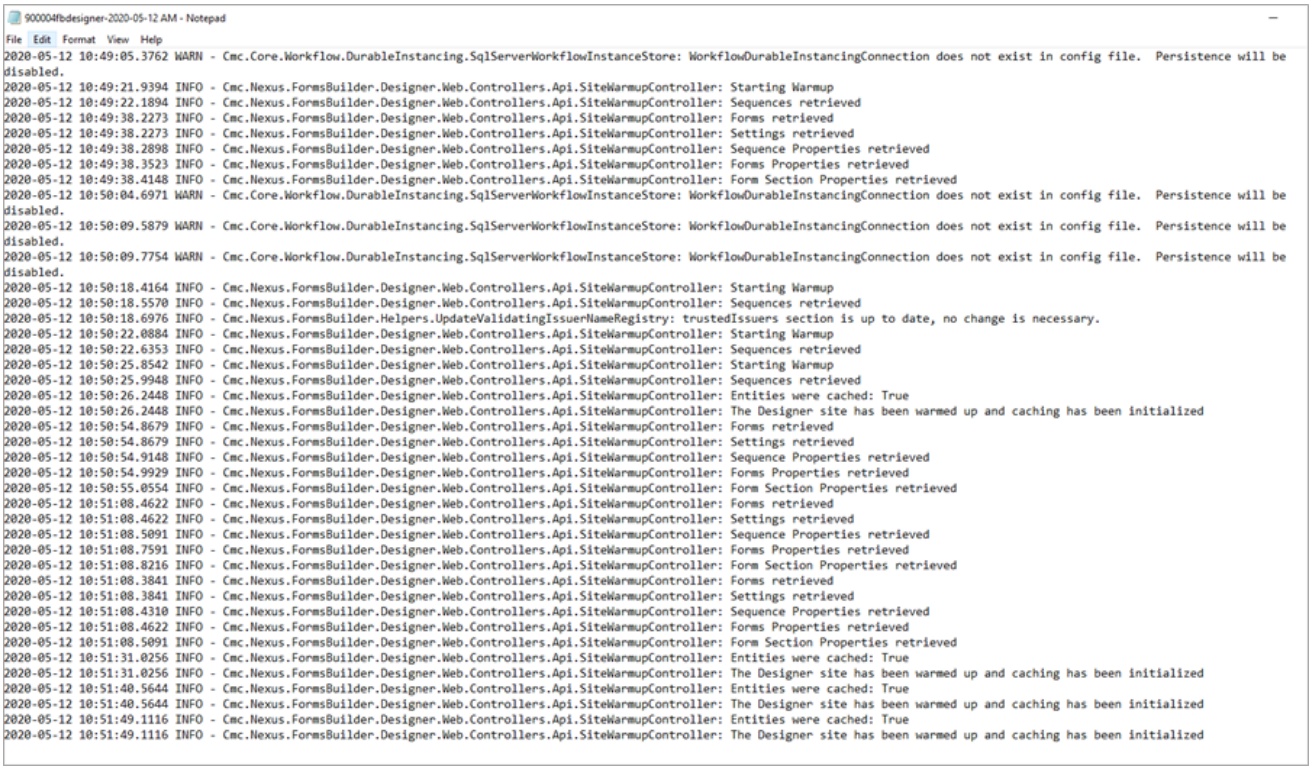
5. For Forms Builder logs, find the folders **<your subscription>fbdesigner** and **<your subscription>fbrenderer**.



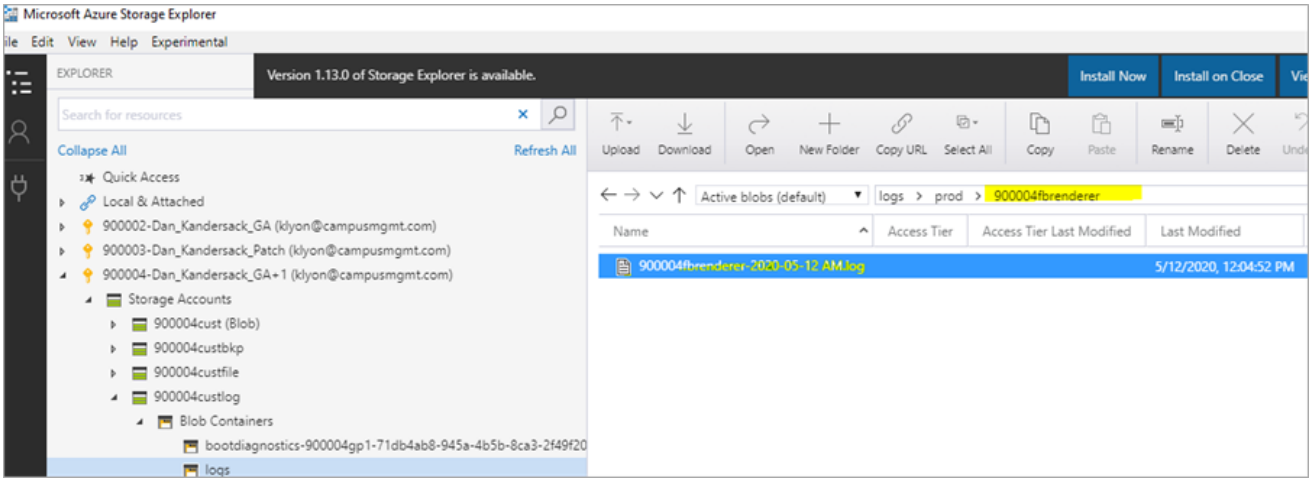
6. For Form Designer logs, select the **<your subscription>fbdesigner** folder and view or download the log.



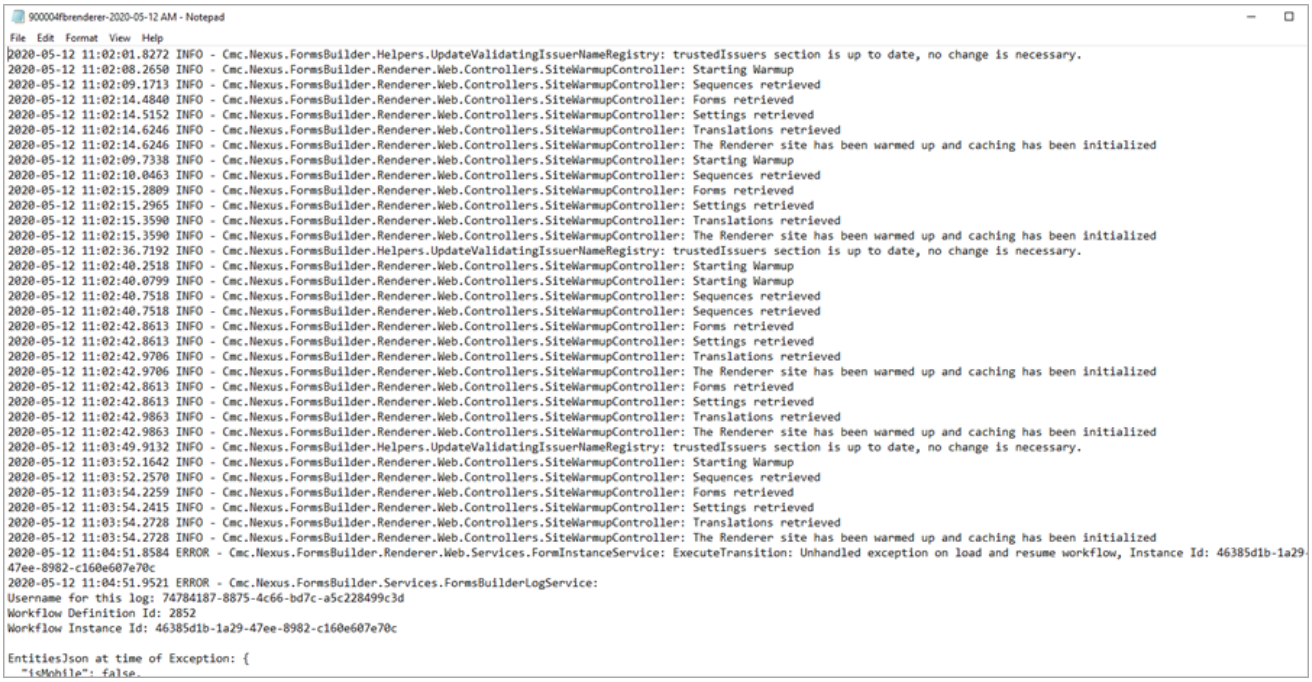
The <your subscription>fbdesigner log is displayed in Notepad.



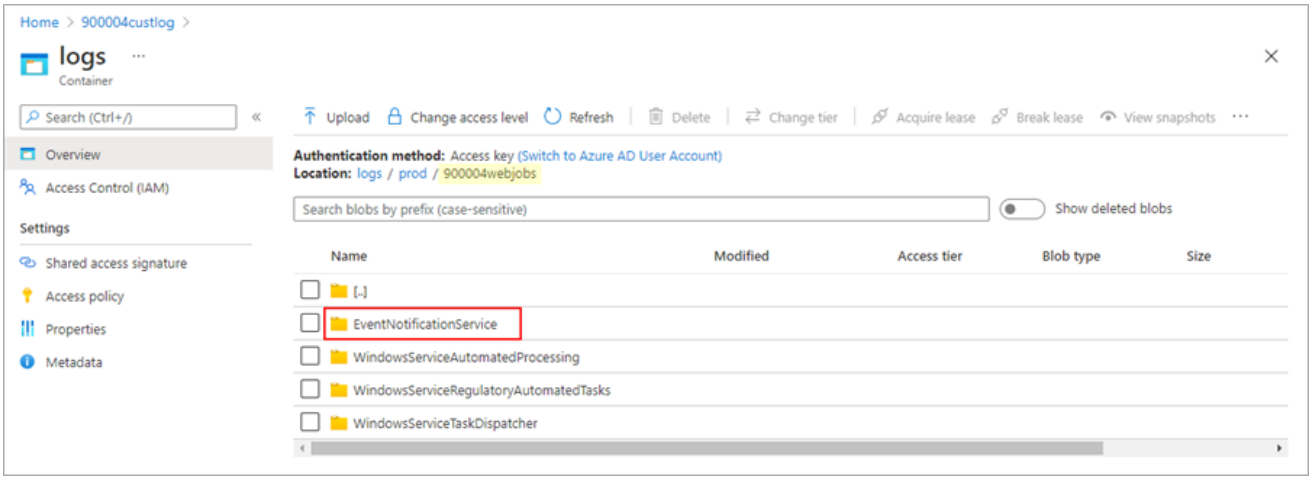
7. For Forms Renderer, select the **<your subscription>fbrenderer** folder and view or download the log.



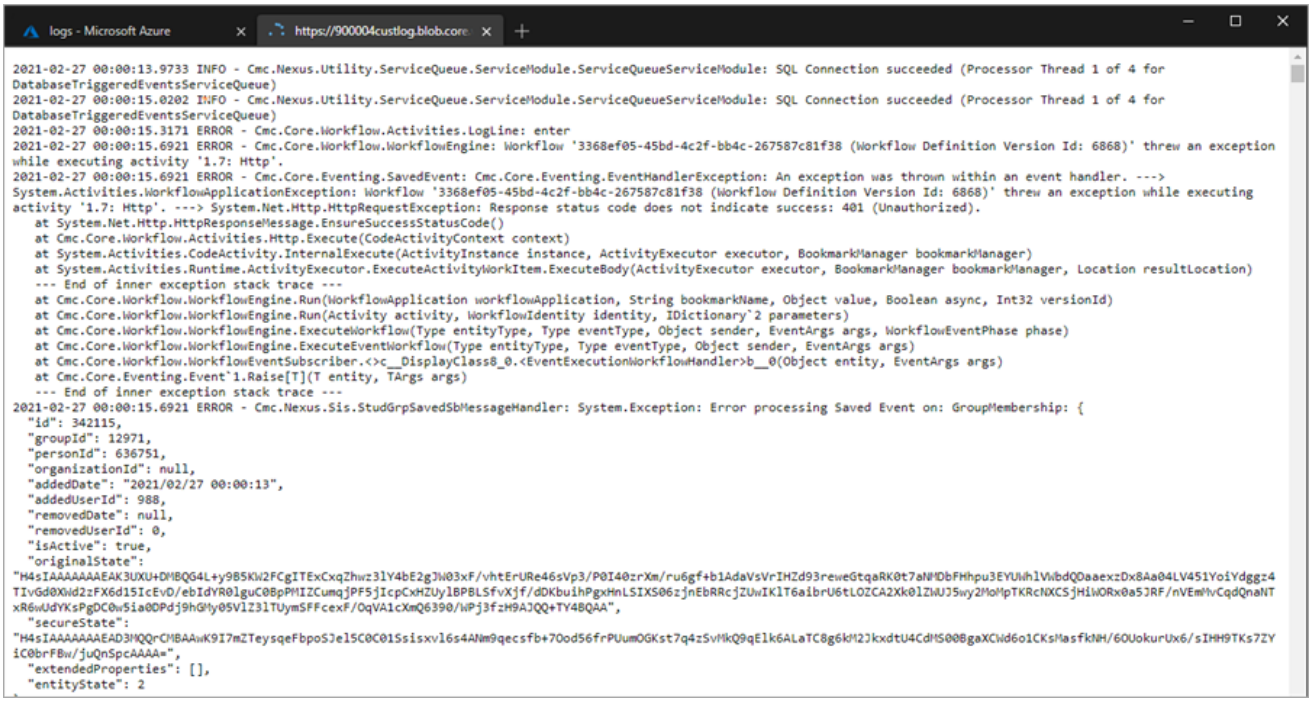
The <your subscription>fbrenderer log is displayed in Notepad.



8. For Workflow Composer logs, select the **<your subscription>webjobs** folder and then select **EventNotificationService**.



View or download the workflow log you want to review.




```
2021-02-27 00:00:13.9733 INFO - Cmc.Nexus.Utility.ServiceQueue.ServiceModule.ServiceQueueServiceModule: SQL Connection succeeded (Processor Thread 1 of 4 for
DatabaseTriggeredEventsServiceQueue)
2021-02-27 00:00:15.0202 INFO - Cmc.Nexus.Utility.ServiceQueue.ServiceModule.ServiceQueueServiceModule: SQL Connection succeeded (Processor Thread 1 of 4 for
DatabaseTriggeredEventsServiceQueue)
2021-02-27 00:00:15.3171 ERROR - Cmc.Core.Workflow.Activities.LogLine: enter
2021-02-27 00:00:15.6921 ERROR - Cmc.Core.Workflow.WorkflowEngine: Workflow '3368ef05-45bd-4c2f-bb4c-267587c81f38 (Workflow Definition Version Id: 6868)' threw an exception
while executing activity '1.7: Http'.
2021-02-27 00:00:15.6921 ERROR - Cmc.Core.Eventing.SavedEvent: Cmc.Core.Eventing.EventHandlerException: An exception was thrown within an event handler. --->
System.Activities.WorkflowApplicationException: Workflow '3368ef05-45bd-4c2f-bb4c-267587c81f38 (Workflow Definition Version Id: 6868)' threw an exception while executing
activity '1.7: Http'. ---> System.Net.Http.HttpRequestException: Response status code does not indicate success: 401 (Unauthorized).
   at System.Net.Http.HttpResponseMessage.EnsureSuccessStatusCode()
   at Cmc.Core.Workflow.Activities.Http.Execute(CodeActivityContext context)
   at System.Activities.CodeActivity.InternalExecute(ActivityInstance instance, ActivityExecutor executor, BookmarkManager bookmarkManager)
   at System.Activities.Runtime.ActivityExecutor.ExecuteActivityWorkItem.ExecuteBody(ActivityExecutor executor, BookmarkManager bookmarkManager, Location resultLocation)
   --- End of inner exception stack trace ---
   at Cmc.Core.Workflow.WorkflowEngine.Run(WorkflowApplication workflowApplication, String bookmarkName, Object value, Boolean async, Int32 versionId)
   at Cmc.Core.Workflow.WorkflowEngine.Run(Activity activity, WorkflowIdentity identity, IDictionary`2 parameters)
   at Cmc.Core.Workflow.WorkflowEngine.ExecuteWorkflow(Type entityType, Type eventType, Object sender, EventArgs args, WorkflowEventPhase phase)
   at Cmc.Core.Workflow.WorkflowEngine.ExecuteEventWorkflow(Type entityType, Type eventType, Object sender, EventArgs args)
   at Cmc.Core.Workflow.WorkflowEventSubscriber.<>c__DisplayClass8_0.<EventExecutionWorkflowHandler>b__0(Object entity, EventArgs args)
   at Cmc.Core.Eventing.Event`1.Raise[T](T entity, TArgs args)
   --- End of inner exception stack trace ---
2021-02-27 00:00:15.6921 ERROR - Cmc.Nexus.Sis.StudGrpSavedSbMessageHandler: System.Exception: Error processing Saved Event on: GroupMembership: {
  "id": 342115,
  "groupId": 12971,
  "personId": 636751,
  "organizationId": null,
  "addedDate": "2021/02/27 00:00:13",
  "addedUserId": 988,
  "removedDate": null,
  "removedUserId": 0,
  "isActive": true,
  "originalState":
"H4sIAAAAAAAAEAK3UXU+DMBQG4L+y9B5KW2FCgITExCxqZhwz3lY4bE2gJW03xF/vhtErURe46sVp3/P0I40zrXm/ru6gf+b1AdaVsVrIHZd93reweGtqaRK0t7aNMDbFHhpu3EYUWhlVWbdQDaaaexzDx8Aa04LV451YoiYdggz4
TIvGd0XWd2zFX6d15IcEvD/ebIdYR0lguC0BpPMIZCumqjPF5jIcpCxHZUylBPBLSfvXjf/dDKbuihPgxHnLSIXS06zjnEbRRcjZUwIKlT6aibrU6tLOZCA2Xk0lZWUJ5wy2MoMpTKRcNXCSjHiWORx0a5JRF/nVEmMvCqdQnaNT
xR6wUdYKsPgDC0w5ia0DPdj9hGMy05VlZ3lTUymSFFcexF/OqVA1cXmQ6390/WPj3fzH9AJQQ+TY4BQAA",
  "secureState":
"H4sIAAAAAAAAEAD3MQQrCMBAAwK9I7mZTeysqeFbpoSJel5C0C01Ssisxvl6s4ANm9qecsfb+7Ood56frPUumOGKst7q4zSvMkQ9qElk6ALaTC8g6kM2JkxdtU4CdMS00BgaXCWd6o1CKsMasfkNH/6OUokurUx6/sIHH9TKs7ZY
iC0brFBw/juQnSpcAAAA=",
  "extendedProperties": [],
  "entityState": 2
```

Basics

The following guidelines for software troubleshooting also apply to building forms with Forms Builder and creating workflows in Workflow Composer.

- **One change at a time**

  If you have a bug, try a fix, render the form, and check if it's fixed. If you have two bugs, fix one, test, and then fix the other bug. Don't try to fix all your bugs at once. Build and test incrementally to isolate the source of a problem.

- **Read the error messages**

  Error messages can be intimidating, but sometimes you find a nugget of helpful information.

- **Read your code** (or have someone else read it for you)

  Often, when you read your own code, you read it as you meant to write it, not as it is actually written. It can be hard to catch little things like off-by-one errors. If you take some time away from your code, then read it fresh, or have someone else read, you might find your code doesn't do exactly what you thought it did.

- **Correct the problems you know of before tackling unknown errors**

  When confronted with many errors, you might spot 1-2 that you know the cause of. It's tempting to leave those for later and tackle the unknown errors first. But sometimes the unknown errors are byproducts of the known errors. Or maybe the known errors make the unknown errors more complicated or confusing.

- **Search for bugs using as many tools as possible, and approach from all angles**

  Embrace all available technologies. In the case of forms and workflows, these tools include log files, browser developer tools, and

- **Sprinkle breakpoints**

  Mark up the code in places of interest, check what entry points get hit, or how certain values change. In the case of workflows, LogLine or LogObject activities can serve as breakpoints for successful workflow execution. We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

Developer Tools

To access the browser developer tool, right-click somewhere on the page and select **Inspect Element** in the context-menu or press **F12**. When opened, the default is to embed the development tool in the browser. It can be separated from the browser and viewed in its own window.

For additional information on how to open developer tool in different browsers, refer to http://webmasters.stackexchange.com/questions/8525/how-to-open-the-javascript-console-in-different-browsers.

For details about the Microsoft Edge F12 Dev Tools, refer to https://developer.microsoft.com/en-us/microsoft-edge/platform/documentation/f12-devtools-guide/.

Console

Web browsers provide a JavaScript console as part of their developer tools. This console is useful for the following reasons:

- Errors and warnings that occur on a webpage are logged into the console
- JavaScript commands for interacting with a webpage can be executed in the console

For more information, refer to http://blog.teamtreehouse.com/mastering-developer-tools-console.

Using the Console to Find Syntax and Other Code Errors

In most coding projects, errors usually consist of syntax, logical, or data input errors. The console view shows JavaScript errors and exceptions, as well as Document Object Model (DOM) exceptions. From inside your code, you can use the console object to send status and program error messages to the console. For example, you can add a line like

```
JavaScript
window.console.log("The file opened successfully");
```

to your JavaScript code to get the status in a script without breaking the execution. For more information about using the console, refer to Using the F12 Tools Console to View Errors and Status.

F12 Developer Tools Console error messages

Refer to https://msdn.microsoft.com/en-us/library/hh180764(v=vs.85).aspx.

DOM Explorer (IE) / Elements (Chrome)

This tool shows the structure of your webpage as it is being rendered in the browser and makes it possible to edit your HTML and styles (CSS) in a live page. You can do this without having to edit and reload your sources, so you can quickly solve display issues.

- Determine why an element is not displaying at the right place or right size.
- Figure out which CSS styles and media queries are being applied to an element.

- Test a series of different colors for an element to determine which looks best.

Fiddler

Fiddler is a free packet analyzer that captures HTTP and HTTPS traffic data between browsers and servers. It can be used for network troubleshooting, session analysis, and debugging. The tool visualizes request or response messages and data exchanged between client and server.

For more information, refer to http://www.telerik.com/fiddler.

Field Length Changes Impacting Forms Builder

Forms Builder has been tested for compatibility with Student Database regarding field length changes.

Steps to Modify the Field Length

If your form contains the following fields, then follow the steps mentioned below:

1. Log in to the Forms Builder Designer.
2. Navigate to open the form(s) which contains the fields with increased length
3. Perform either of the steps mentioned below:
    - Drag and drop the same field in the form and delete the existing field OR
    - Verify the Max Length of the field and update the Max Length property in the existing field in the form
4. Save the form and try to render the sequence again.

List of Field Length that Affects Forms Builder

A list of field length changes that affect Forms Builder is mentioned below:

Table Name	Entity/Label name	Column	Current Length	Increased Length
AdAttendanceWork		CDescrip	60	150
AdAttendanceWork		CourseCode_Section	25	50
AdClassSched	ClassSection	Code	12	50
AdClassSched	ClassSection	Descrip	75	150
AdClassSched_Proxy		ClassSchedSectionCode	75	200
AdClassSched_Proxy		CourseCode	75	150

AdCourse	Course	CatalogCode	10	50
AdCourse	Course	Code	12	50
AdCourse	Course	Descrip	75	150
AdDegreePath-wayTemplate	DegreePathwayTemplate	AdDegreePath-wayDescription	100	150
AdEnrollSched	StudentCourse	Descrip	75	150
AdEnrollSched	StudentCourse	TransferredCourseCode	16	50
AdEnrollSched	StudentCourse	Trans-ferredCourseDescrip	75	150
AdEn-rollSchedStatusChanges	StudentCourseStatusChange	Descrip	75	150
AdEn-rollSchedStatusHistory	StudentCourseStatusHistoryChange	Descrip	75	150
AmCollegeCourse	CollegeCourse	CollegeCourseCode	16	50
AmCollegeCourse	CollegeCourse	CollegeCourseDescrip	75	150
AmCollegeTransfer	StudentTransferCredit	CollegeCourseCode	16	50
AmCollegeTransfer	StudentTransferCredit	CollegeCourseDescrip	75	150
AmCollegeTransfer_ Proxy		NewCollegeCourseCode	16	50
AmCollegeTransfer_ Proxy		NewCol-legeCourseDescrip	75	150
AmCollegeTransfer_ Proxy		OldCollegeCourseCode	16	50
AmCollegeTransfer_ Proxy		OldCol-legeCourseDescrip	75	150
AmHighSchoolCourse	HighSchoolCourse	CourseCode	16	50
AmHighSchoolCourse	HighSchoolCourse	CourseDescrip	75	150
si_CourseSec-tionExtractHistory		Code	12	50
SyAudit_AdCatalogYear	CatalogYearDataChange	NewValue	200	250
SyAudit_AdCatalogYear	CatalogYearDataChange	OldValue	200	250
SyAudit_AdClassSched	ClassSectionDataChange	CourseCode	75	150
SaTransAdjustDetail	Stu-dentAc-countTransactionAdjustmentDetail	PriorValue	100	150

SaTransAdjustDetail	StudentAccountTransactionAdjustmentDetail	NewValue	100	150
Si_SyAudit_AdInstructorCourseAssociation		CourseCode	20	50
SaPendingCharge	StudentAccountPendingTransaction	Descrip	100	200
SaAutoBillBatchDetail	AutomaticBillingBatchDetail	Description	100	200
SaRevenueDetailRptWork	StudentRevenueDetailReportWork	CourseCode	12	50
SaRevenueSummaryRptWork		CourseCode	12	50
syStudent	Student	email	50	128
syStudent	Student	FirstName	25	128
syStudent	Student	FullName	52	376
syStudent	Student	LastName	25	128
syStudent	Student	OtherEmail	50	128
syStudent	Student	StuNum	10	30

HTTP Status Codes

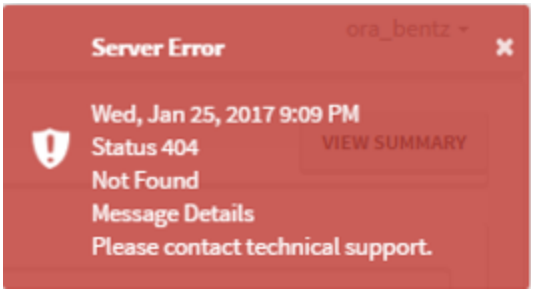
While working in Forms Builder (or any web application), you may encounter error messages that display HTTP status codes. HTTP status codes are divided into two groups: client errors with 4xx status codes and server errors with 5xx status codes.

4xx Client Errors

This group of HTTP status codes indicates that the request for a webpage or other resource contains bad syntax or cannot be filled for some other reason, presumable by fault of the client.

Common client error HTTP status codes are 400 (Bad Request), 401 (Unauthorized), 404 (Not Found), and 408 (Request Timeout).

Example



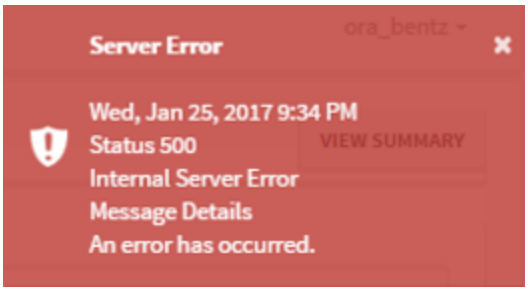
A 404 error indicates that the server itself was found, but that the server was not able to retrieve the requested page. Check the log file for additional information about the cause for this error.

5xx Server Errors

This group of HTTP status codes indicates that the request for a webpage or other resource is understood by the server, but the server is incapable of filling it for some reason.

Common server error HTTP status codes are 500 (Internal Server Error), 502 (Bad Gateway), and 503 (Service Unavailable). **Always check the log to determine if additional information about the 5xx error is available.**

Example



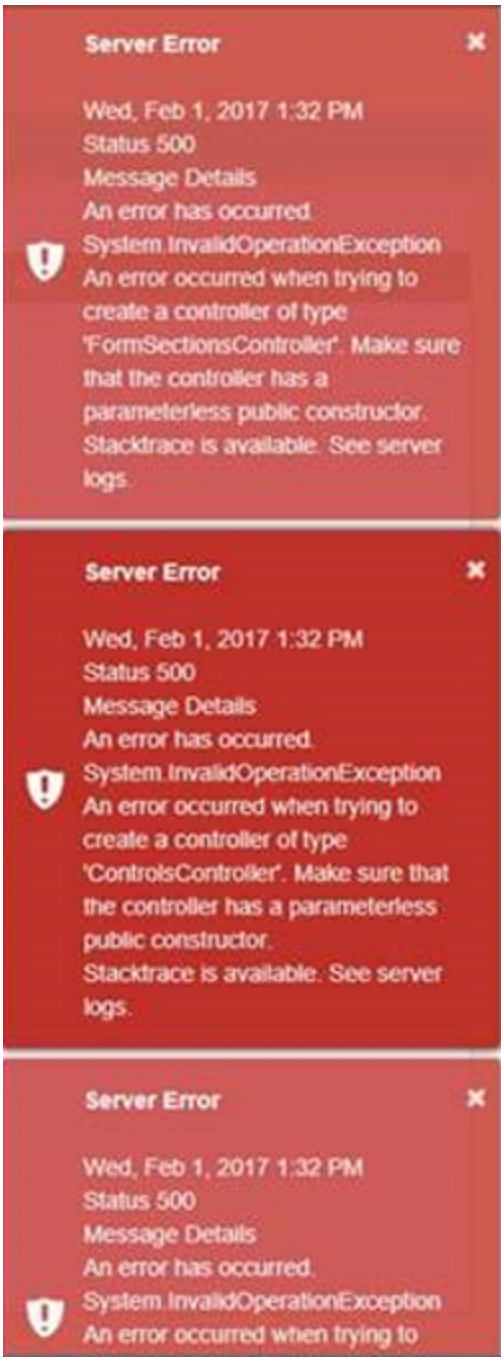
If the error is temporary, reloading the webpage and clearing the cache (Ctrl+F5) may solve the issue. Otherwise, contact your network administrator.

You may encounter a series of 404 and 500 errors during installation or upgrade procedures.

Installation Errors Related to CRM Contracts.dll

When Forms Builder is used with CampusNexus CRM, the **Cmc.NexusCrm.Contracts.dll** from the \bin folder of the Web Client for CampusNexus CRM needs to be copied to the installation folder of Workflow Composer and to the \bin folder of Forms Renderer. If the dll file is also copied to the \bin folder of Form Designer, a series of server errors will be displayed, and Forms Builder will not function.

> ⚠ **Do not copy the Cmc.NexusCrm.Contracts.dll to the \bin folder of Forms Builder Designer.**



For more information, refer to Set Up the Database Environment and CampusNexus CRM Integrations.

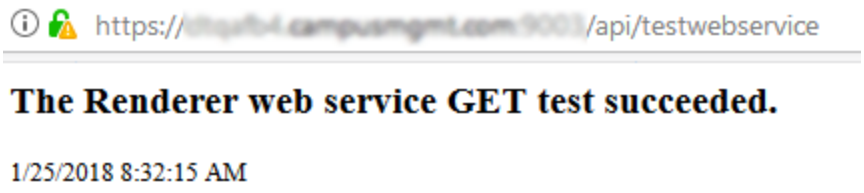
Test Web Services for Designer and Renderer

In Forms Builder 3.4 and later the connectivity of Designer and Renderer can easily be tested using a web service method that was added to each web service. All you need is any browser and type the following URLs. Neither URL is case sensitive. Depending on the configuration of the site http or https can be used.

http://<designersite>:<port>/api/testwebservice

http://<rendersite>:<port>/api/testwebservice

The browser will return text indicating the GET test succeeded and the time of day. The test is not dependent on form or sequence designs, workflows, or CRM/Student database contents.



If this test fails, you know you have some work to do on the installation of the product and the web service. For example, you may find 502 errors in the log indicating issues with the setup and functioning of IIS and the network.

```
1-The remote server returned an unexpected response: (502) Bad Gateway.
2-The remote server returned an error: (502) Bad Gateway.
```

Log Files

The information below does not apply to Forms Builder 3.5.2 and later in an Azure environment. For information about logging and debugging workflows in an Azure environment, refer to Logging in Azure.

Enhanced Logging in Forms Builder 3.4 and Later

Additional logging capability is provided in Forms Builder 3.4 and later:

- WorkflowinstanceId

  The FormBuilderLogService has been enhanced so that whenever it logs the user name and workflowdefinitionid, it now also logs the workflowinstanceid, if available.

- Fatal exceptions

  If a final exception is caught in the FormInstanceController (which essentially aborts the web service call), the exception is now logged as Fatal, since it is not just an error. A Fatal exception essentially terminates a workflow and causes aborts on further attempts to access it.

- Entire DocuSign return envelope

  The WebhookController used in DocuSign sequences has been enhanced to log much more info. This code is executed when DocuSign returns a Completed status for a sequence with multiple signers. The entire return envelope from DocuSign is now logged at the Debug level. There is also more Info available as it starts up the previous workflow.

> **In Forms Builder 3.5.1 and later**, the ability to set NLog levels in the Settings workspace of Form Designer is removed to prevent conflicts with Azure log configurations. Azure logs are stored in customer-specific tables. **If your Forms Builder deployment is in an Azure environment, contact Anthology Inc. obtain access to the Azure log tables or to request changes in the NLog settings.**

Several logger.debug statements and client-side logs have been modified to **Info** level to make them available to help debug issues in an Azure environment since in an Azure environment the log level is set to Info level for all products. The Info level is set for logs related to:

- Site Warmup
- LookupUser
- Account Controller
- PDF creation and e-sign documents
- Payment processing for PayPal, ACI, and IATS

NLog Levels

NLog Level	Hierarchy	Description
Fatal	1 - Only fatal messages will be logged.	A web service call has thrown an exception. The username, workflowdefintionid, and workflowinstanceid are logged.

NLog Level	Hierarchy	Description
Error	2 - Logs include the first two levels.	Any exceptions that are not being handled by the application.
Warn	3 - Logs include the first three levels.	Messages about potential oddities from which the application automatically recovers or about variable/property values that may be close to being out of the acceptable range.
Info	4 - Logs include the first four levels.	Reserved for customer logging in workflows (refer to LogLine/LogObject Activities).
Debug	5 - Logs include the first five levels.	All additional Forms Builder logging.
Trace	6 - Logs include all messages.	Extensive additional Cmc.Core logging for use by developers.
When a bookmark is not found (unknown cause), Forms Builder no longer tries to unload the workflow twice and log twice. This exception was previously only found in the Trace logs. In Forms Builder 3.4 and later, it is moved to the Debug level.		

Best Practices for Logging

Important

Log files may contain confidential information such as user names and passwords, account information, etc. It is your responsibility to protect sensitive user and system data.

To mitigate the risks of exposing sensitive data, observe the following best practices:

- Set the log level in production environments to the lowest, least detailed log level. Increase the log level in test environments only when needed. Reset the log level when testing is complete.

  The default logging provider used by Anthology Inc. products is NLog. NLog allows you to configure log targets, levels, rules, layouts, etc. through configuration. To configure logging, modify the NLog.config file in the application's executing directory. For Web applications, this file exists alongside the web.config file.

- When LogLine or LogObject workflow activities are used to capture entities that contain sensitive information, remove such activities as soon as testing is complete.

  We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

  If, instead you followed the recommendations, and the development machine NLog minLevel is set to "Info" and all logging is done at the "Information" level, and the production machine NLog minLevel is set to "Error" (default), then nothing needs to be done because the production machine will not log "Information" LogLine or LogObject activities. The additional benefit is that the logging is still available if a problem occurs only in a production machine and lowering the NLog minLevel to "Info" temporarily (and restarting the app pool) will allow troubleshooting.

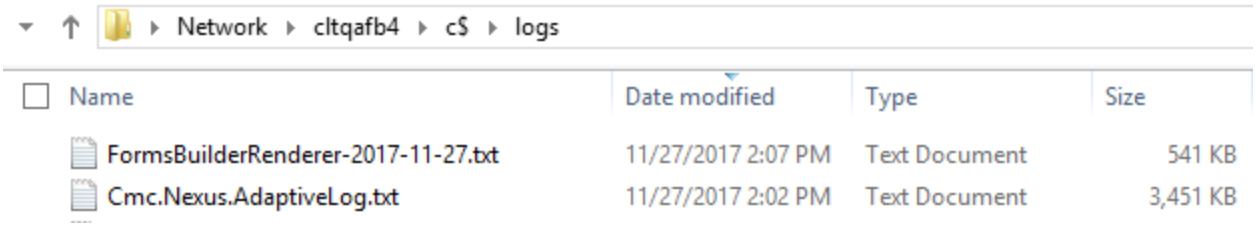
Location of Log Files

Forms Builder Logs

- Local logs: **C:\logs**
- Server logs: **\\<server>\c$\logs**

  Where <server> is the name of the server where Form Designer and/or Forms Renderer is installed.

  The log folder on the server contains numerous installation logs and error logs for Anthology applications. The most commonly used log file for troubleshooting Forms Builder issues are the **FormsBuilderRenderer<date>.txt** and **Cmc.Nexus.AdaptiveLog.txt**. Sort the folder by "Date modified" and open the most recent log.



Event Logs

Event logs for workflows that are executed on an Anthology Student server are written to the following folder on the server machine:

Program Files (x86)\CMC\C2000\Services\Nexus Event Notification Service <version>\logs

The logs capture all workflow events including LogLine output, events associated with long running workflows, and errors captured by the Service Module Host.

Logs for Saved Events

For Saved events, logs are found in:

Program Files (x86)\CMC\C2000\Services\CampusVue Nexus Event Notification Service<version>\logs.

Logs for Saving Events

The host process for Saving events from the legacy interface for Anthology Student is the CampusLink WCF API (worker process). The logs are on the API server under **cmc.campuslink.webservices.Wcf\logs**.

The logs for Saving events from the standard interface for Anthology Student are on the Web Client server in the **\logs** folder in the **Cmc.Nexus.AdaptiveLog** file.

LogLine/LogObject Activities

Add LogLine or LogObject activities at critical points within the workflow, for example, after "Get" or "Save" operations.

We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

Use the following settings in LogLine activities to catch all errors and all attributes associated with an object:

- Level = **Information**
- Text = **Newtonsoft.Json.JsonConvert.SerializeObject(<object>, Newtonsoft.Json.Formatting.Indented)**

  Where `<object>` is replaced with the name of the entity for which you want to capture details, e.g., studentEntity. The log file will contain the values for all attributes of the specified entity.

NLog Levels

1. On the test system, in the CMCFormsRenderer_V3 folder, locate the **NLog.config** file.
2. In the <rules> section of the NLog.config file, change the **minLevel** to "**Info**". Do not change this on your production system.

```
<rules>
  <logger
    name = "*"
    minLevel = "Info"
    writeTo = "file" />
</rules>
```

Now any LogLines in your test machine that log to “Information” will be found in the logs.

On your production machine where the minlevel is “Error”, these will not log, making it unnecessary to remove the LogLines from a production ready workflow (for performance reasons), with the option of enabling them if necessary.

Common Errors and Solutions

Scroll through the log file and locate the pertinent time stamps or search the log file with keyword “error”.

Some error messages include details about the cause for the error and tips to correct the issue. Correcting one issue may lead to the resolution of multiple related errors.

The following tables list common errors found in the Forms Renderer and Form Designer logs and provide suggestions for actions to take to correct the errors.

FormsBuilderRenderer Log

Error Type	Example	Action to take
Durable Instancing	Cmc.Nexus.FormsBuilder.Renderer.Web.Services.FormInstanceService The workflow version for this sequence may have been updated after the sequence was published. Records in the [System.Activities.DurableInstancing].[InstancePromotedPropertiesTable] table may be referencing a different version of the workflow. FormInstance WorkflowDefinitionVersionId: 2577 WorkflowDefinitionVersionId from DB: 2995	Clear persisted workflows and rerun the sequence.
Endpoint Not Found Exception	Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.FormInstancesController Unable to create new form instance: System.ServiceModel.EndpointNotFoundException: There was no endpoint listening at http://<server->.anthology.com/Cmc.Nexus.Web/Services/Common/StudentService.svc that could accept the message. This is often caused by an incorrect address or SOAP action.	A form is trying to use an entity that does not yet have a service running. Check if the Anthology web service exists; if not, check if the service is supported.
Index Out Of Range Exception	Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.FormInstancesController Unable to execute a form instance service transition: System.IndexOutOfRangeException: Index was outside the bounds of the array.	In this case, a File Upload component was used with a GetAttachments activity and the ControlIdentifier argument was not correct (copied from another workflow). When the ControlIdentifier is incorrect, it cannot be connected to the File Upload control and GetAttachments will get no data.
JSON Serialization Exception	Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.FormInstancesController Unable to execute a form instance service transition: Newtonsoft.Json.JsonSerializationException: Cannot deserialize the current JSON object (e.g. {"name":"value"}) into type 'System.String[]' because the type requires a JSON array (e.g. [1,2,3]) to deserialize correctly. To fix this error either change the JSON to a JSON array (e.g. [1,2,3]) or change the deserialized type so that it is a normal .NET type (e.g. not a primitive type like integer, not a collection type like an array or List<T>) that can be deserialized from a JSON object. JsonObjectAttribute can also be added to the type to force it to deserialize from a JSON object.	Check the data type used most likely for the in/out argument.

Error Type	Example	Action to take
Null Reference Exception	Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.AccountController Unable to logout: System.NullReferenceException: Object reference not set to an instance of an object.	Check the casing on the in/out arguments. If you initialized StudentID and not StudentId, the StudentId will be null.
Unable to get form from database	Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.FormRecordController Unable to get Form data from database: System.ApplicationException: Form Welcome does not exist in the database.	Using SQL check if the form exists in the database. If someone has manually deleted the form, work with your DB administrator to recover it, or create a new Welcome form.
Workflow Definition Version not loadable	WorkflowDefinitionVersion Id 'xxx' is not loadable.	Check if the versions of activities in the workflow are compatible with the versions of the products in your environment. This error occurs if you try to use newer activities in older product environments. Try to open the workflow in the environment where it is failing. If you cannot open the workflow, most likely the form sequence will not render properly either.

FormsBuilderDesigner Log

Error Type	Example	Action to take
Unexpected character encountered	Cmc.Nexus.FormsBuilder.Designer.Web.Controllers.Api.FormsController Unable to save Form data: Newtonsoft.Json.JsonReaderException: Unexpected character encountered while parsing value: [. Path 'formsections[0].columns[0][1].ControlProperties[2].Value', line 1, position 7415.	Check the syntax in the Model property binding.

Logging in Azure

In Azure environments, all logging is "Info" level logging. Logs are accessed through Application Insights or Azure Blob Storage. For more information, refer to Azure Storage Explorer.

> ⚠ In Forms Builder 3.5.1 and later, the ability to set NLog levels in the Settings workspace of Form Designer is removed to prevent conflicts with Azure log configurations. **If your Forms Builder deployment is in an Azure environment, contact Anthology Inc. obtain access to the Azure Blob Storage or to request changes in the NLog settings.**

Several logger.debug statements and client-side logs have been modified to **Info** level to make them available to help debug issues in an Azure environment since in an Azure environment the log level is set to Info level for all products. The Info level is set for logs related to:

- Site Warmup
- LookupUser
- Account Controller
- PDF creation and e-sign documents
- Payment processing for PayPal, ACI, and IATS

Best Practices for Logging

Important

Log files may contain confidential information such as user names and passwords, account information, etc. It is your responsibility to protect sensitive user and system data.

To mitigate the risks of exposing sensitive data, observe the following best practices:

- Set the log level in production environments to the lowest, least detailed log level. Increase the log level in test environments only when needed. Reset the log level when testing is complete.

  The default logging provider used by Anthology Inc. products is NLog. NLog allows you to configure log targets, levels, rules, layouts, etc. through configuration. To configure logging, modify the NLog.config file in the application's executing directory. For Web applications, this file exists alongside the web.config file.

- When LogLine or LogObject workflow activities are used to capture entities that contain sensitive information, remove such activities as soon as testing is complete.

  We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

  If, instead you followed the recommendations, and the development machine NLog minLevel is set to "Info" and all logging is done at the "Information" level, and the production machine NLog minLevel is set to "Error" (default), then nothing needs to be done because the production machine will not log "Information" LogLine or LogObject activities. The additional benefit is that the logging is still available if a problem occurs only in a production machine and lowering the NLog minLevel to "Info" temporarily (and restarting the app pool) will allow troubleshooting.

LogLine/LogObject Activities

Add LogLine or LogObject activities at critical points within the workflow, for example, after "Get" or "Save" operations.

We recommend setting the **Level** value to **Information** for any LogLine or LogObject activities. Refer to Best Practices for Logging and Logging in Azure.

Use the following settings in LogLine activities to catch all errors and all attributes associated with an object:

- Level = **Information**
- Text = **Newtonsoft.Json.JsonConvert.SerializeObject(<object>, Newtonsoft.Json.Formatting.Indented)**

  Where `<object>` is replaced with the name of the entity for which you want to capture details, e.g., studentEntity. The log file will contain the values for all attributes of the specified entity.

Troubleshoot E-Sign Forms

Write the PDF to Disk

Normally, you will not be able to view PDF files generated to send to the e-sign partner until after they are signed and saved in your provider's database. The view of the PDF returned from the e-sign partner is not the actual PDF, but instead a rasterized image of the PDF document created by the e-sign partner. Internally, Forms Builder has this initial PDF data contained within the workflow and model, and in the last step during the GetSignedDocument activity, retrieves the signed PDF from the e-sign partner, so that you can save it.

However, as a troubleshooting tool, you can enable the writing of the PDF to disk at the time it is returned from the conversion process and before being sent to the e-sign partner. To do so, locate the <appSettings> section in your Renderer web.config, and if the following do not exist, add them.

```
<add key="SaveViewCreatorPDFToDisk" value="false" />
```

```
<add key="SaveViewCreatorPDFToDiskPath" value="" />
```

Setting the first one to true will cause the file to be written to disk in the C:\Logs directory. To change the path, override the location with the 2nd key. However, you must ensure the directory you choose is writable by the currently configured IIS web service Application Pool Identity.

Error Code "TAB_OUT_OF_BOUNDS" - DocuSign

If you find the following error in the Renderer log for a form sequence with DocuSign components, you are likely submitting multiple Forms Builder forms as a single DocuSign envelope:

> 2017-05-25 11:40:17.9980 25 Error Cmc.Nexus.FormsBuilder.Services.DocuSignService DocuSign.eSign.Client.ApiException: Error calling CreateEnvelope: {"errorCode": "TAB_OUT_OF_BOUNDS",

To work around this error from DocuSign, ensure that DocuSign controls are not split across page boundaries.

The following styles enforce the default page breaks. They can be modified as needed. Copy the styles and save your changes in a custom style sheet. For more information, refer to Custom Content and Custom Styles.

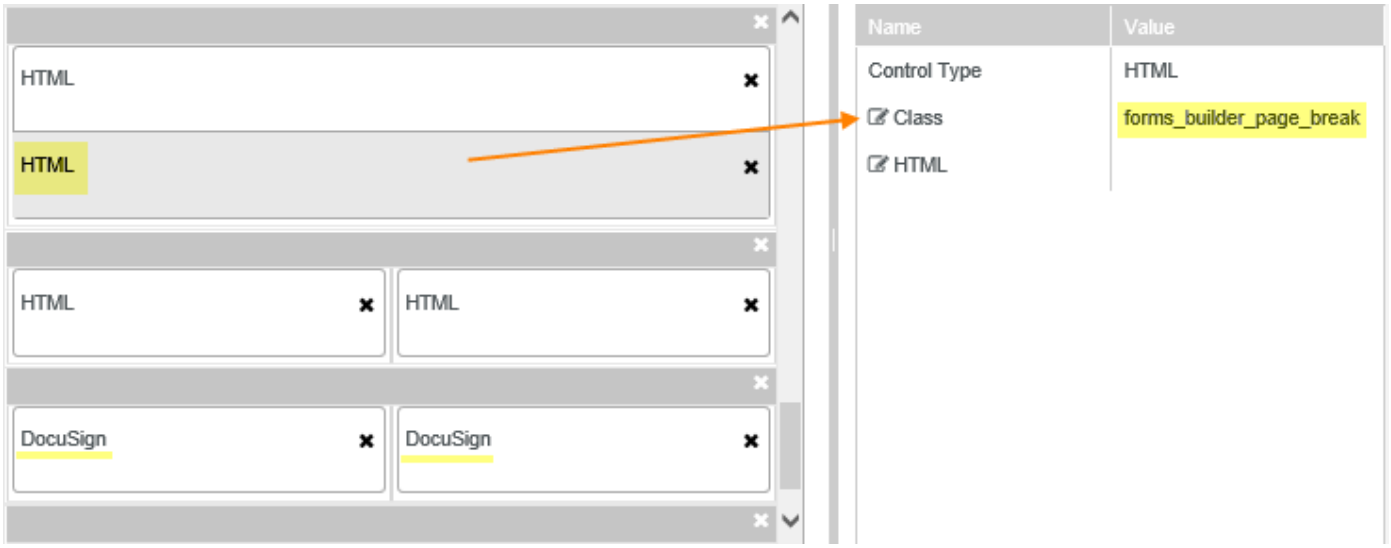
```
/* The following is used during PDF creation for DocuSign and forces page breaks before each form except the first form. This can be commented or modified if some other pagination scheme is desired. */

@media print {
  #viewCreatorForm > div:not(:first-child) {
  page-break-before: always;
  }
}
```

/* The above @media print does not address page breaks within a long form. DocuSign may reject PDF files where DocuSign signatures cross page boundaries in the PDF files, usually with an error like "TAB_OUT_OF_ BOUNDS". Adding appropriate pages breaks is best accommodated by adding an HTML component in the form anywhere a page break is required. Leave the HTML control empty but add the unique class name on the Class property setting: forms_builder_page_break. This will force a page break in the PDF file before the control. */

```
@media print {
   .forms_builder_page_break {
   page-break-before: always;
   }
}
```

On any longer forms that spread across multiple pages when converted to PDF, we recommend that you drop an HTML component with Class `forms_builder_page_break` in a location on your form that will ensure the DocuSign component does not cross a page boundary.



E-Sign Document is Blank

If an e-sign document is blank, check the URL in the PrintUrlToPdf activity. If one URL character is wrong or spaces are not substituted with “+”, the target PDF will be empty. To verify you have the proper URL, check it with ViewCreator just after you have entered data on the form to be converted. Open a separate tab and ensure your form is displayed at the following URL:

`http://<server><domain>:<port>/#/viewCreator/WorkflowDefinitionId/forms=FormName`

On a slow server the time before the converter starts may not be sufficient. In the Renderer web.config the default setting for ViewCreatorDefaultStartConversionTimerInMilliseconds is 500. Try increasing it to 1000.

Ensure that e-sign components and the Form Section they are contained in DO NOT have *Visible* set to *false* (as that will also hide from the e-sign partner). Use the build-in style hideDocuSignWhiteSpaceInFormSection in the Class property of the Form Section instead.

PrintUrlToPdf Times Out

In Forms Builder 3.3 and later, the viewCreator wait is no longer timer-based, it is now event-based. Forms may need to be re-saved, which will automatically update components to include a call to the event-based method "cmc-on-initialized".

If the PrintUrlToPdf activity times out, especially when many documents are merged into a single PDF file, simply re-save all forms in the sequence.

Hyperlinks Display with Target URL in Parentheses

If a PDF displays the target URL of a link in parentheses after the link text:

Click here (http://google.com)

Add the following to a custom style sheet in the `/Content/Custom/` folder on the Renderer website:

```
@media print {
   a[href]:after {
   content: none !important;
   }
}
```

The result is a proper link like this: Click here

Webhook Responses for Multiple Signer Sequences

Please confirm if all of these WebhookController messages at INFO level are found in the log and that the work-flow returns back to sequence after completing the DocuSign signatures. Similar messages are also logged for Adobe Sign and SignNow multi-signer sequences.

INFO - Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.WebhookController: Starting WebhookController - Post ...

INFO - Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.WebhookController: WebhookController - Envelope returned from DocuSign:

INFO - Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.WebhookController: WebhookController - Envelope Status: Delivered

INFO - Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.WebhookController: Starting WebhookController - Post ...

INFO - Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.WebhookController: WebhookController - Envelope returned from DocuSign:

INFO - Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.WebhookController: WebhookController - Envelope Status: Completed

INFO - Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.WebhookController: WebhookController - Searching for Workflow Instance Id: 664191dc-55ff-4c37-8a27-59d649de3141

INFO - Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.WebhookController: WebhookController - Status of “Signer1” is: Completed

INFO - Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.WebhookController: WebhookController - Status of "Signer2" is: Completed

INFO - Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.WebhookController: WebhookController - Found variableToSet: signer2Complete for Workflow Instance Id: 664191dc-55ff-4c37-8a27-59d649de3141

INFO - Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.WebhookController: WebhookController - ResumeBookmark: Continue when all signatures are collected; Instance Id: 664191dc-55ff-4c37-8a27-59d649de3141

INFO - Cmc.Nexus.FormsBuilder.Renderer.Web.Controllers.WebhookController: WebhookController - Calling FormInstanceService ExecuteBookmarkNow method for Instance Id: 664191dc-55ff-4c37-8a27-59d649de3141

The next line is the LogLine from back in the workflow:

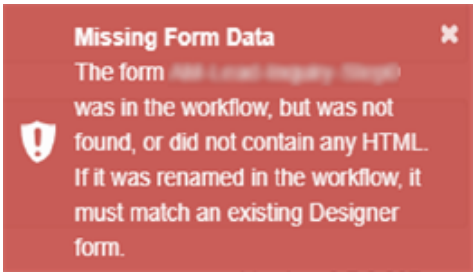
```
2020-10-21 21:18:32.7880 INFO- Cmc.Core.Workflow.Activities.LogLine
```

You will have Workflow Instance Id values that are specific to your workflows.

Troubleshoot Fields and Components

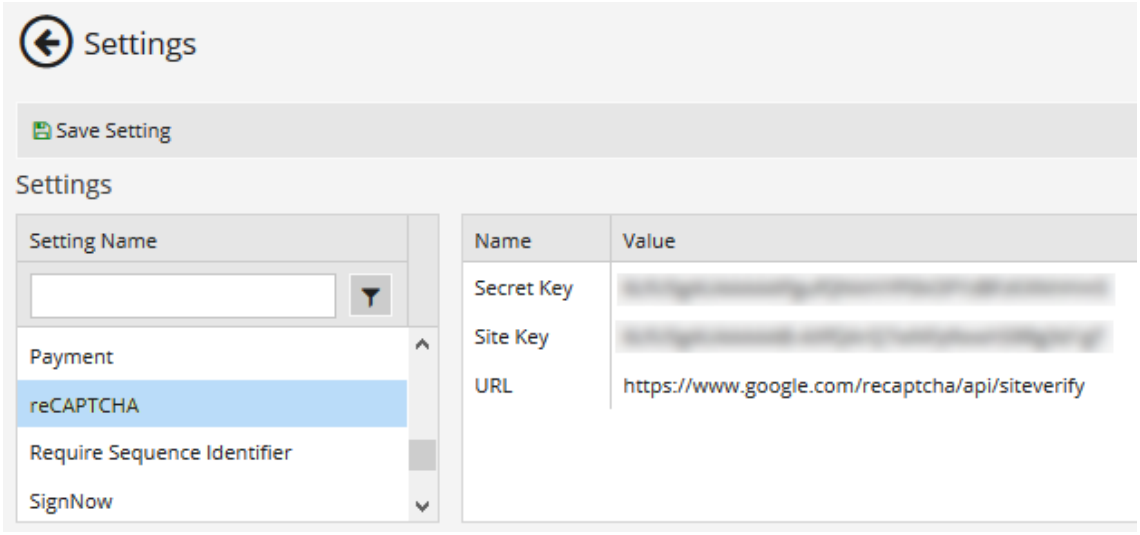
Missing Keys

If Form Designer displays an error similar to the example below, a component on a form is missing a site key and/or secret key.



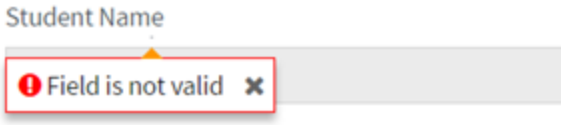
In case of a CAPTCHA component, go to the Forms Builder Settings workspace, select **reCAPTCHA** and populate the **Secret Key**, **Site Key**, and **URL**. Then select **Save Settings**.

You will need to re-enter the reCaptcha Site Key and Secret Key values after upgrading Forms Builder.



Validation Error on Text Boxes

Scenario: A workflow combines the FirstName and LastName fields of the StudentEntity class to display the full student name. When the user selects Next on the rendered sequence, a validation error occurs on the Student Name text box.

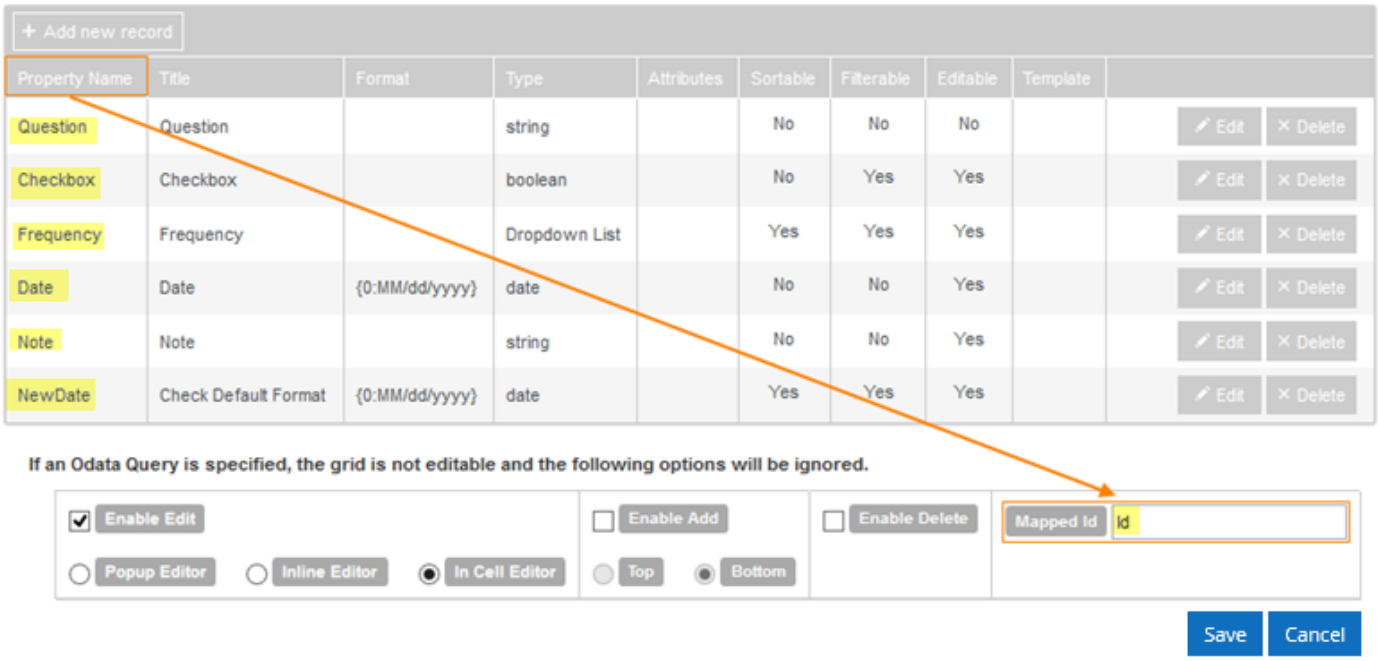


Solution: Check for extra padding on FirstName and LastName fields from the database. The Anthology entity model defines the FirstName and LastName fields as text boxes, each with a Max Length of 25. If the workflow creates a FullName variable with an assignment of `FirstName + " " + LastName`, it is best practice to add a trim to trim the extra spaces. Otherwise, with the extra padding, the length may exceed the MaxLength of 50 for the combined fields.

An alternate solution is to remove the Min Length property setting on the failing text boxes. i.e., leave it blank.

Invalid Property Names in Grids

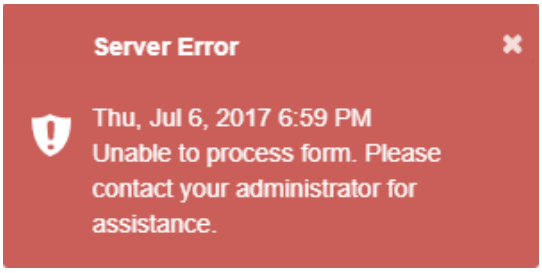
When assigning Property Name values in the column editor, make sure that the values are different than the Mapped ID value. Otherwise, the following error may be displayed: *"An item with the same key has already been added."*



Troubleshoot Rendered Sequences

Workflow Error on Rendered Forms

When a server-side error occurs during the processing of a rendered sequence that cannot be corrected via form resubmission, an error message is displayed. The default message text is *"Unable to process form. Please contact your administrator for assistance."*

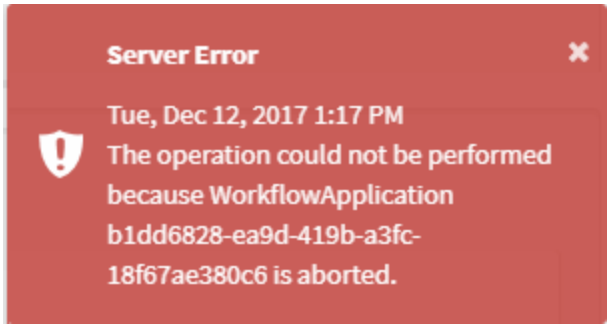


You can edit the message text on the Settings tile in Form Designer and save your custom message. You can use HTML markup to encode a URL or email address if desired.

By design, the error details will only be captured in the log files. You need to check the log files to troubleshoot the error. For more information, refer to Log Files.

Server Error - Workflow Aborted

If a user encounters the following error when selecting Next in a rendered sequence, the most likely cause is that the SQL server connection is slow or down. Hence the instance could not be persisted in the DurableInstance Store at that point. The user should refresh the sequence in the browser and retry.



When this error occurs, you will probably also find some exceptions in the Renderer log such as:

```
2017-12-12 18:40:00.1218 2941 Error                          Cmc.Nex-
us.FormsBuilder.Renderer.Web.Services.FormInstanceService Aborted: Sys-
tem.Runtime.DurableInstancing.InstanceOwnerException: The execution of an
InstancePersistenceCommand was interrupted because the instance owner registration for owner ID
'dd96f9bb-4995-4480-b80d-9e0d99ea703b' has become invalid. This error indicates that the in-
memory copy of all instances locked by this owner have become stale and should be discarded,
along with the InstanceHandles. Typically, this error is best handled by restarting the host.
```

Remedy:

Execute an IISreset on the server.

Forms are Skipped

If a rendered sequence does not display all forms in the sequence, check for missing WaitForFormBookmark activities in the transitions before and after the skipped forms.

E-Sign Document is Blank

If an e-sign document is blank, check the URL in the PrintUrlToPdf activity. If one URL character is wrong or spaces are not substituted with "+", the target PDF will be empty. To verify you have the proper URL, check it with ViewCreator just after you have entered data on the form to be converted. Open a separate tab and ensure your form is displayed at the following URL:

```
http://<server><domain>:<port>/#/viewCreator/WorkflowDefinitionId/forms=FormName
```

On a slow server the time before the converter starts may not be sufficient. In the Renderer web.config the default setting for ViewCreatorDefaultStartConversionTimerInMilliseconds is 500. Try increasing it to 1000.

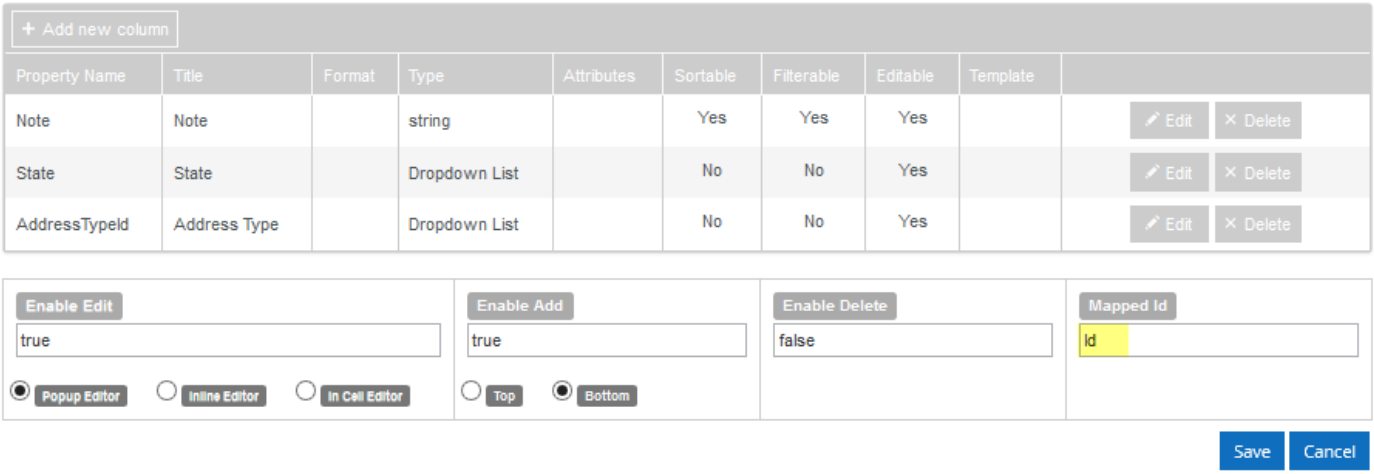
Ensure that e-sign components and the Form Section they are contained in DO NOT have *Visible* set to *false* (as that will also hide from the e-sign partner). Use the build-in style hideDocuSignWhiteSpaceInFormSection in the Class property of the Form Section instead.

Disappearing Grid Rows on Edit

Solution: The Mapped Id used for the grid must be unique per row.

- For most Anthology Student entities, the **Id** field can be used.
- For CampusNexus CRM entities, the **KeyId** field can be used.

If using a SerializableDynamicObject not bound to a defined entity, the grid will automatically assign a unique Id to be used.



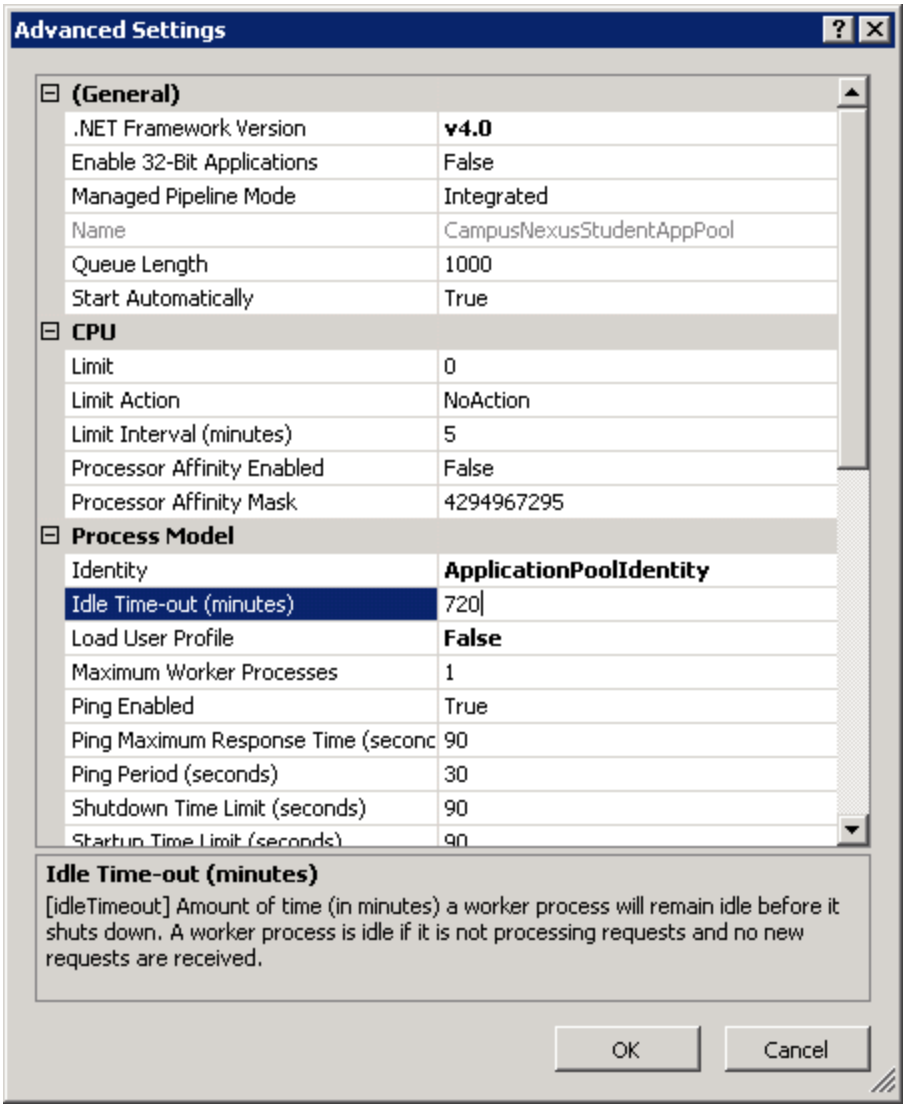
Slow Loading of Authenticated Sequences

If the first form in an Anthology Student sequence is slow every now and then, but subsequent loads are okay, this could mean that the Anthology Student application, whose services are most likely invoked with a Get or Create activity on the first form's entry, may be taking some time to start up.

The default value of Idle Time-out setting for the Application Pool used by the Web Client for Anthology Student is 20 minutes. If you don't have much traffic or intermittent traffic, consider increasing this value to 12 (720 minutes) or 24 hours (1440 minutes).

1. In Internet Information Service (IIS) Manager, expand the server node, and select **Application Pools**.
2. Select the **CampusNexusStudentAppPool** and select **Advanced Settings**.

3. In the Process Model section, increase the **Idle Time-out (minutes)** value to 720 or 1440.

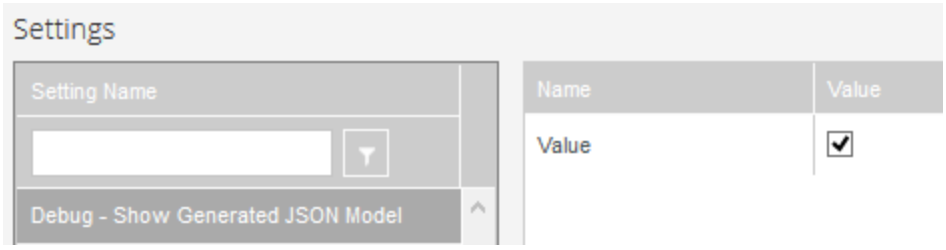


Visually Examine Data in Renderer

The following JSON display has been replaced by a setting in Forms Builder 3.4 which allows you to turn it on for the entire site (normally only done in a development environment). However, the following technique may still be useful to embed this code in a single form temporarily on a production site to display data for a data-dependent situation.

You can add a visual display of the current data in a form by exposing the JSON model. There are two ways to look at the data:

A. You can look at the JSON on a global basis for every sequence and every form with the following option in the Settings workspace of Form Designer:



B. You can use the JSON Debug Info component to help debug forms in production mode where the JSON information is needed for a single form and not the entire site (refer to JSON Debug Info).

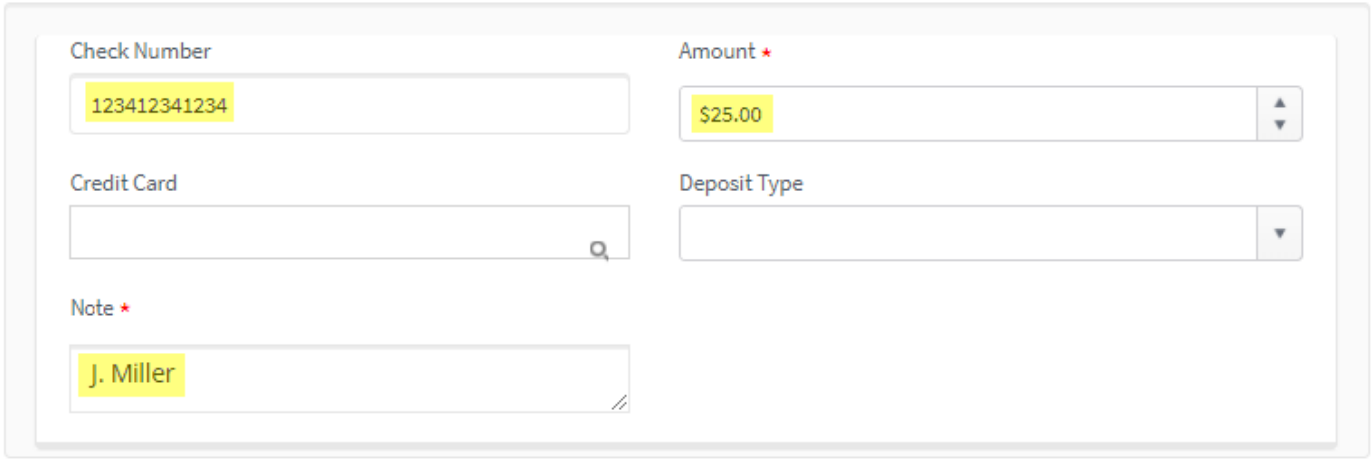
Debug - Show Generated JSON Model

When the "Debug - Show Generated JSON Model" option is enabled in the Forms Builder Settings workspace, additional data that shows the values for objects on the form will be shown at the bottom of each rendered form. This data can be helpful for troubleshooting, especially for complex components on a page where knowing the data that is available to a workflow during a transition will aid in debugging a workflow.

On form load, the Generated Model for Debugging section shows the Renderer Media Variables. As the form fields are populated with values, the debugging section displays the values associated with each object on the form, i.e., all model entity data on the page and new values entered are displayed in real time.

Notes:

- On form transition, the entity data is serialized, and the debugging section will present all fields (not just the fields completed for the entity in the form).
- A createEntity or getEntity activity in the workflow will also cause all data fields of an entity to be included in the debugging section.



Generated Model for Debugging

Version: 3.4.0.16


```
{
  "isMobile": false,
  "isMobileNeg": true,
  "xsMedia": false,
  "xsMediaNeg": true,
  "gtxsMedia": true,
  "gtxsMediaNeg": false,
  "smMedia": false,
  "smMediaNeg": true,
  "gtsmMedia": true,
  "gtsmMediaNeg": false,
  "mdMedia": true,
  "mdMediaNeg": false,
  "gtmdMedia": false,
  "gtmMediaNeg": true,
  "lgMedia": false,
  "lgMediaNeg": true,
  "gtlgMedia": false,
  "gtlgMediaNeg": true,
  "xlMedia": false,
  "xlMediaNeg": true,
  "landscapeMedia": true,
  "landscapeMediaNeg": false,
  "portraitMedia": false,
  "portraitMediaNeg": true,
  "printMedia": false,
  "printMediaNeg": true,
  "debuggerIsAttached": false,
  "depositEntity": {
    "CheckNumber": "123412341234",
    "Amount": 25,
    "Note": "J. Miller"
  }
}
```


Renderer Media Variables

Values specified for the depositEntity object on the form

DbUpdateConcurrency Exception

A DbUpdateConcurrency error occurs when an attempt is made to update an instance of an entity via a Save activity, but that instance has been modified by another user in the time from when the instance was initially retrieved in the workflow to the point in time when the Save activity executes.

Example of a DbUpdateConcurrency exception in a Renderer log file:

> 2018-02-27 13:30:16.7645 54 Error Cmc.Nexus.Crm.Workflow.SaveDocument System.ServiceModel.FaultException`1[System.ServiceModel.ExceptionDetail]: Store update, insert, or delete statement affected an unexpected number of rows (0). Entities may have been modified or deleted since entities were loaded. See http://go.microsoft.com/fwlink/?LinkId=472540 for information on understanding and handling optimistic concurrency exceptions. (Fault Detail is equal to An ExceptionDetail, likely created by IncludeExceptionDetailInFaults=true, whose value is: System.Data.Entity.Infrastructure.DbUpdateConcurrencyException: Store update, insert, or delete statement affected an unexpected number of rows (0). Entities may have been modified or deleted since entities were loaded. See http://go.microsoft.com/fwlink/?LinkId=472540 for information on understanding and handling optimistic concurrency exceptions. ----> System.Data.Entity.Core.OptimisticConcurrencyException: Store update, insert, or delete statement affected an unexpected number of rows (0). Entities may have been modified or deleted since entities were loaded. See http://go.microsoft.com/fwlink/?LinkId=472540 for information on understanding and handling optimistic concurrency exceptions. at System.Data.Entity.Core.Mapping.Update.Internal.UpdateTranslator.ValidateRowsAffected(Int64 rowsAffected, UpdateCommand source) at System.Data.Entity.Core.Mapping.Update.Internal.UpdateTranslator.Update() at System.D...).

The best practice to avoid this error is to add a TransactionScope activity to the workflow. Use the defaults of IsolationLevel = Serializable, and a timeout of 1 minute.

Within that TransactionScope, add a **GetEntity** activity to retrieve the instance of the entity **prior to** the execution of the **SaveEntity** activity. Any property values that need to be updated prior to saving can be done so via Assign statements right after the Get activity and right before the Save activity.

A transaction locks the database to give the workflow a chance to read and update with no other process simultaneously doing the same. Read about the other less aggressive isolation levels as they may be adequate for the purpose based on the type of updates being done and produce less overhead. Google “TransactionScope IsolationLevel Activities”. A “RepeatableRead” may be sufficient.

This pattern will eliminate any chance that another user will update this record in between the execution of the Get and Save activities within the workflow.

Access Denied Error

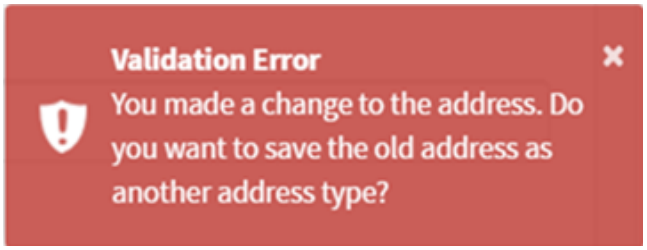
If the API keys are not set up correctly, an "Access denied" error will be found in the Renderer log, for example, when a Forms Builder workflow calls an Anthology Student activity.

Solution: Ensure that the API keys across all products match. For more information, refer to API Keys.

Student Address Changes

The message *"You made a change to the address. Do you want to save the old address as another address type?"* is displayed when a student's address is modified including any of the following fields of the StudentEntity:

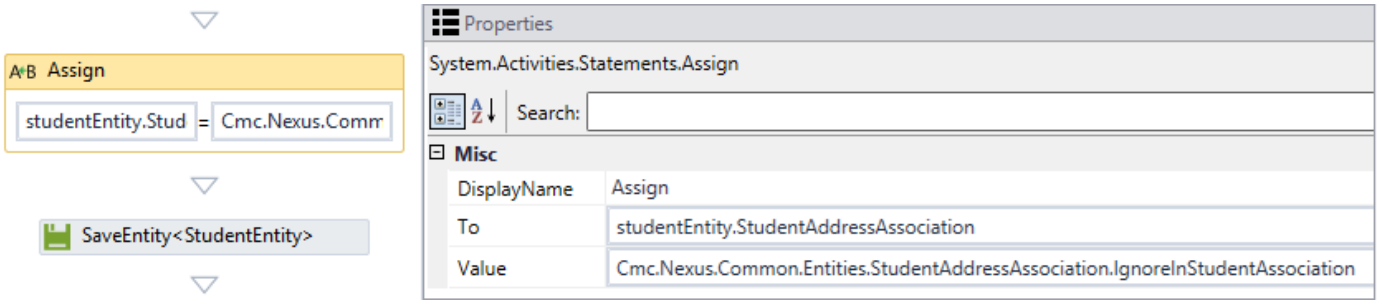
- FirstName
- LastName
- MiddleName
- StreetAddress
- City
- State
- PostalCode
- CountryId
- PhoneNumber
- WorkPhoneNumber
- EmailAddress



The message is not a validation message that can be handled in Forms Builder or in Workflow Composer during a SaveEntity activity for the StudentEntity. To avoid this message after updating a student's address, include an **Assign** activity in the workflow with the following assignment:

```
studentEntity.StudentAddressAssociation = Cmc.Nex-
us.Common.Entities.StudentAddressAssociation.IgnoreInStudentAssociation
```

Insert the Assign activity just prior to the **SaveEntity<StudentEntity>** activity.



If you change an address from Forms Renderer, it will not reflect as the Related Address in the Student Profile of a student in Anthology Student.

Troubleshoot Workflows

Many workflow activities rely on authenticated CampusLink API calls and keys.

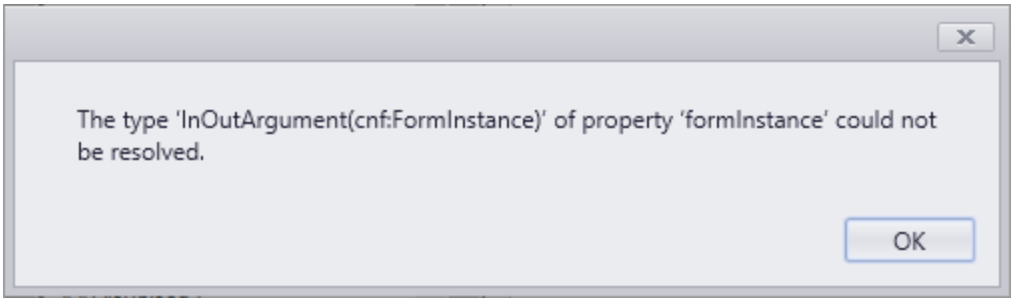
- Prior to Anthology Student 22.0, authentication for API calls required entries in the SyRegistry tables and synchronized passwords and permissions for the API user and staff user. For more information, refer to API Errors with SyRegistry Authentication in Workflow help.
- Anthology Student 22.0 introduces an alternate method for the authentication of CampusLink API calls. The new authentication method does not use the SyRegistry table. For more information, refer to Authentication for CampusLink API Calls in Workflow help.

Exception Handling

The user who creates workflows is responsible for catching exceptions. Any unexpected and uncaught exceptions will abort workflows. For the guidelines on exception handling within workflows, refer to Recommended Coding for Activity Errors in Workflow Help.

Workflow Definition Is Not Displayed

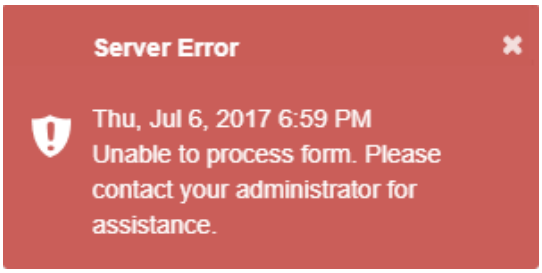
If you try to open the workflow definition for a sequence and a message similar to the following is displayed, the Forms Builder Contracts package has not been installed in Workflow Composer.



To correct this, launch Workflow Composer directly (not from Sequence Designer), select **Package Manager**, and install add the appropriate Forms Builder Contracts package for your environment (refer to Set Up the Database Environment).

Workflow Error Indication on Rendered Forms

When a server-side error occurs during the processing of a rendered sequence that cannot be corrected via form resubmission, an error message is displayed. The default message text is *"Unable to process form. Please contact your administrator for assistance."*



You can edit the message text on the Settings tile in Form Designer and save your custom message. You can use HTML markup to encode a URL or email address if desired.

By design, the error details will only be captured in the log files. You need to check the log files to troubleshoot the error. For more information, refer to Log Files.

Common Workflow Errors

Always check the Error List tab in Workflow Composer to obtain more details about an error.

Always clear persisted workflows before enabling a new version of a workflow.

Spelling and Syntax Errors

Common errors are spelling and syntax errors in argument and variable names, values, and expressions. Be mindful of case sensitivity.

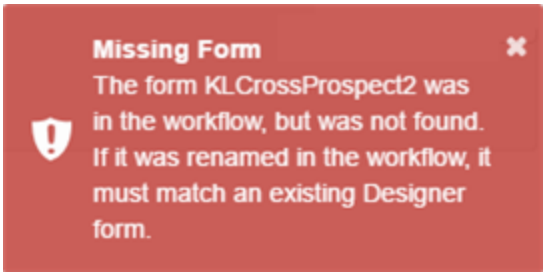
Important

Arguments that will be bound to values in Forms Builder are case sensitive. If they are not bound, they are not case sensitive. *Variables* should not be case sensitive.

Many errors can be avoided by taking advantage of IntelliSense features. For example, type a period (**.**) or press **Ctrl+J**to obtain the valid members from a list. Press **Space** to select a list member.

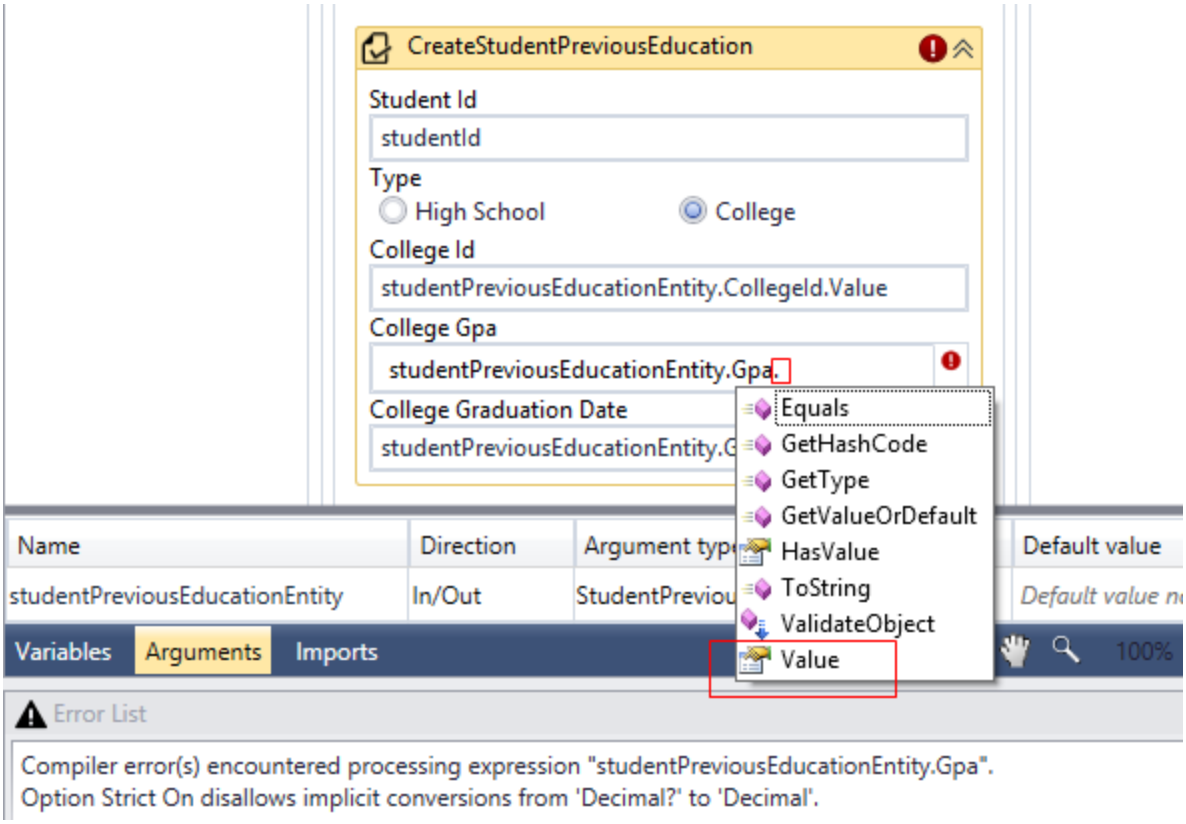
Mismatch Between State and Form Names

When editing a workflow definition, keep in mind that a state in the state machine workflow equates to a form within the sequence. The name of a **State** must match the name of a **Form** to be rendered properly. If Renderer encounters a State in workflow definition that does not match name of any Form created in Form Designer, an error similar to the following will be generated.



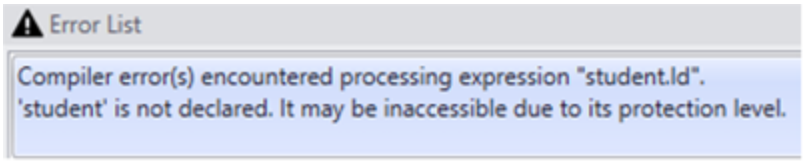
Missing ".Value" Attribute

When entering attributes for entities in workflow activities on objects that can be null, users often forget to select the ".Value" attribute. If an object might be null, *HasValue* in an If/Else Condition can check if there is a Value before trying to use it, otherwise it will cause an error. To clear the error, use IntelliSense to find the dot (.) Value.

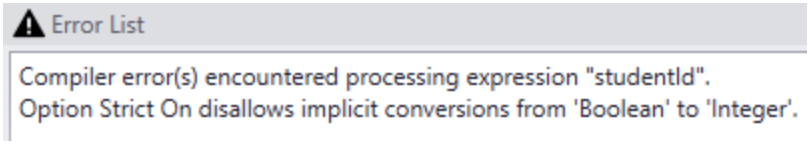


Undeclared Variables and Incorrect Variable Types

Undeclared Variable



Incorrect Variable Type



Incorrect Notation for Variables

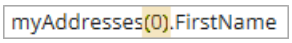
Array variables in the Property Settings pane of Form Designer use AngularJS notation with "square brackets" [].

Example

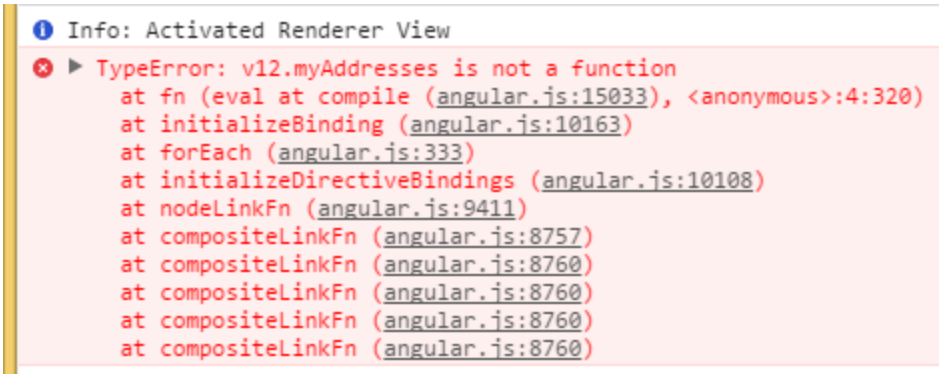


Array variables in Workflow Composer require VB.NET notation with "rounded brackets" ().

Example



If you use "rounded brackets" () in Form Designer, rendering of the sequence will fail, and the browser Developer Tools will show the following error:



SaveDocument Does Not Produce Expected Updates

Anthology Student allows administrators to specify document policies that define which users can assign, read, edit, and close particular kinds of documents. The policies also define which document status changes are permitted. The document policies are assigned to staff users.

Check your document policy settings for allowed document status changes if the workflow for a form sequence includes a SaveDocument activity but does not produce the expected document updates.

NullReferenceException in Log Files

Your workflow will abort if it throws exceptions (and could leave stale persisted instances). One very common exception is a NullReferenceException found in the log file. This occurs when you try to reference a property and its value on an object, but the object is null.

Example 1

myInt is defined as a Nullable<Int32> type.

Since you know you must access the value with a myInt.Value property, you log a line or use it in a workflow without checking whether it is actually null first. If myInt is null, myInt.Value will throw that exception and abort your workflow!

Example 2

myEntity is an entity you are using. It has many properties. If you reference any of those properties but myEntity has not been defined yet, again it will throw that exception. Likewise, if any of its properties are nullable and you try to use that property, *Example 1* applies.

A NullReferenceException can also occur if you attempt to assign a value to an entity type object that has not yet had an associated GetEntity or CreateEntity activity.

Example 3

myObj is an object of some type. It has not been defined. You execute myObj.ToString or reference myObj.myProperty. Either will also throw that exception.

Remedy

1. Here is a cautious way to log myInt by using the VB.Net ternary operator.

   If (myInt.HasValue, “This is my value “ & myInt.Value.ToString, “My value was null, and I did not expect that”)

   If you are using other complex properties and you are assigning from a nullable type to a non-nullable type, you need to make sure it is checked.

   In an assign statement

   myNonNullableIntType = if (myInt.HasValue, myInt.Value, -1)

   The value -1 could be any real integer that would signify that there was no actual value. Note that if this is an entity and that happened to be a child id, an invalid value could throw other exceptions when an attempt was made to store it in the database. In that case an “If” activity, which takes some other action (such as sending back a validation error), is more appropriate.

2. myEntity should be checked for null before using it by using an “If” activity, unless you know it cannot be null.

   if (myEntity is Nothing)

3. myObj should be checked whether it is null when logging (unless you know it can’t be), and myProperty should also be checked if it is nullable.

   if (myObj is Nothing, “myObj was null”, if (myObj.myProperty.HasValue, myObj.myProperty.Value.ToString, “myObj.myProperty was null”))

Error Converting Null Object When Using Drop-down List Component

When the Model property in a Drop-down List component is bound to an integer value and the selection of a value is optional, the following errors occur if the variable is **not** defined as Nullable<Int32> in Workflow Composer.

```
***Workflow failed:****<br/>Error converting value {null} to type 'System.Int32'. Path 'myLead', line 1, position 15600.

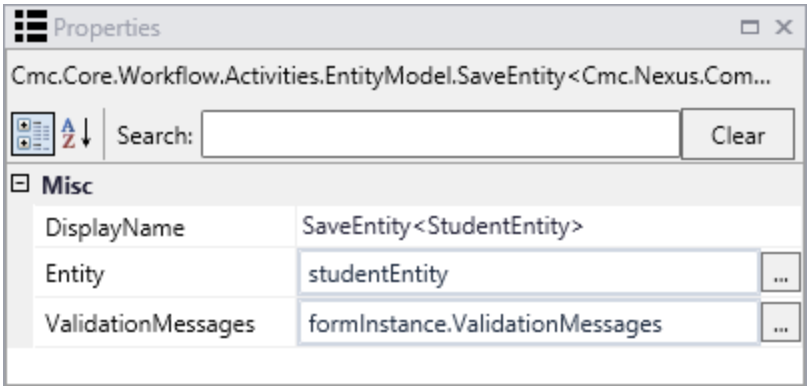
Error converting value {null} to type 'System.Int32'. Path 'myLead', line 1, position 15600.

Null object cannot be converted to a value type.
```

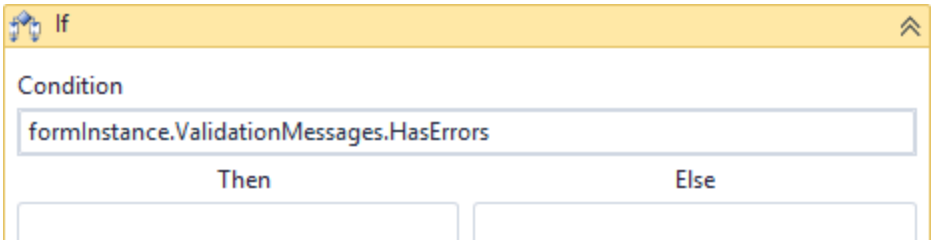
For more information, refer to Drop-down List component.

Validation Messages

In activities that provide a ValidationMessages field, you can specify the value **formInstance.ValidationMessages**. The in/out argument *formInstance* is available with every form. It provides a set of attributes including *ValidationMessages*.



You can also specify conditions such as **formInstance.ValidationMessages.HasErrors** or **not formInstance.ValidationMessages.HasErrors** and proceed as needed depending on the result of the expression.



Print an Array of Validation Messages

If you are trying to print validation messages from an array (validationMessages(0).item), the following error may appear in the log file:

Cmc.Nexus.FormsBuilder.Renderer.Web.Services.FormInstanceService System.NullReferenceException: Object reference not set to an instance of an object. at lambda_method(Closure , ActivityContext)

Before attempting to print an item in an array that may never have been initialized, ensure that the workflow contains the If/Then condition "validationMessages.HasErrors". Another way to define the condition is “If Not ArrayVariable is Nothing”.

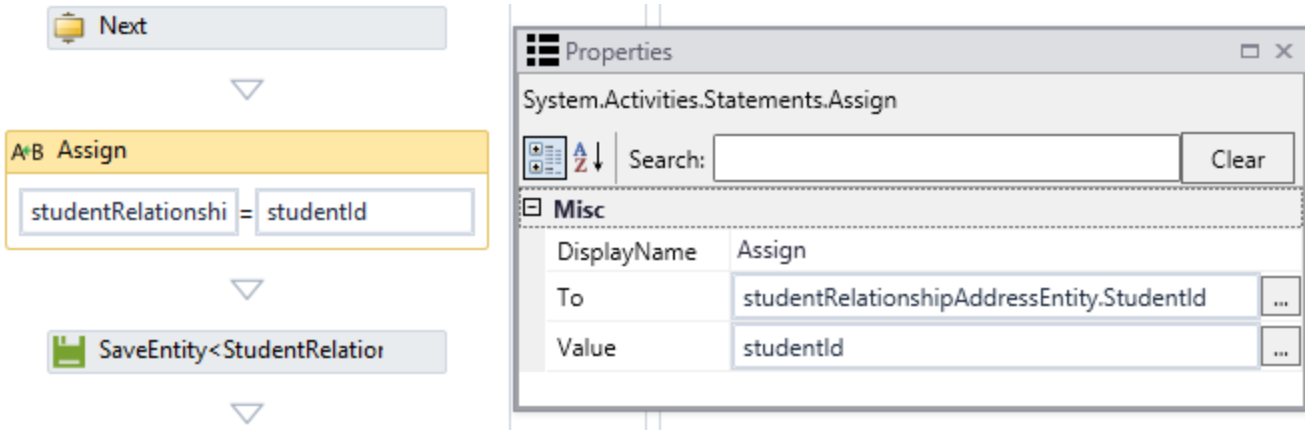
Assign Ids

Some entities in the Anthology object model require an Id value in order to save the entity. If the Id is not entered on a form, the associated SaveEntity activity in the workflow will fail, unless the Id is assigned within the workflow.

Example

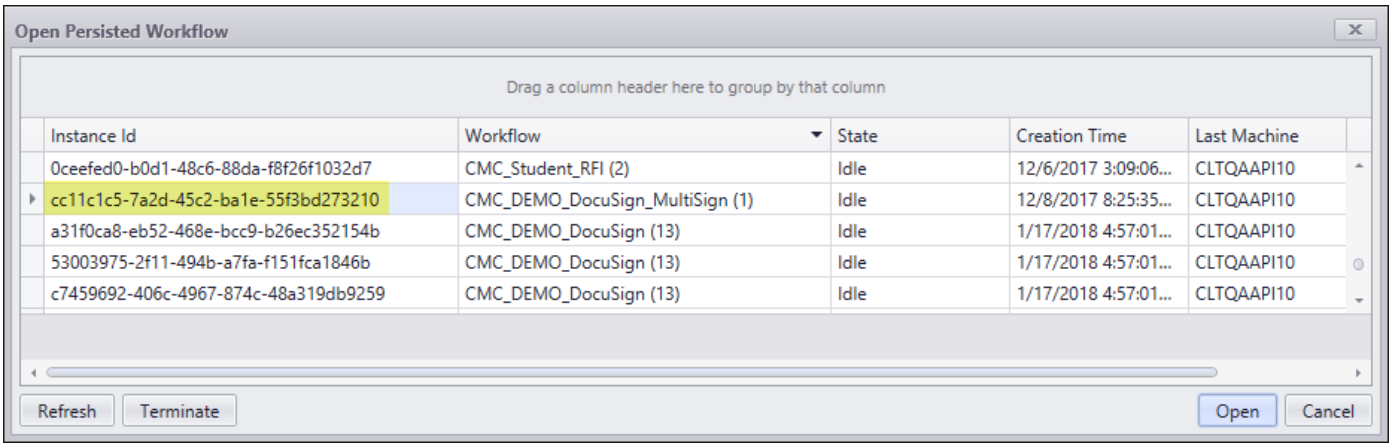
A form collects relationship address data for a student who logged into Portal. The workflow contains a SaveEntity activity for the StudentRelationshipAddressEntity. When the form sequence is rendered, the following error occurs: *The student records you are trying to access are invalid.*

To resolve this error, ensure the workflow assigns a StudentId and not just the new values for the new address before the SaveEntity<StudentRelationshipAddressEntity> activity. The Value field in the Assign activity could contain a variable (studentId) that is populated based on the LookupUser activity.



SQL Query to Determine the UserName for a Persisted Workflow

The Instance Id of a persisted workflow can be used to query the SQL database to determine which student each of the workflow instances applies to.



You can run the following SQL query to determine the UserName associated with the persisted workflow. Replace the highlighted section with the InstanceId from Workflow Composer.

```
SELECT ipp.Value2 as UserName
FROM [System.Activities.DurableInstancing].[InstancePromotedPropertiesTable] ipp
INNER JOIN [System.Activities.DurableInstancing].[InstancesTable] i ON ipp.SurrogateInstanceId = i.Sur-
rogateInstanceId
WHERE i.Id = 'cc11c1c5-7a2d-45c2-ba1e-55f3bd273210'
```

If the user is a GUID, then the sequence was anonymous.

Troubleshooting Tips & Tricks

Angular JS

https://docs.angularjs.org/guide/introduction

AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. AngularJS's data binding and dependency injection eliminate much of the code you would otherwise have to write. In addition, it all happens within the browser, making it an ideal partner with any server technology.

AngularJS is what HTML would have been, had it been designed for applications. HTML is a great declarative language for static documents. It does not contain much in the way of creating applications, and as a result building web applications is an exercise in what do I have to do to trick the browser into doing what I want?

The impedance mismatch between dynamic applications and static documents is often solved with:

- **a library** - a collection of functions, which are useful when writing web apps. Your code is in charge and it calls into the library when applicable, e.g., jQuery.
- **frameworks** - a particular implementation of a web application, where your code fills in the details. The framework is in charge and it calls into your code when it needs something app specific. E.g., durandal, ember, etc.

AngularJS takes another approach. It attempts to minimize the impedance mismatch between document centric HTML and what an application needs by creating new HTML constructs. AngularJS teaches the browser new syntax through a construct we call directives. Examples include:

- Data binding, as in {{}}.
- DOM control structures for repeating, showing, and hiding DOM fragments.
- Support for forms and form validation.
- Attaching new behavior to DOM elements, such as DOM event handling.
- Grouping of HTML into reusable components.

Angular JS Resources

https://www.w3schools.com/js/js_comparisons.asp

https://docs.angularjs.org/guide/expression

Validation Regex Property in Forms Builder

Validation Regex is the regular expression pattern to validate the input. Use a site like http://RegExLib.com to search and test Regex patterns, or construct your own with Regex tools like Expresso. If the user input does not produce a Regex match, the "Validation message" will be displayed.

Examples

Regex that enforces a minimum of 11 digits for a phone number where the first digit must be "1":

```
1\d{10}
```

Regex that enforces a phone number in the format (###)###-####:

```
^\([0-9]{3}\)[0-9]{3}\-[0-9]{4}$
```

Regex that matches a hyphen-separated social security number:

```
^\d{3}-\d{2}-\d{4}$
```

Understanding OData

http://www.odata.org/getting-started/understand-odata-in-6-steps/

OData (Open Data Protocol) is a standard that defines the best practice for building and consuming RESTful APIs. OData helps you focus on your business logic while building RESTful APIs without having to worry about the approaches to define request and response headers, status codes, HTTP methods, URL conventions, media types, payload formats and query options etc. OData also guides you about tracking changes, defining functions/actions for reusable procedures, and sending asynchronous/batch requests etc. Additionally, OData provides facility for extension to fulfil any custom needs of your RESTful APIs.

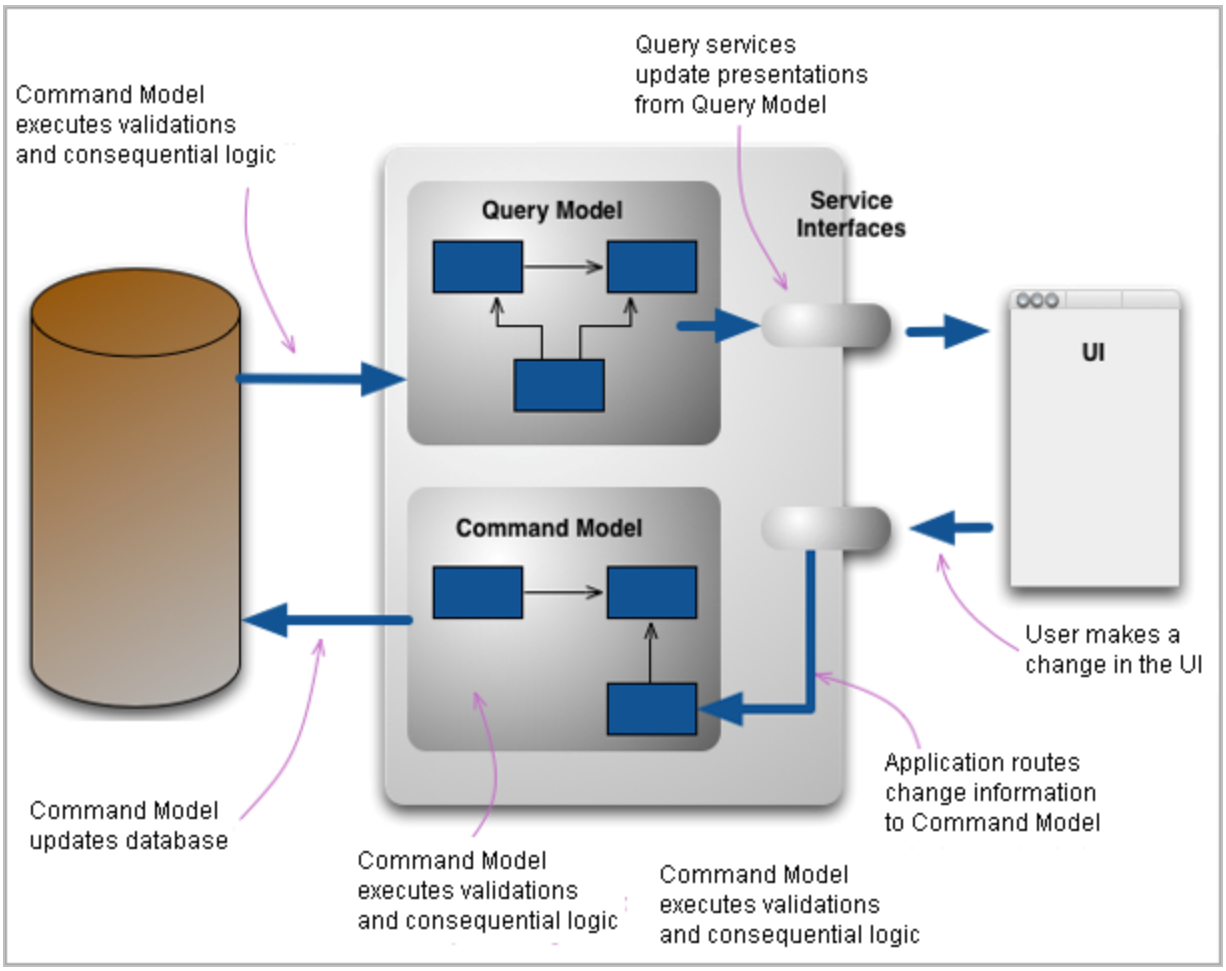
OData RESTful APIs are easy to consume. The OData metadata, a machine-readable description of the data model of the APIs, enables the creation of powerful generic client proxies and tools. Some of them can help you interact with OData even without knowing anything about the protocol.

To learn more about the OData query syntax, refer to http://www.OData.org/ and look for "Basic Tutorial" and "Advanced Tutorial".

Important
OData calls must be made from **within** Forms Builder or Workflow.
OData calls via ApiProxy outside of Forms Builder or Workflow are **not** supported.

Data Model

The Anthology data model is based on the Command Query Responsibility Segregation (CQRS) pattern. As such, it provides a Query Model and a Command Model.



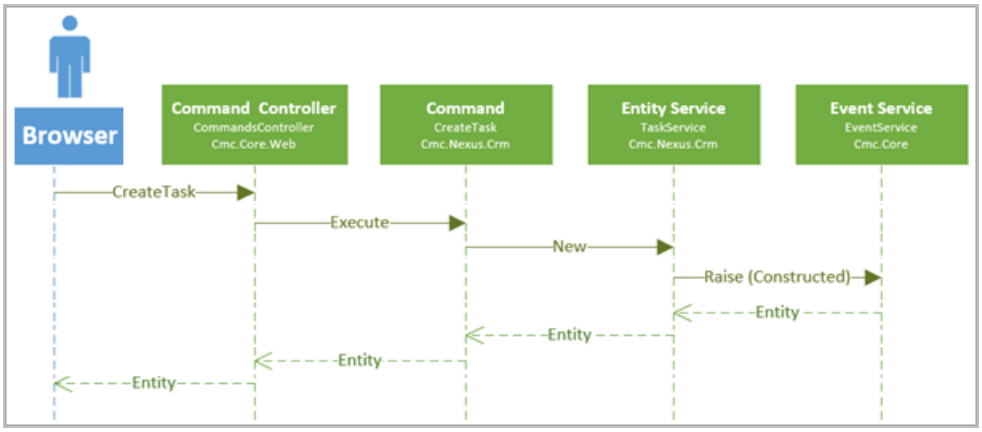
The classes in the following namespaces (assemblies) represent the Command Model interfaces:

- Cmc.Nexus.Academics
- Cmc.Nexus.Admissions
- Cmc.Nexus.CareerServices
- Cmc.Nexus.Common
- Cmc.Nexus.FinancialAid
- Cmc.Nexus.StudentAccounts
- Cmc.Nexus.StudentServices

Command Model

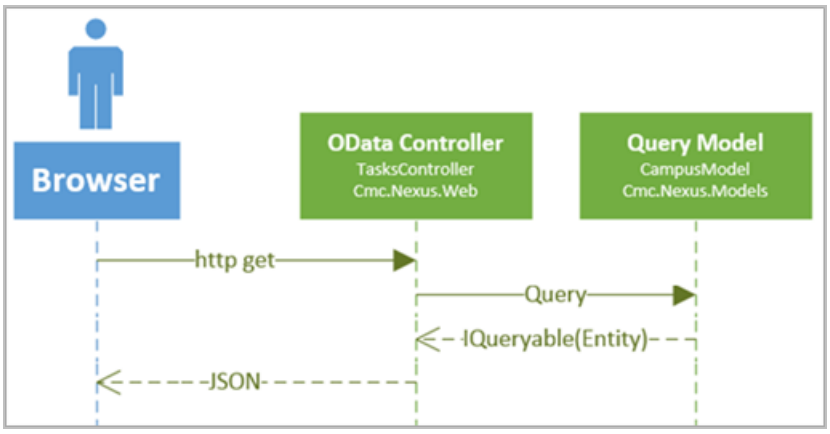
The Command Model is optimized to handle transactions and calculations. Most of the business and decision logic resides in the assemblies listed above.

The Command Model is exposed to the web client through WebAPI Controllers. Unlike the query model, the Command Model's relationships only include the data that will be updated in a single transaction. An example of a transaction sequence is the CreateTask command shown below.



Query Model

The Query Model shows the relationships between the Anthology Student data. The classes in the Cmc.Nexus.Models namespace (assembly) represent the Query Model interfaces. The entity properties of these classes are exposed to the client through OData Controllers. This enables users to navigate through all the relationships in the data model using OData queries as shown in the sequence diagram below.



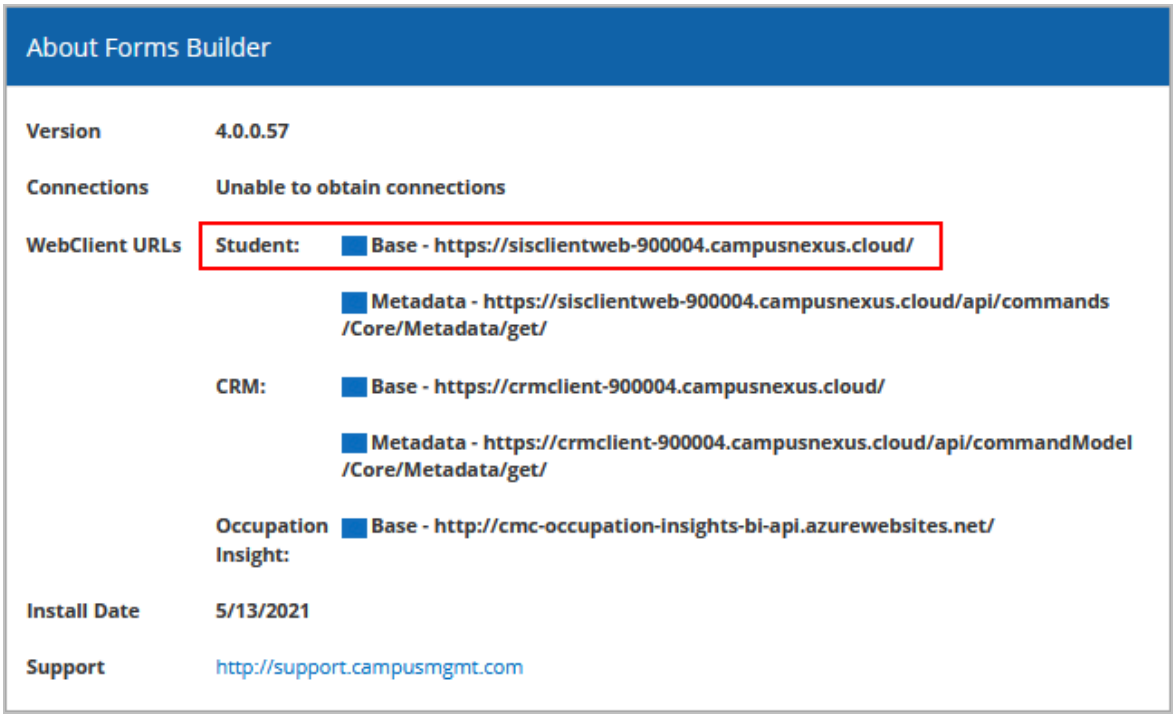
OData Queries

Forms Builder supports list controls such as Drop-down, Multiselect, and Typeahead. Each list control has a Lookup Query property that is used to retrieve specific values from the database. Lookup Queries are specified as OData queries.

The data model provides a query model and a command model. For the purposes of constructing OData queries, refer to the query model. The query model is available at the following URL:

```
<Base URL>/ds/metadata/ModelMetadata/GetFullModel
```

Where <Base URL> is the Student Base URL displayed in the About Forms Builder window:

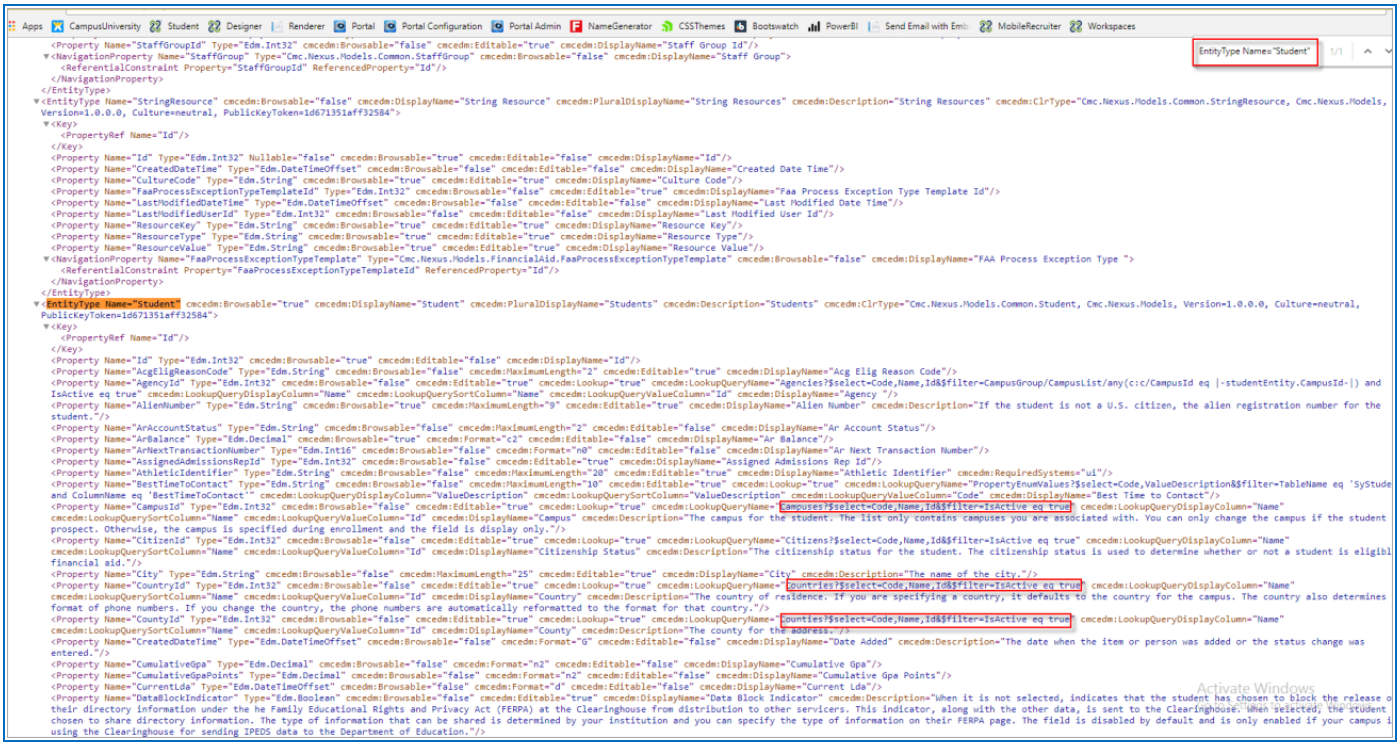


In this example, the query model is available at:

http://student.nexus-02/ds/metadata/ModelMetadata/GetFullModel

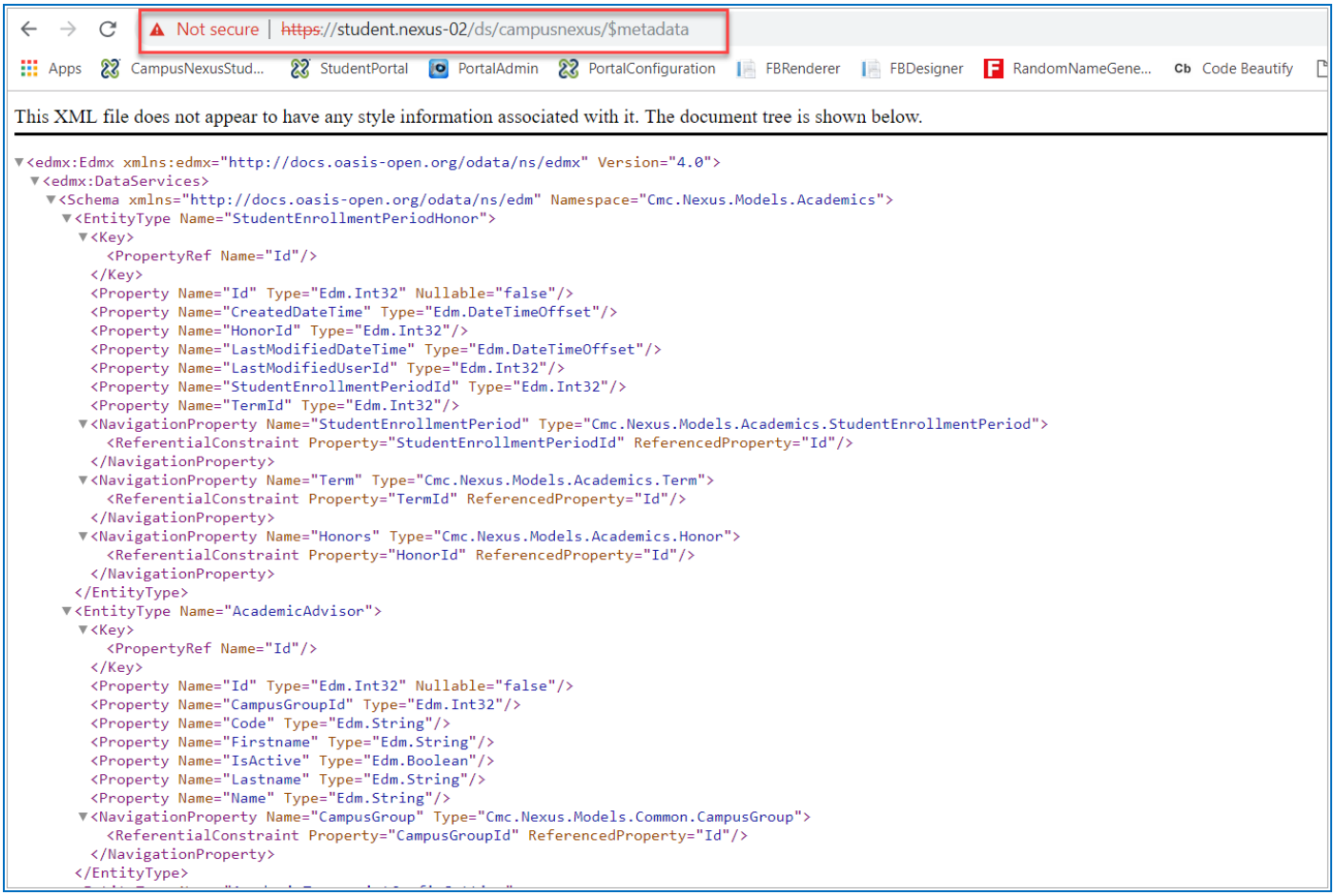
Query the data model (http://student.nexus-02/ds/metadata/ModelMetadata/GetFullModel). Search the metadata for the entity you are working, e.g., EntityType Name="Student". The metadata for the "Student" entity provides several prebuilt OData queries.

All pre-built queries contain "select" options for the Code, Name, and Id columns and are ordered by Name.



Access the **Student Base URL** in a browser. In our example, the Student Base URL is as follows with **/ds/campusnexus/$metadata** appended to the end of the URL to view the metadata values:

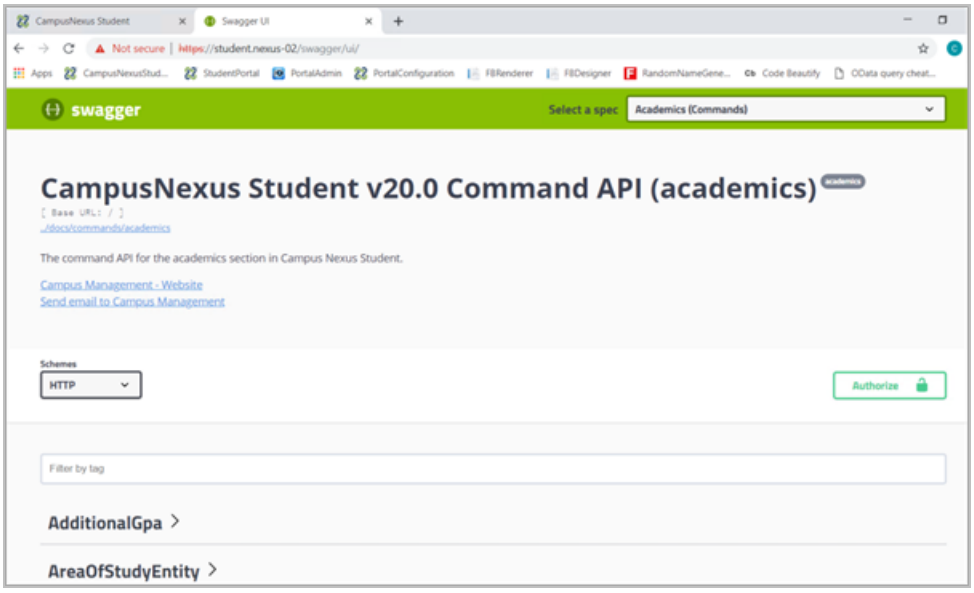
http://student.nexus-02/ds/campusnexus/$metadata



Rest APIs - Swagger

You can access the Swagger UI for the REST APIs for the Web Client of Anthology Student version 19.0 and later using the following steps:

1. Launch Anthology Student in a browser.
2. Log in using administrator-level credentials. If you do not have administrator credentials, refer to the **Note** below.
3. Right-click the tab within your browser, and select **Duplicate Tab**.
4. Replace **#/home** at the end of the URL with **swagger/ui** and hit **Enter**. The Swagger UI will load.



5. **Select a specification** from the drop-down list in the header bar and expand the list of entities and methods.

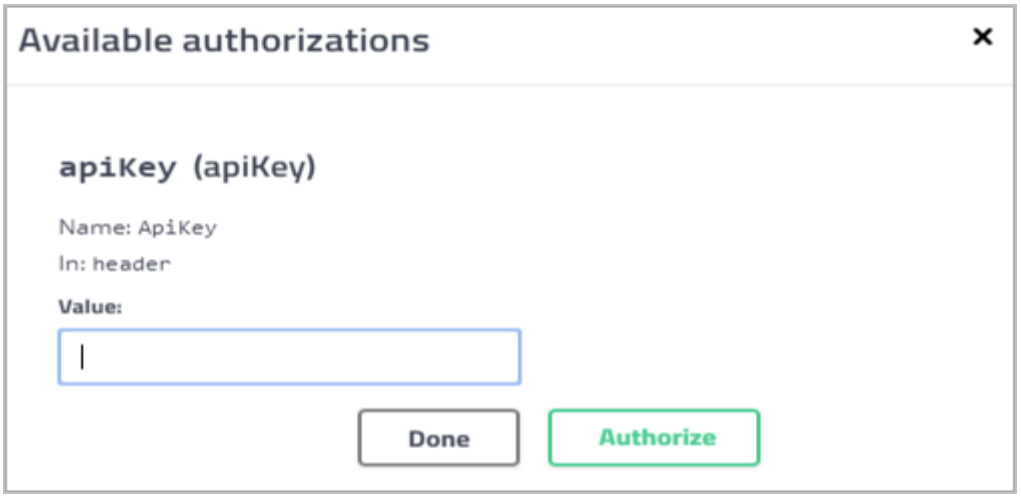
   If you are unsure where the entity you wish to work with resides, refer to Cmc.Nexus.Models in Workflow Help.

6. Use the **Try it out** option in the Swagger UI to test sample values, commands, or models.
7. Use Command Model specifications to leverage the logic behind the REST APIs. Use Query Model specifications to query the OData Model.

If you have not logged in using an administrator account or someone provisioned your account to use the APIs, use the following steps to authorize a user to leverage the REST API logic.

1. On the Server that is hosting the CMCStudentWebClient, open the **web.config** file and search for the **key="apiKey"**.
2. Double-click or highlight the content of the apiKey value (e.g., "c9tZIh-nBgJugdL8H8YVlRJzFkXGSrY6Lk2HCYs4JVcbIjdbUC3SIpNxOJPkZo8qG") to copy it.
3. On the REST API page, select **Authorize**, paste your value into the **Value:** field, and select the **Authorize** but-

ton.


Available authorizations
apiKey (apiKey)
Name: ApiKey
In: header
Value:
Done
Authorize

Log File Locations & Names

The location of logs depends on whether processes are executed in a cloud environment or on premises.

On Premise Installations

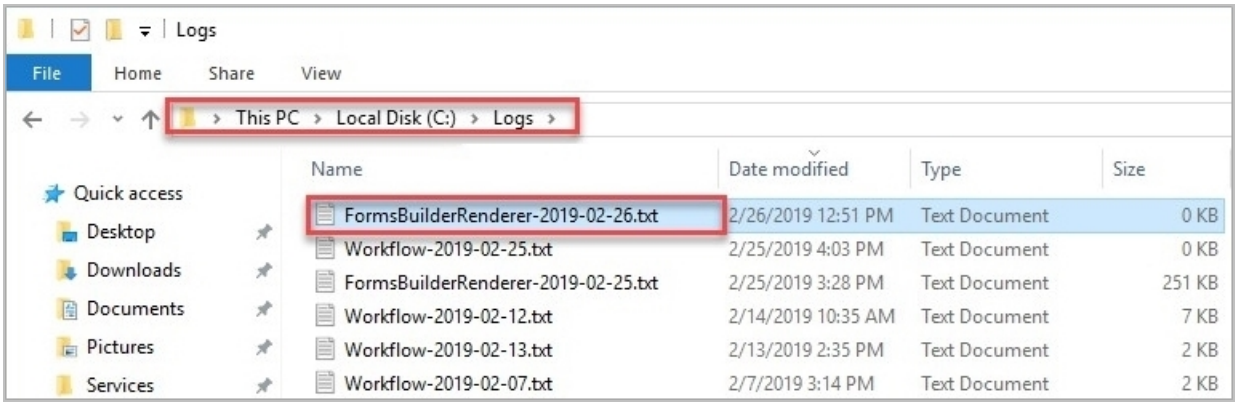
Forms Builder

Log files for Forms Builder are located on the server that Forms Builder Renderer/Designer is installed.

```
\\server\c$\logs
```

(where server is the hostname for your environment)

Look for **FormsBuilderRenderer-date.txt** and **FormsBuilderDesigner-date.txt**.

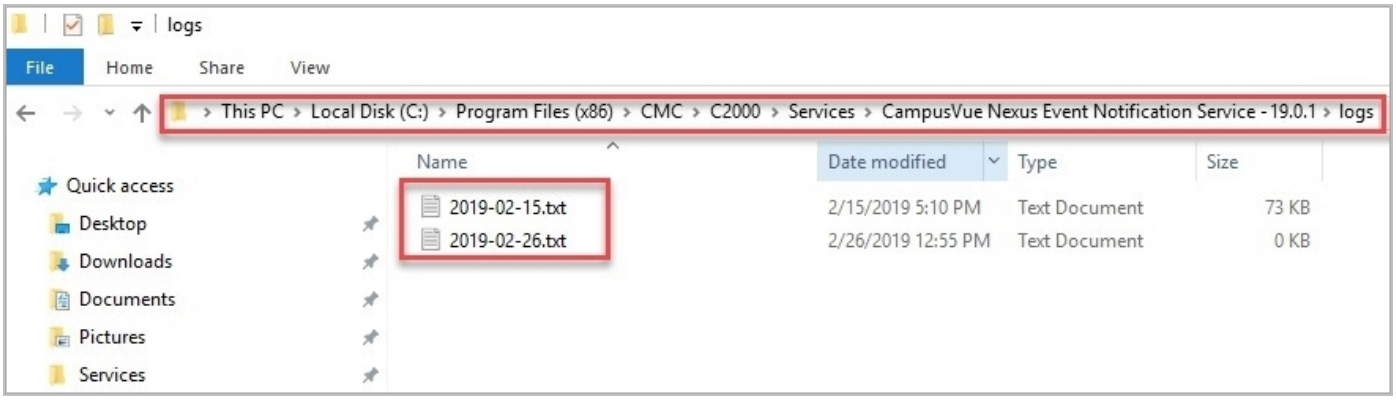


Workflow Saved Events

Log files for Workflow Saved Events are located on the server where the Event Notification Service (Service Module Host) is installed, which is normally the API server.

```
\\server\c$\Program Files (x86)\CMC\C2000\Services\CampusVue Nexus Event Notification
Service - 19.0.1 (Version)\logs
```

(where server is the hostname for your environment)

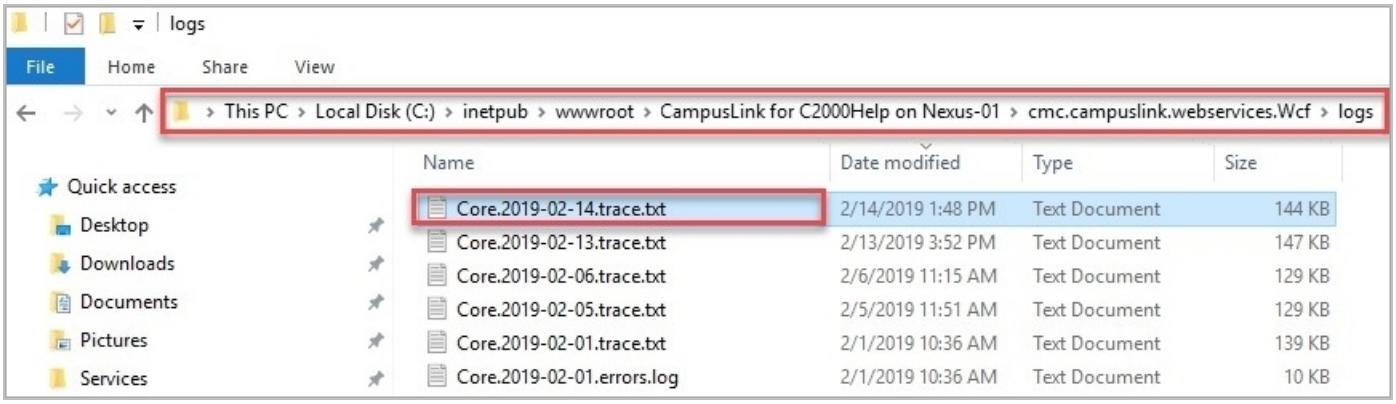


Workflow Saving Events

Log files for Workflow Saving Events are located on the server where the Campuslink web services are installed.

```
\\server\c$\inetpub\wwwroot\CampusLink for C2000Help on Nexus-01 (Database and Server Name of the environment)\cmc.campuslink.webservices.Wcf\logs
```

(where server is the hostname for your environment)

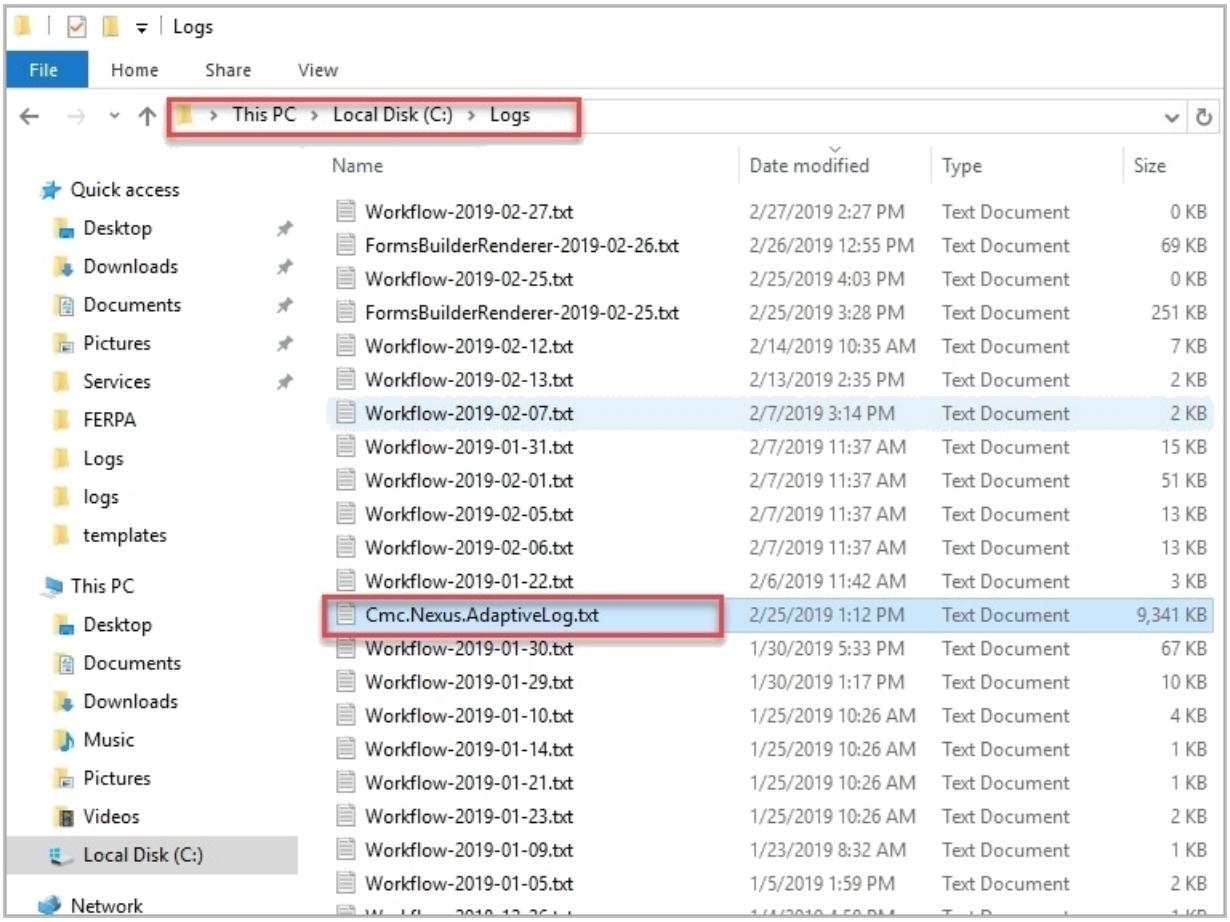


Web Client

Log files for Web Client which includes services and activity logs are located on the Web server.

```
\\server\c$\Logs\CMC.Nexus.AdaptiveLog.txt
```

(where server is the hostname for your environment)



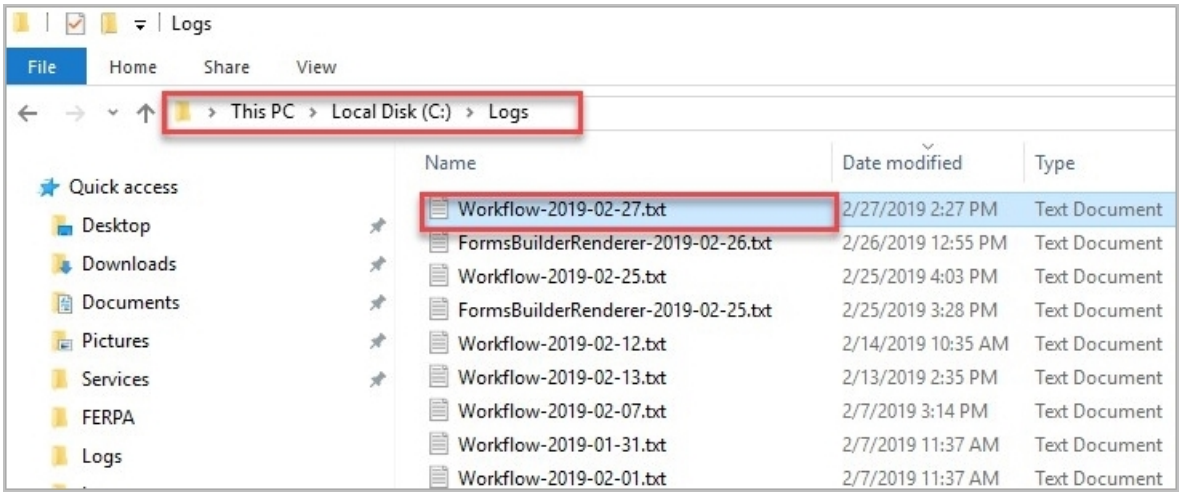
Workflow Composer

Log files for Workflow Composer are located on the server where Workflow is installed.

```
\\server\c$\logs
```

(where server is the hostname for your environment)

Look for **Workflow-date.txt**



Azure Cloud Subscriptions

Azure Blob Storage

Currently, all logs for the products in your subscription are found in the Azure blob storage container. For more information, refer to Azure Blob Storage.

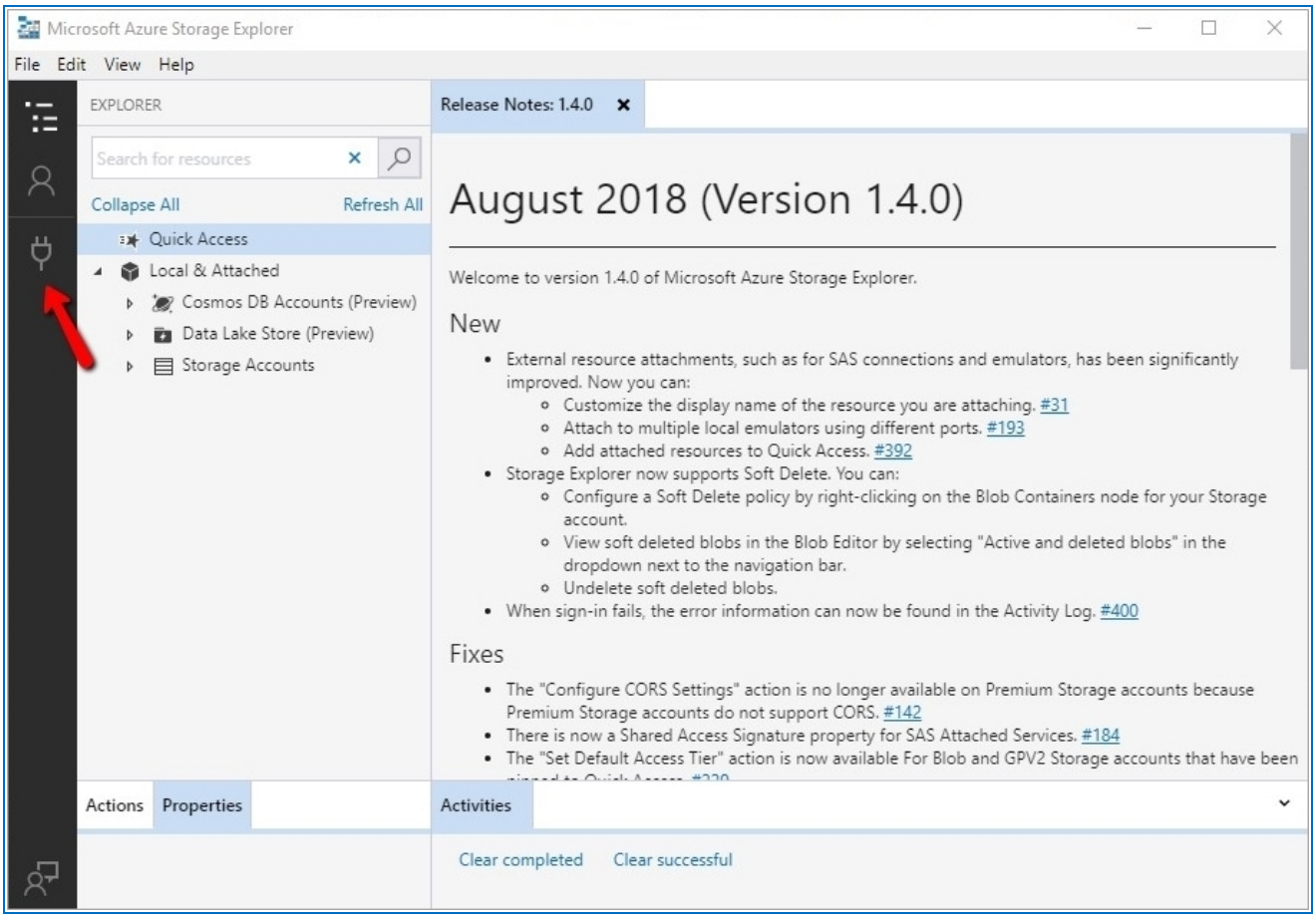
Azure Storage Explorer

Previously, for Azure customers, internal staff members accessed logs using Azure Storage Explorer. If the staff member had access to Last Pass, the storage account name and key could be retrieved and a storage account added to Azure Storage Explorer. If the staff member did not have access to Last Pass, the necessary information had to be provided by Client Services.

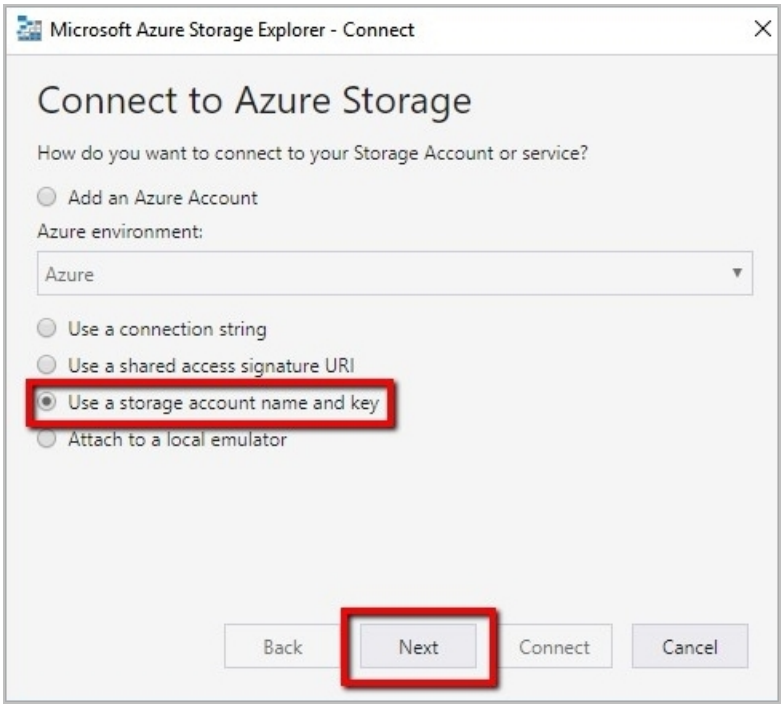
1. Download and install the Microsoft Azure Storage Explorer free version from the below link:

   https://azure.microsoft.com/en-us/features/storage-explorer/

2. Open Microsoft Azure Storage Explorer and select the **Plug** icon.



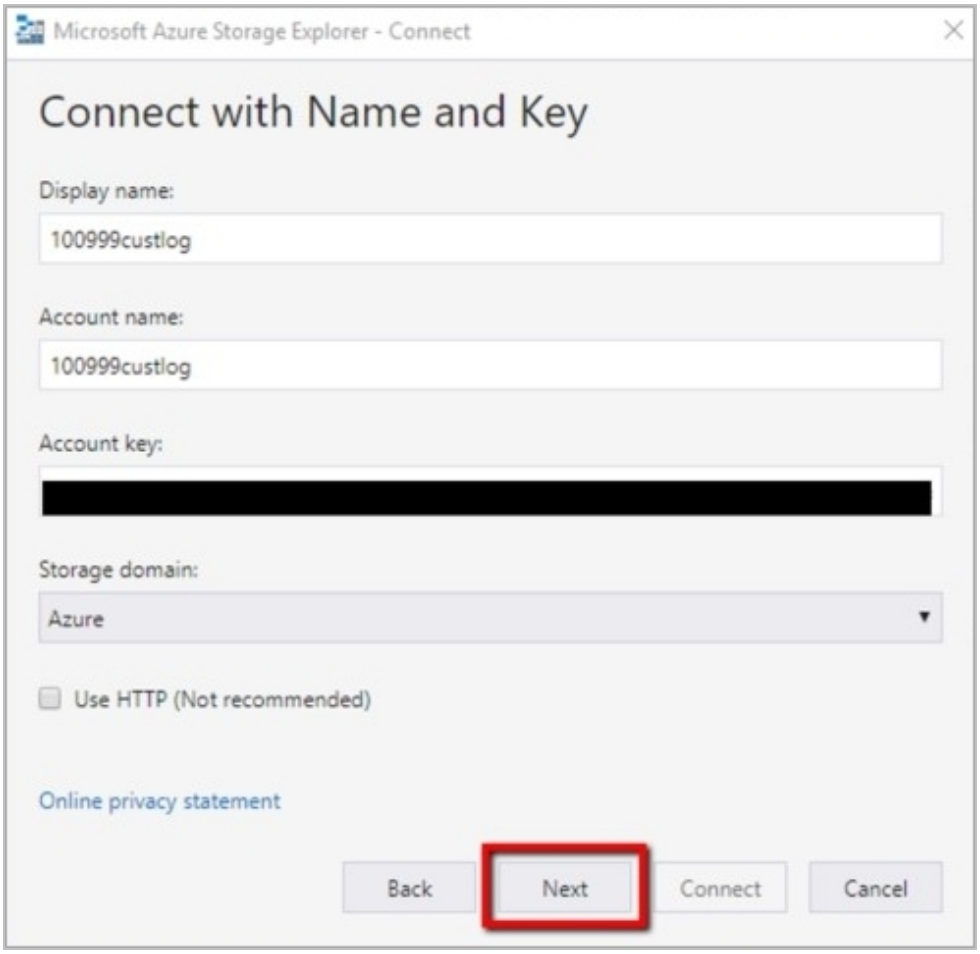
3. Select **Use a storage account name and key** and select **Next**.



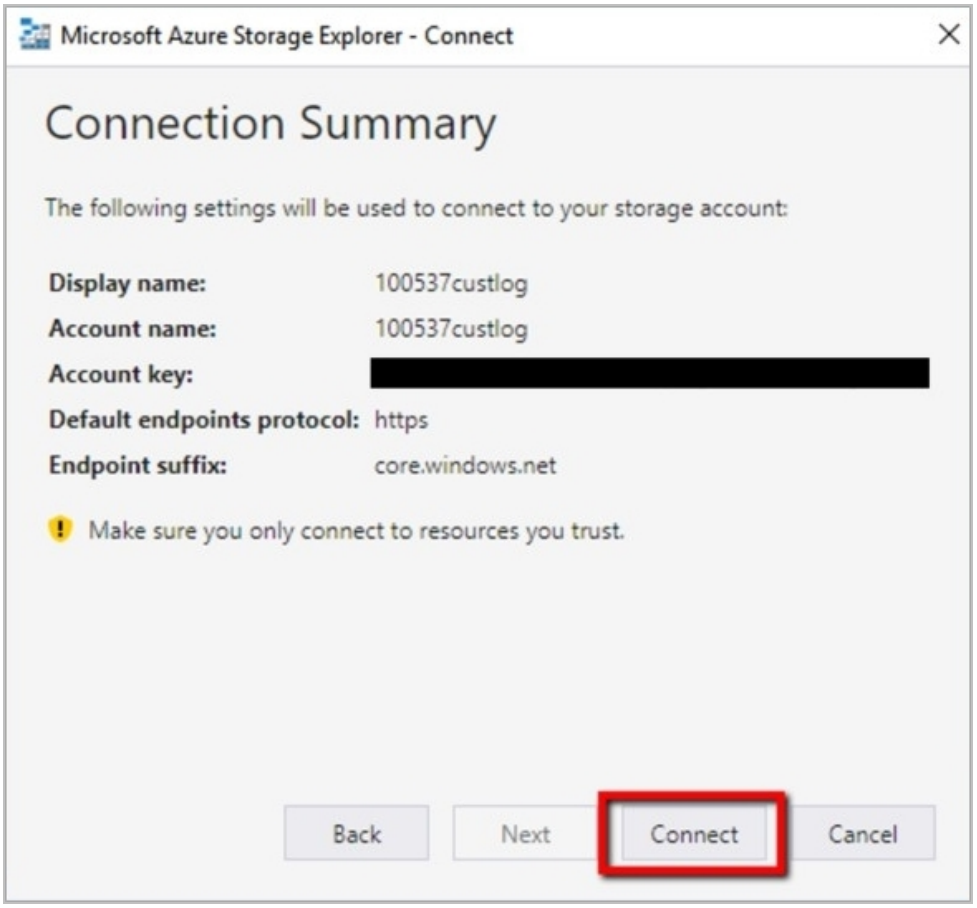
4. Enter the **Display Name**, **Account Name**, and **Account Key**.

   The Account Key information is located in LastPass.

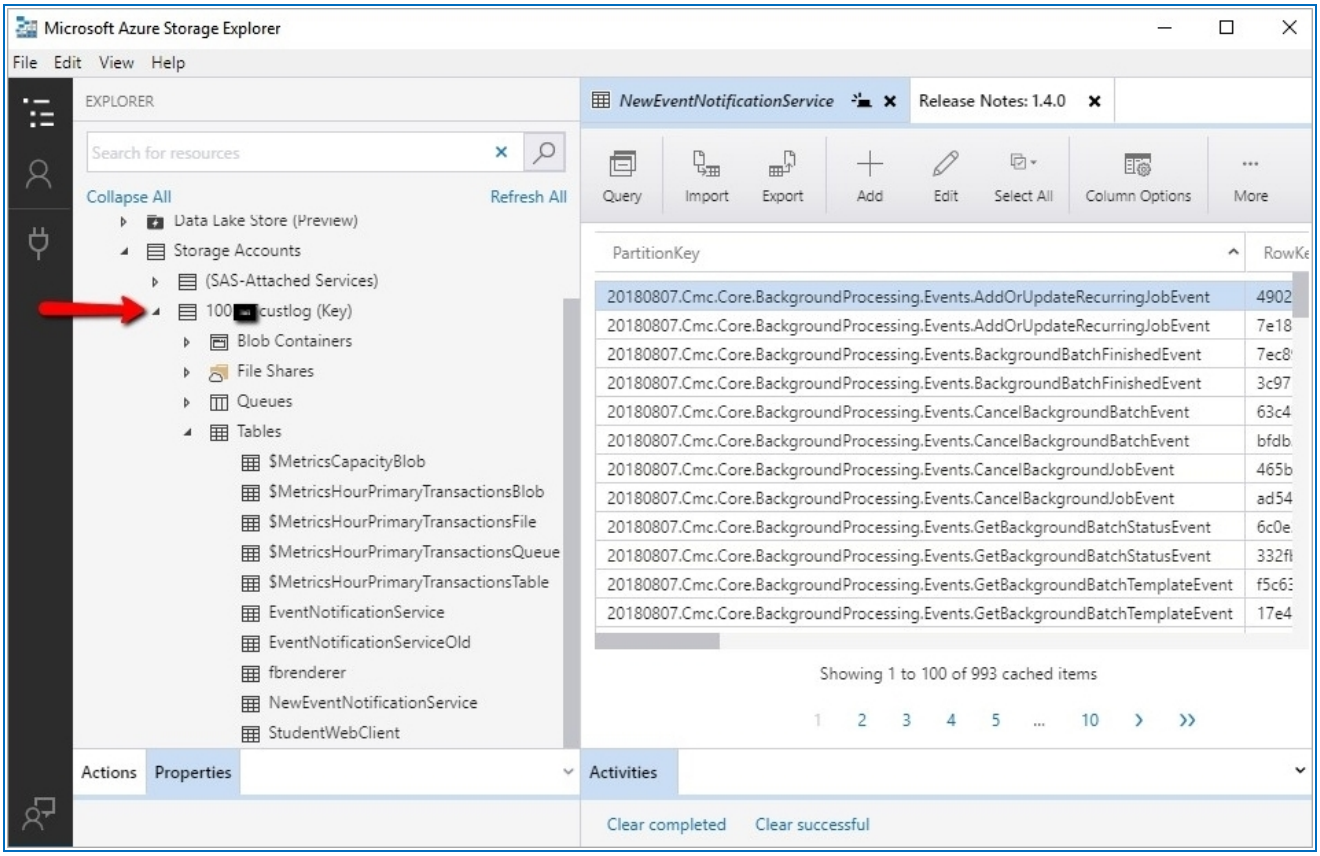
   Search for **CustomerID** and **custlog** all one word (e.g., 100999custlogs). In the notes field you will find the Account Key. Select **Next**.



5. On the next screen, select **Connect**.



6. After connecting, the Storage Account **100999custlog** and default set of tables are visible.



Tips

Best Practices for Logging

Log files may contain confidential information such as user names and passwords, account information, etc. It is your responsibility to protect sensitive user and system data.

To mitigate the risks of exposing sensitive data, observe the following best practices:

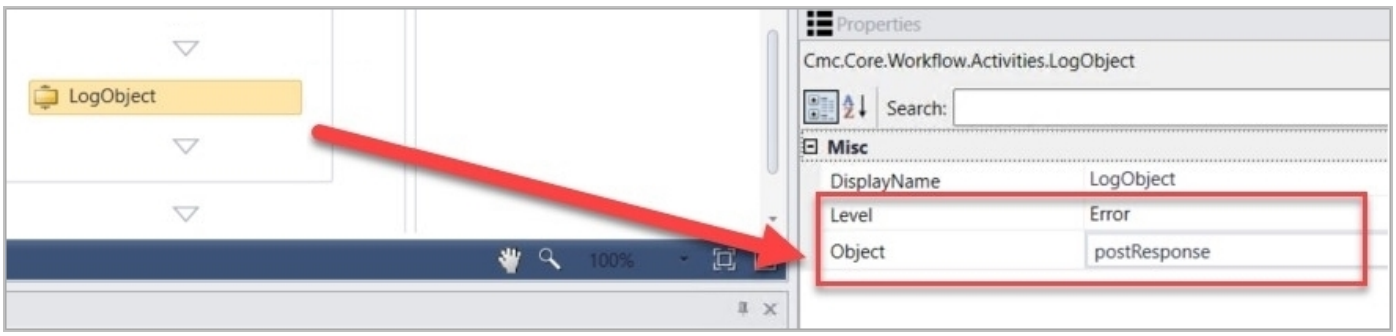
- Set the log level in production environments to the lowest, least detailed log level. Increase the log level in test environments only when needed. Reset the log level when testing is complete.
- The default logging provider used by Anthology products is NLog. NLog allows you to configure log targets, levels, rules, layouts, etc. through configuration. To configure logging, modify the NLog.config file in the application's executing directory. For Web applications, this file exists alongside the web.config file.
- When LogLine workflow activities are used to capture entities that contain sensitive information, remove such LogLine activities as soon as testing is complete.
- If, instead you followed the recommendations, and the development machine NLog minLevel is set to "Info" and all logging is done at the "Information" level, and the production machine NLog minLevel is set to "Error" (default), then nothing needs to be done because the production machine will not log "Information" LogLine or LogObject activities. The additional benefit is that the logging is still available if a problem can only occurs

in a production machine and lowering the NLog minLevel to “Info” temporarily (and restarting the app pool) will allow troubleshooting.

NLog Level	Hierarchy	Description
Fatal	1 - Only fatal messages will be logged.	A web service call has thrown an exception. The username, workflowdefintionid, and workflowinstanceid are logged.
Error	2 - Logs include the first two levels.	Any exceptions that are not being handled by the application.
Warn	3 - Logs include the first three levels.	Messages about potential oddities from which the application automatically recovers or about variable/property values that may be close to being out of the acceptable range.
Info	4 - Logs include the first four levels.	Reserved for customer logging in workflows (refer to LogLine Activities).
Debug	5 - Logs include the first five levels.	All additional Forms Builder logging.
Trace	6 - Logs include all messages.	Extensive additional Cmc.Core logging for use by developers.

LogObject

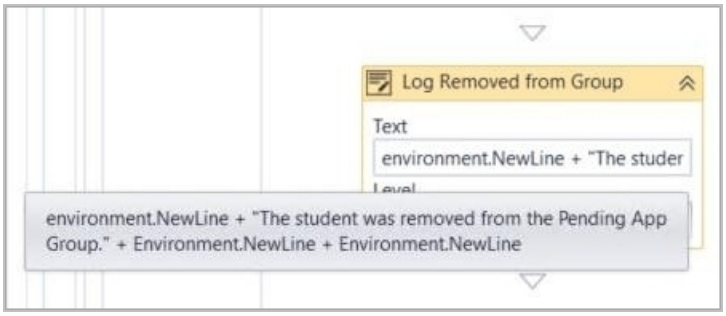
Use the LogObject activity in Workflow to log everything being created on an entity.




```
"PaymentProviderInfo": {
  "Partner": "Paypal",
  "MerchantCode": "rblackwood",
  "UserName": "rblackwood",
  "Password": "CampusNexus01",
  "PaymentGatewayUrl": "https://pilot-payflowpro.paypal.com",
  "HostedPageUrl": "https://pilot-payflowlink.paypal.com"
},
"TransactionId": "A10AACF3D182",
"CorrelationId": "408"
```

LogLine

Use the LogLine activity in Workflow to log data being retrieved and/or sent, log what step the worklow is on, etc.




```
 Enrolled the student.

2018-11-27 15:41:00.5649  64 Error                          Cmc.Core.Workflow.Activities.LogLine
The student was removed from the Pending App Group.


2018-11-27 15:41:00.8996  64 Error                          Cmc.Core.Workflow.Activities.LogLine
The termID is :  2180The Class Section ID is: 3260


2018-11-27 15:41:00.9564  64 Error                          Cmc.Core.Workflow.Activities.LogLine
The EnrollID for the student is:   3451The studentID is:    30098


2018-11-27 15:41:03.7725  64 Error                          Cmc.Core.Workflow.Activities.LogLine
The student was registered in the orientation course.


2018-11-27 15:41:03.9604  64 Error                          Cmc.Core.Workflow.Activities.LogLine
The student was added to the Freshman Registration Group.
```

CampusLink web.config File

Edited the web.config for the Cmc.CampusLink.WebServices.Wcf site to include the error text when there is a fault:

Change the includeExceptionDetailInFaults to 'true':

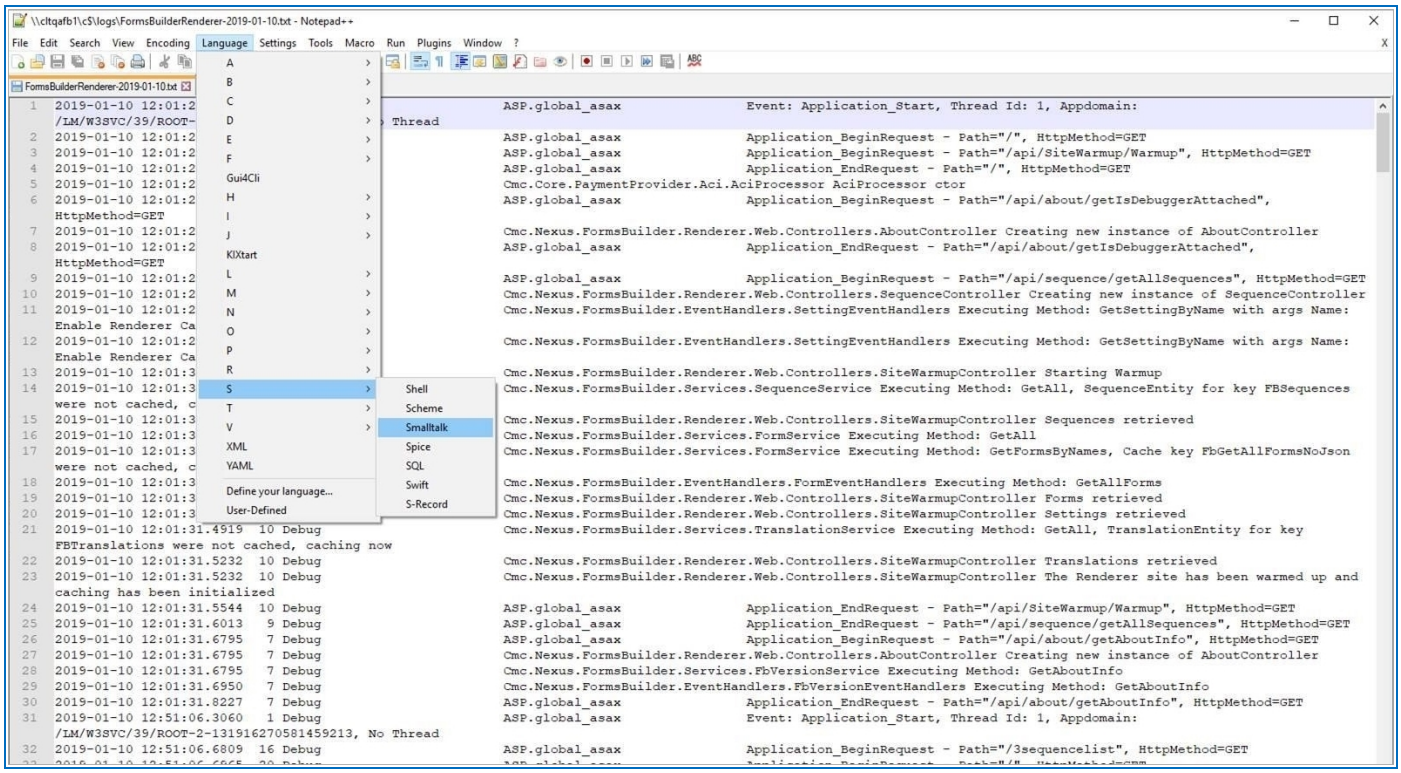
```
<behavior>
<serviceMetadata httpGetEnabled="true" httpsGetEnabled="true" />
<serviceDebug includeExceptionDetailInFaults="true" />
```

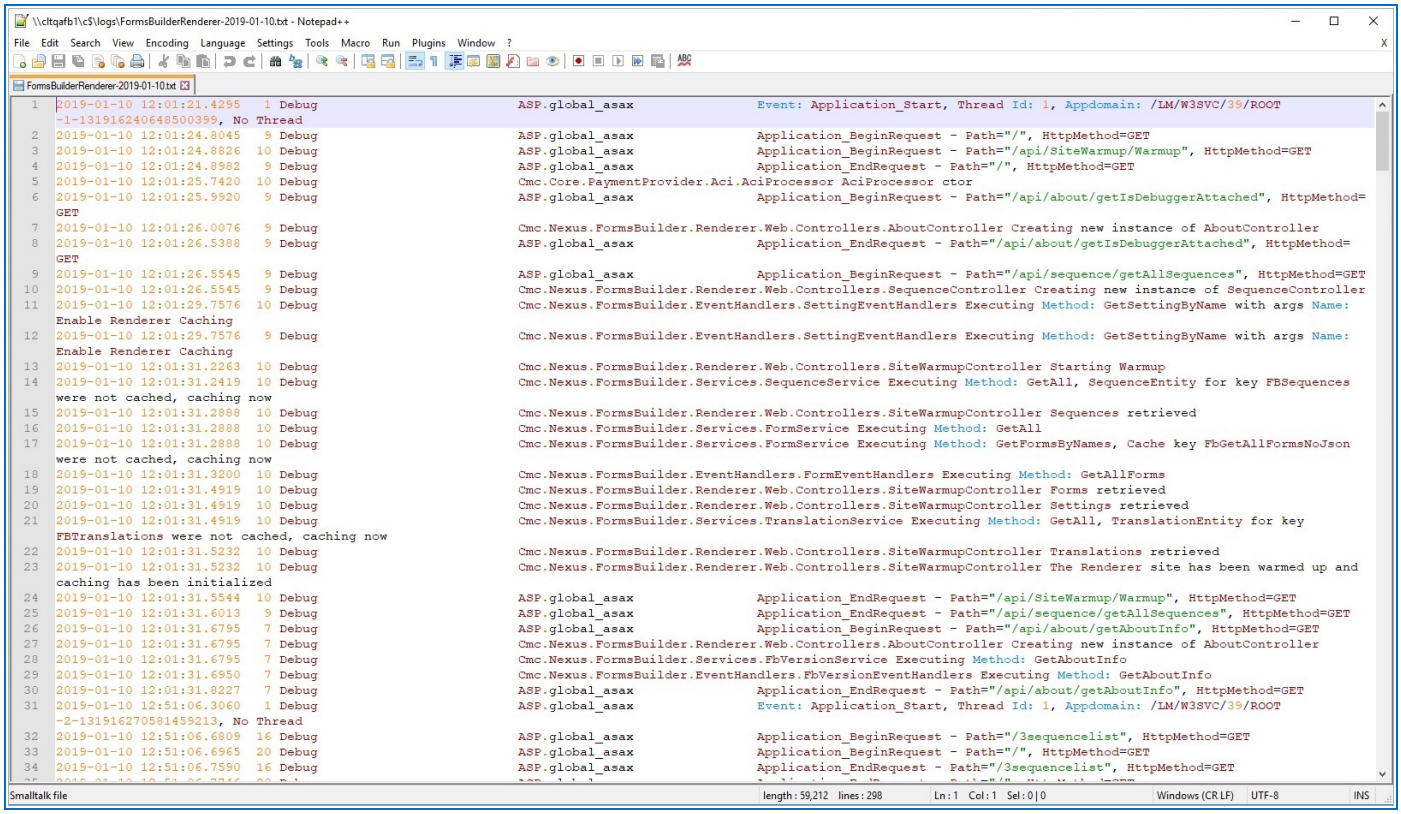
This will log additional information if using activities that call the CampusLink webservices.

For example: Term invalid.

Reading Log Files

If viewing the log files in Notepad++, change the language to Smalltalk and it highlights and displays the information where it is easier to read.


\\cltqafb1\c$\logs\FormsBuilderRenderer-2019-01-10.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
FormsBuilderRenderer-2019-01-10.txt
A
B
C
D
E
F
Gui4Cli
H
I
J
KIXtart
L
M
N
O
P
R
S
T
V
XML
YAML
Define your language...
User-Defined
Shell
Scheme
Smalltalk
Spice
SQL
Swift
S-Record


\\cltqafb1\c$\logs\FormsBuilderRenderer-2019-01-10.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
FormsBuilderRenderer-2019-01-10.txt
Smalltalk file

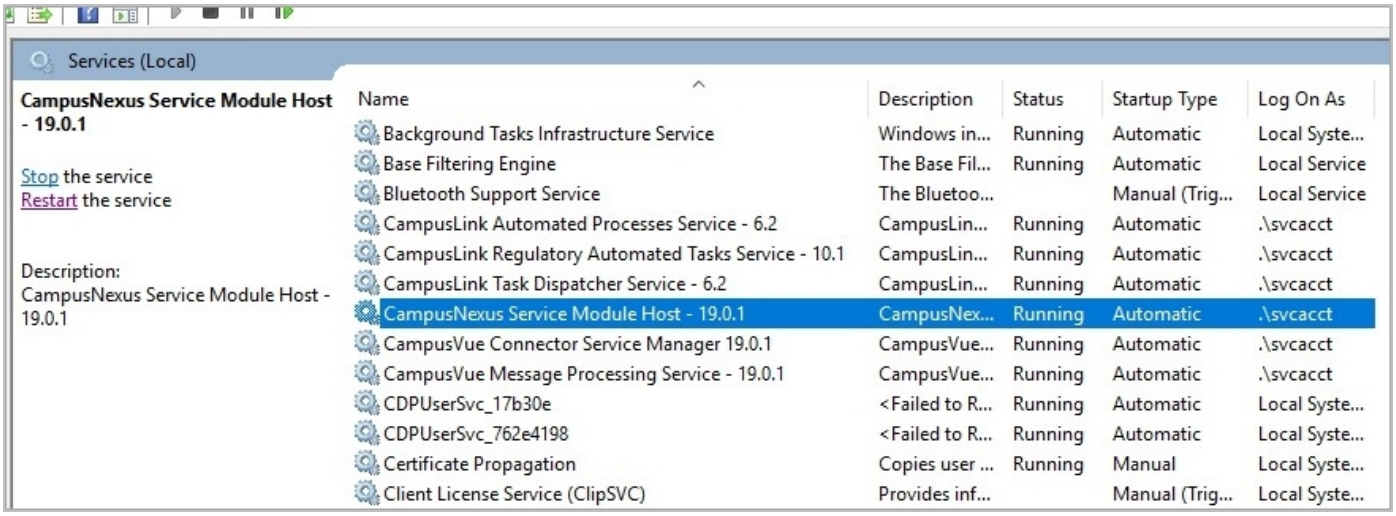
Service Module Host

The Service Module Host is the windows service that listens to and processes the messages in the Service Broker Queue. If a workflow has been published, it will execute when the event is raised and processed by the Service Module Host.

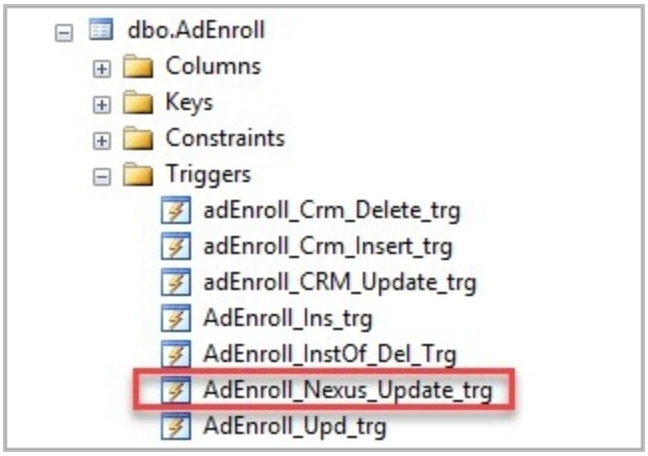
Triggers have been added to the main tables that events can be raised against. When anything occurs in the user interface that updates these tables then an entry is written to the Service Broker Queue and the Service Module Host sweeps the queue to determine if a published workflow applies to the record. If so, the business logic within the Workflow executes and processes the record. If not, the record is cleared from the queue.

Typically, the service is installed either on the API or COM server. Once installed it must be running even if no workflows have been created or published because the Nexus triggers are always inserting records into the Service Broker Queue and those records need to be processed.

Service Module Host



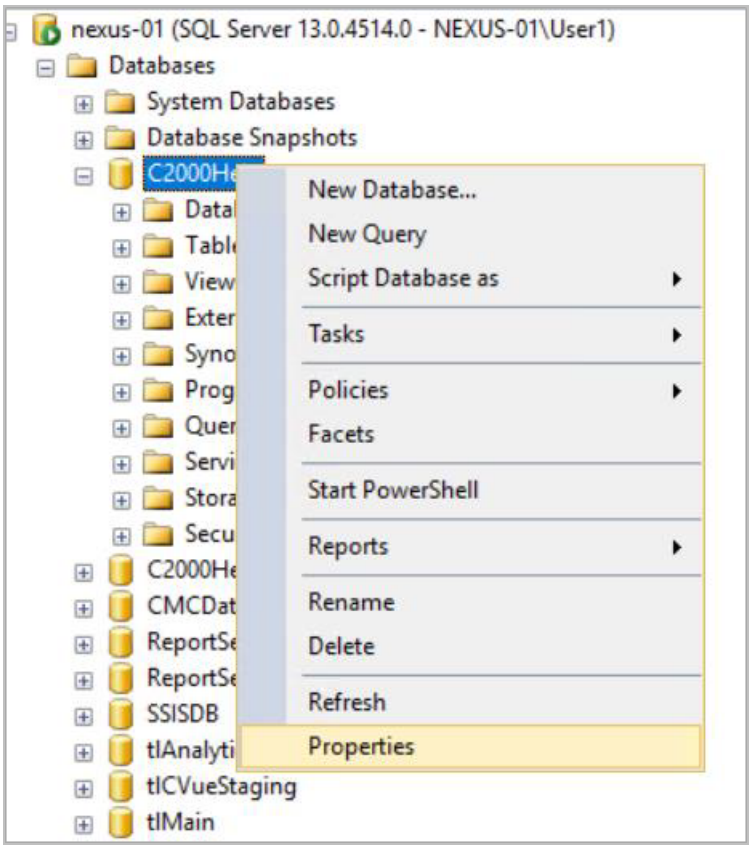
Nexus Triggers



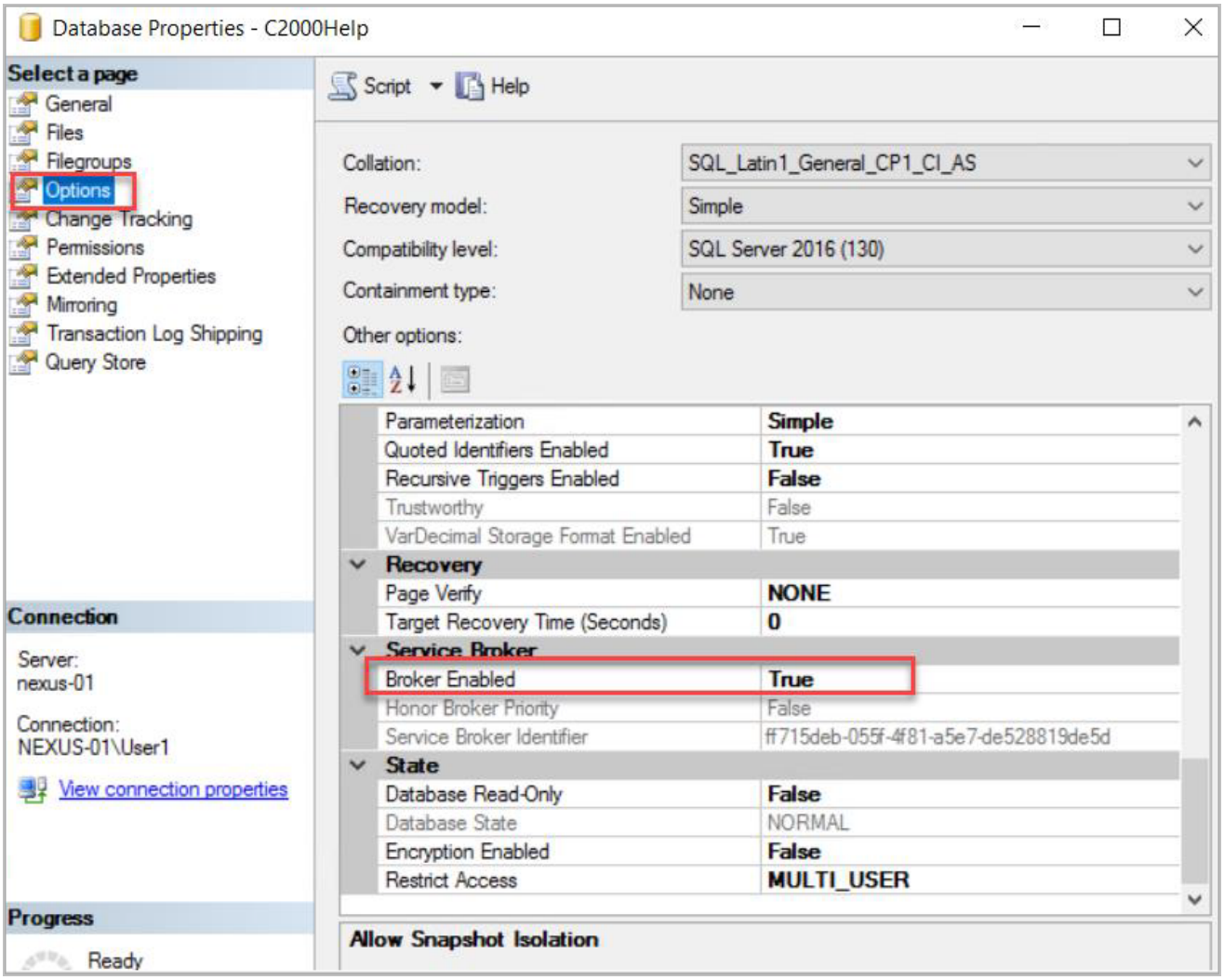
Service Broker Queue

The Service Broker Queue must be Enabled. To check if it is enabled, perform the following steps:

1. In SQL, right-click the database and go to **Properties**.



2. Select Options and scroll down in the window to **Service Broker**. The **Broker Enabled** property should read **True**.

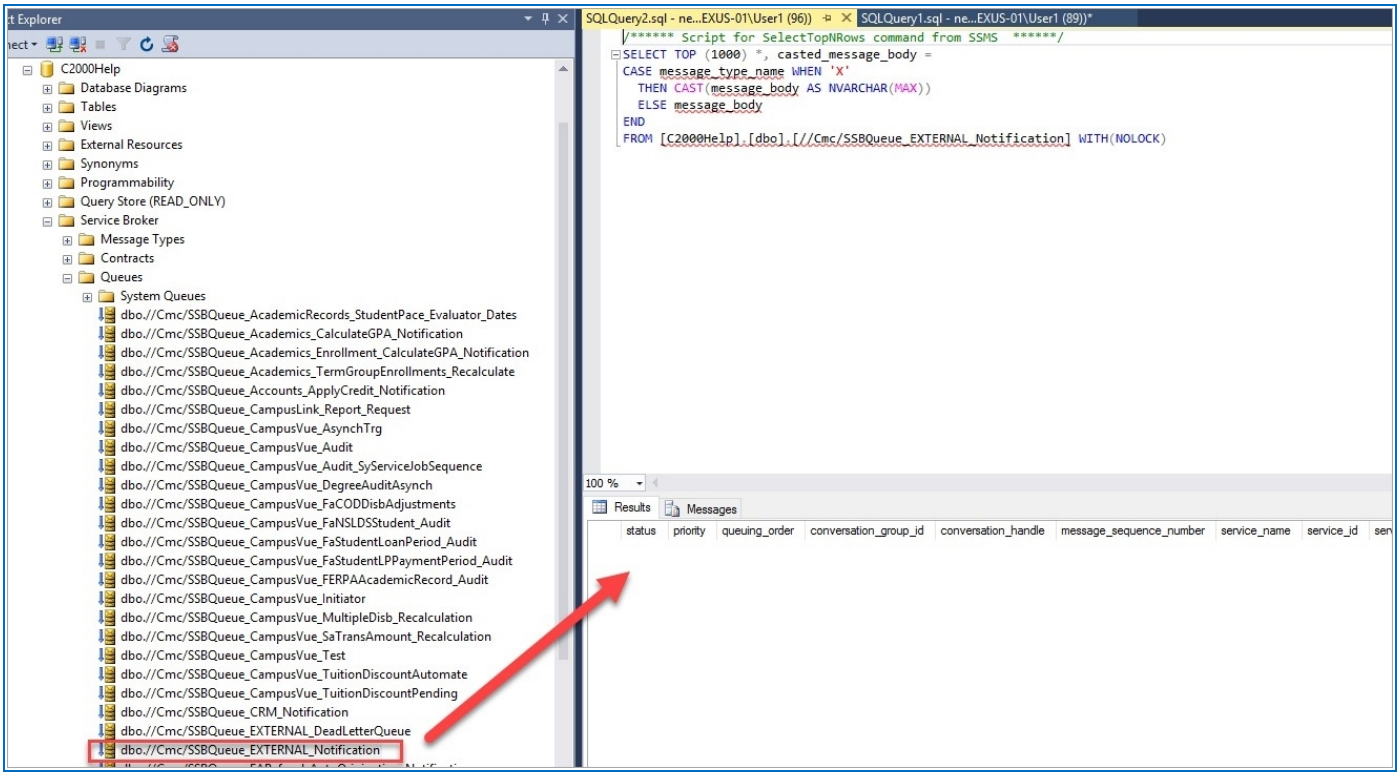


3. You can also use the query below to determine if the Service Broker is enabled. If the Broker Enabled property is False, run the below scripts to enable the broker.

```
--Check to see if Service Broker is Enabled
SELECT is_broker_enabled
FROM sys.databases
WHERE database_id = DB_ID()

--Enable Service Broker
alter database c2000Help set single_user with rollback immediate
ALTER DATABASE c2000Help SET ENABLE_BROKER WITH ROLLBACK IMMEDIATE
ALTER DATABASE c2000Help SET MULTI_USER WITH ROLLBACK IMMEDIATE
```

4. You can query the Service Broker External Notification queue as shown below in order to verify that the queue is empty.



The queue should always be empty or close to empty as the Service Module Host should always be running and will clear out the queue immediately. If the queue has many rows then check to make sure the Service is running.

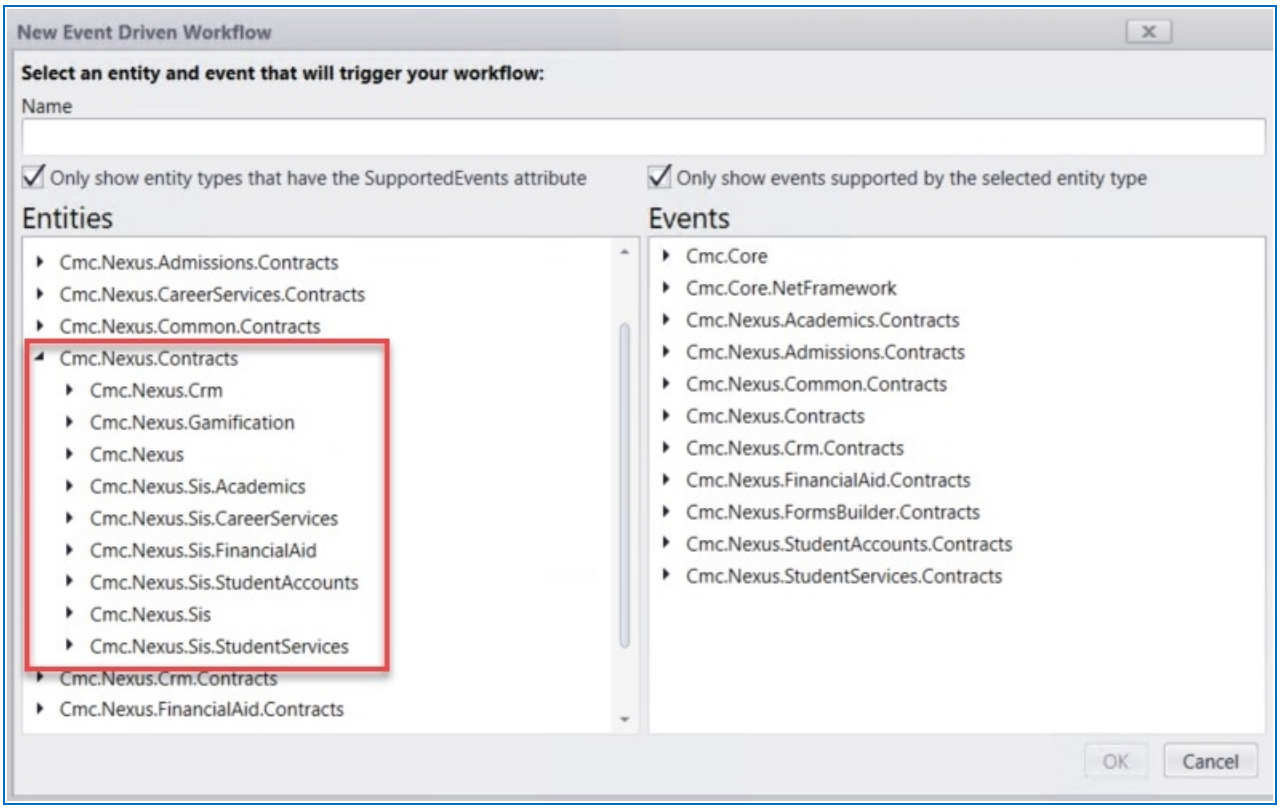
> ⚠ Never bring a customer live on a workflow until the queue is empty. If a workflow is published and the queue is populated, once the Service is started it will try to process each row against a published workflow. This could cause student records to have actions triggered that should not be. Start the service and let the queue empty and then publish the Workflow.

When is Service Module Host Used?

When creating a new Workflow in Workflow Composer, in the list of Entities a user can view the available Contracts and select the applicable Entity/Service along with the corresponding triggering Event. The list is populated once the Packages are imported into the Workflow Composer. When importing the packages into the composer, there will be V1 Contracts and Activities and V2 Contracts and Activities.

V1 Contracts

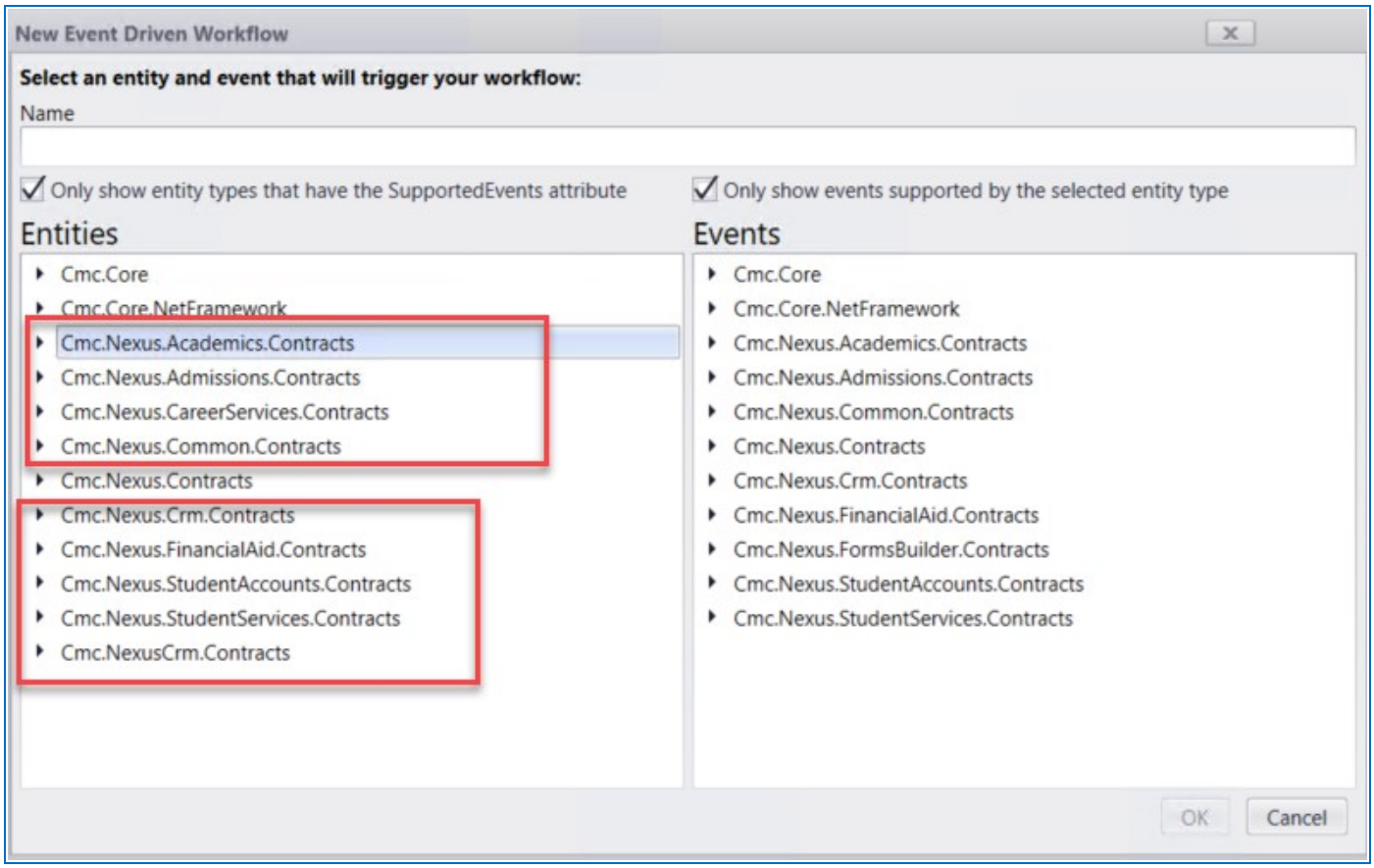
The V1 Contracts are referred to as the "old namespacing" before the Nexus re-architecture occurred. These Contracts are listed under Cmc.Nexus.Contracts. All workflows created using the V1 Contracts and Event of type Saved will always use the Service Module Host. The triggers for V1 saved events are at the database level so such workflows will be executed whether 'Save' is performed on Web Client, Anthology Student (Desktop Client), or CampusLink.



V2 Contracts

The V2 Contracts are referred to as the "new namespacing" which was introduced with the Nexus re-architecture. These Contracts are listed by module or area in the Entities list. All workflows created using the V2 Contracts and Event of type Saved will call the services directly and will not use the Service Module Host.

If in a V2 Contract Workflow, there is a V1 or V2 activity that is saving a Course for example and there is also a V1 Student Course Saved Event Workflow published in the environment, then the V2 Contract will execute and the V1 Student Course Saved Event Workflow will execute under the Service Module Host.



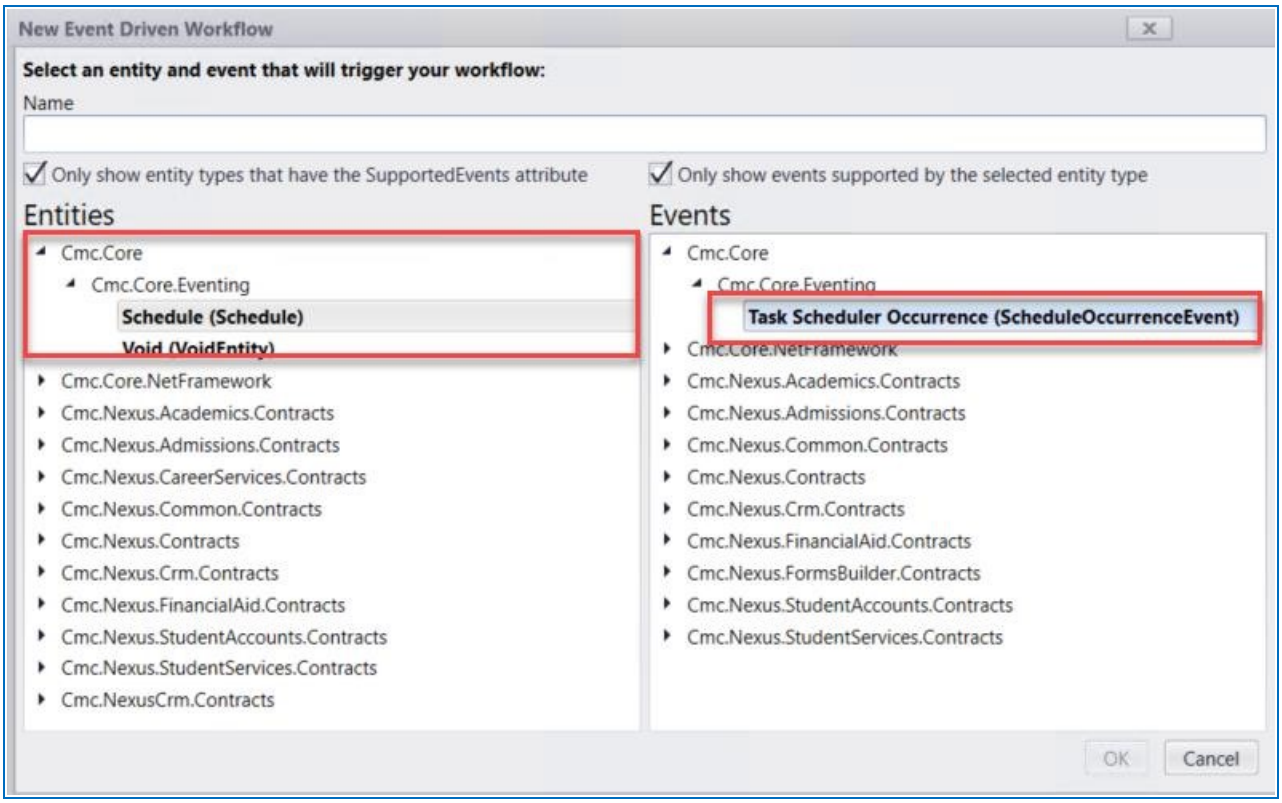
Forms Builder

Forms Builder works as a workflow engine and calls the services directly from within Forms Builder. The Service Module Host is not relied upon for any activity used in the Forms Builder workflow.

> If in a Forms Builder workflow, there is a V2 activity that is saving a Course for example and there is also a V1 Student Course Saved Event workflow published in the environment, then the Forms Builder workflow will execute and the V1 Student Course Saved Event workflow will execute under the Service Module Host.

Task Scheduler Occurrence Event

A Scheduled Workflow relies upon the scheduled job and does not require the Service Module Host.

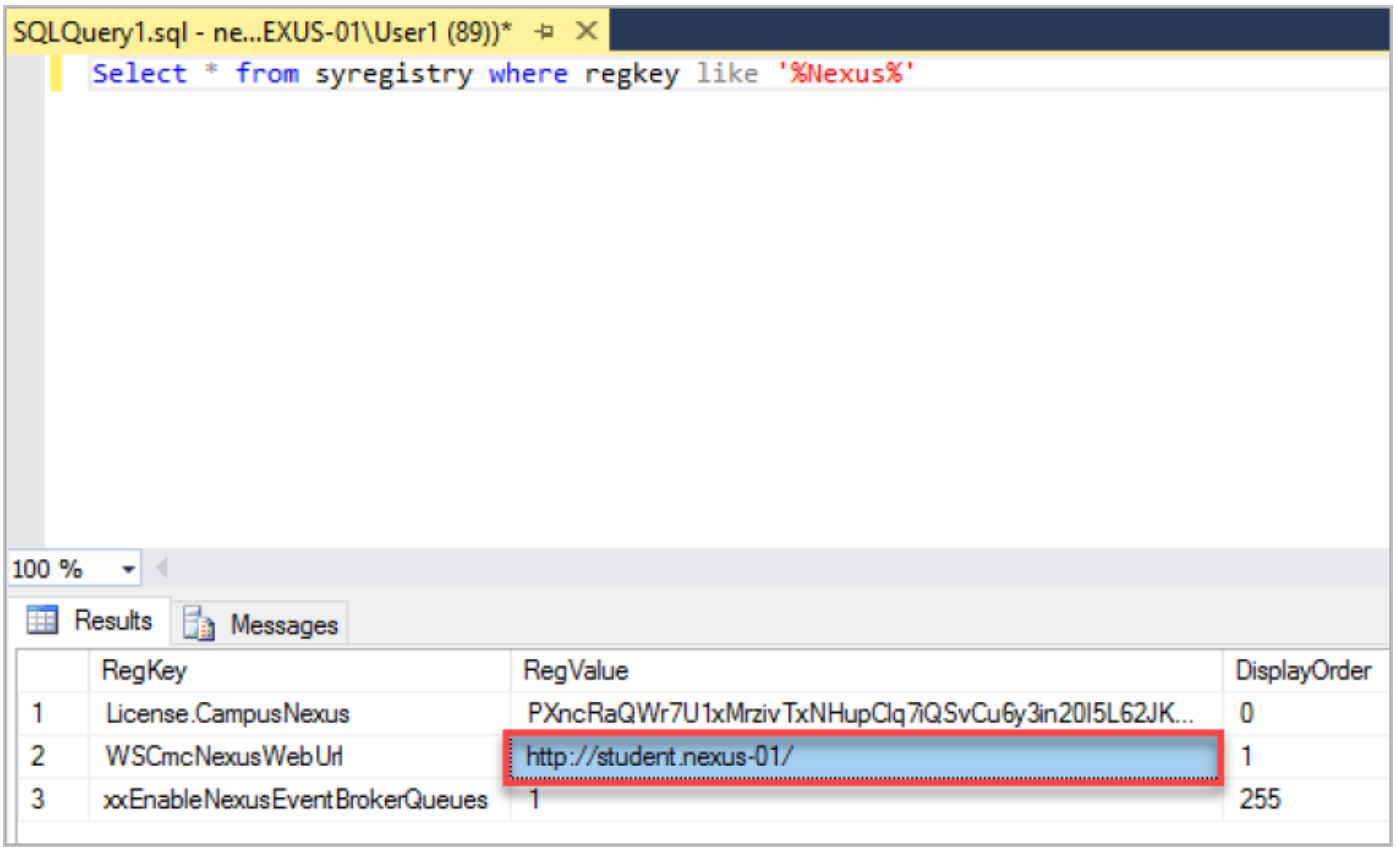

New Event Driven Workflow
Select an entity and event that will trigger your workflow:
Name
Only show entity types that have the SupportedEvents attribute
Only show events supported by the selected entity type
Entities
Cmc.Core
Cmc.Core.Eventing
Schedule (Schedule)
Void (VoidEntity)
Cmc.Core.NetFramework
Cmc.Nexus.Academics.Contracts
Cmc.Nexus.Admissions.Contracts
Cmc.Nexus.CareerServices.Contracts
Cmc.Nexus.Common.Contracts
Cmc.Nexus.Contracts
Cmc.Nexus.Crm.Contracts
Cmc.Nexus.FinancialAid.Contracts
Cmc.Nexus.StudentAccounts.Contracts
Cmc.Nexus.StudentServices.Contracts
Cmc.NexusCrm.Contracts
Events
Cmc.Core
Cmc.Core.Eventing
Task Scheduler Occurrence (ScheduleOccurrenceEvent)
Cmc.Core.NetFramework
Cmc.Nexus.Academics.Contracts
Cmc.Nexus.Admissions.Contracts
Cmc.Nexus.Common.Contracts
Cmc.Nexus.Contracts
Cmc.Nexus.Crm.Contracts
Cmc.Nexus.FinancialAid.Contracts
Cmc.Nexus.FormsBuilder.Contracts
Cmc.Nexus.StudentAccounts.Contracts
Cmc.Nexus.StudentServices.Contracts
OK
Cancel

Forms Builder Access Errors

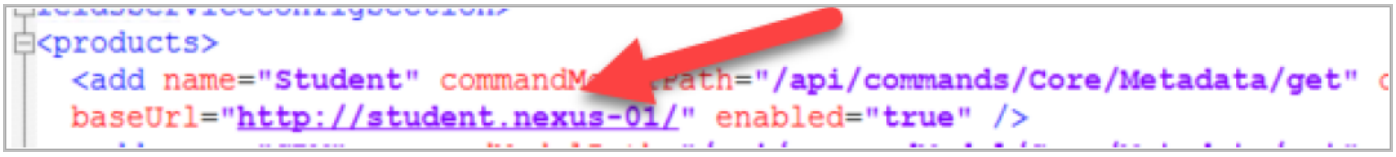
Web Client URL

If there is trouble running an authenticated form, check the Web Client URL in SyRegistry. The URL in the SyRegistry table must match exactly the URL that is stored in the web.config file for Forms Builder Renderer and Designer. This may also cause issues in Workflow Composer if the URL does not match.

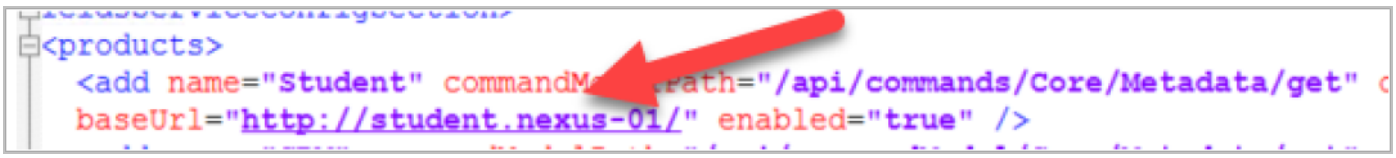
Syregistry query



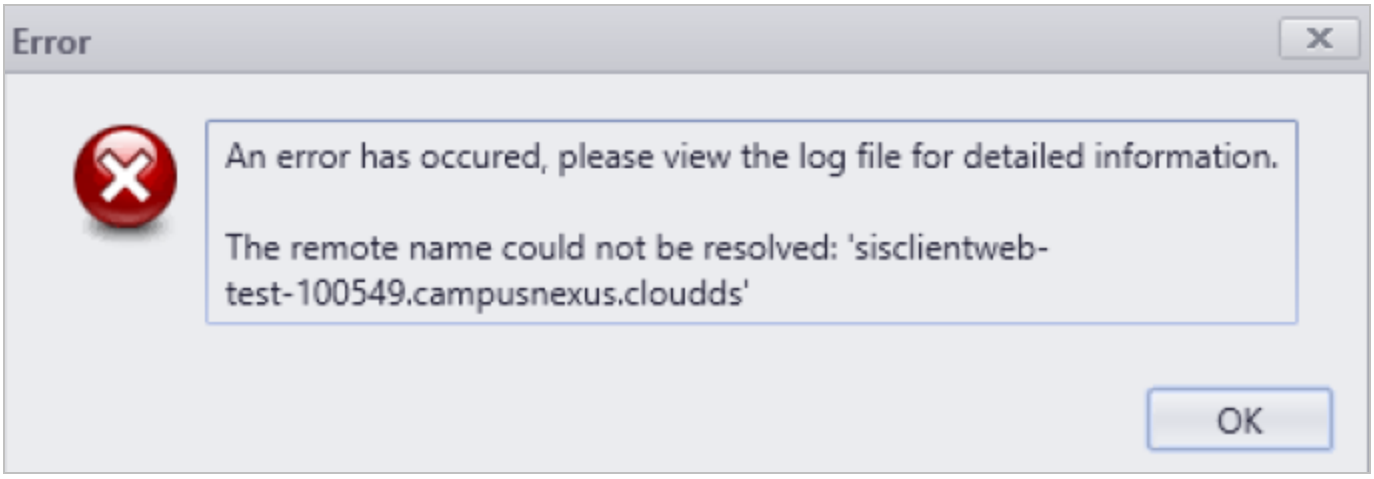
Forms Builder Renderer Config file

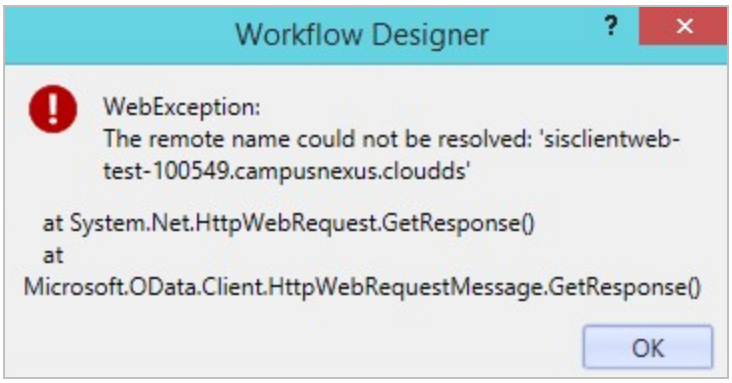


Forms Builder Designer Config file



Example Workflow Composer Error

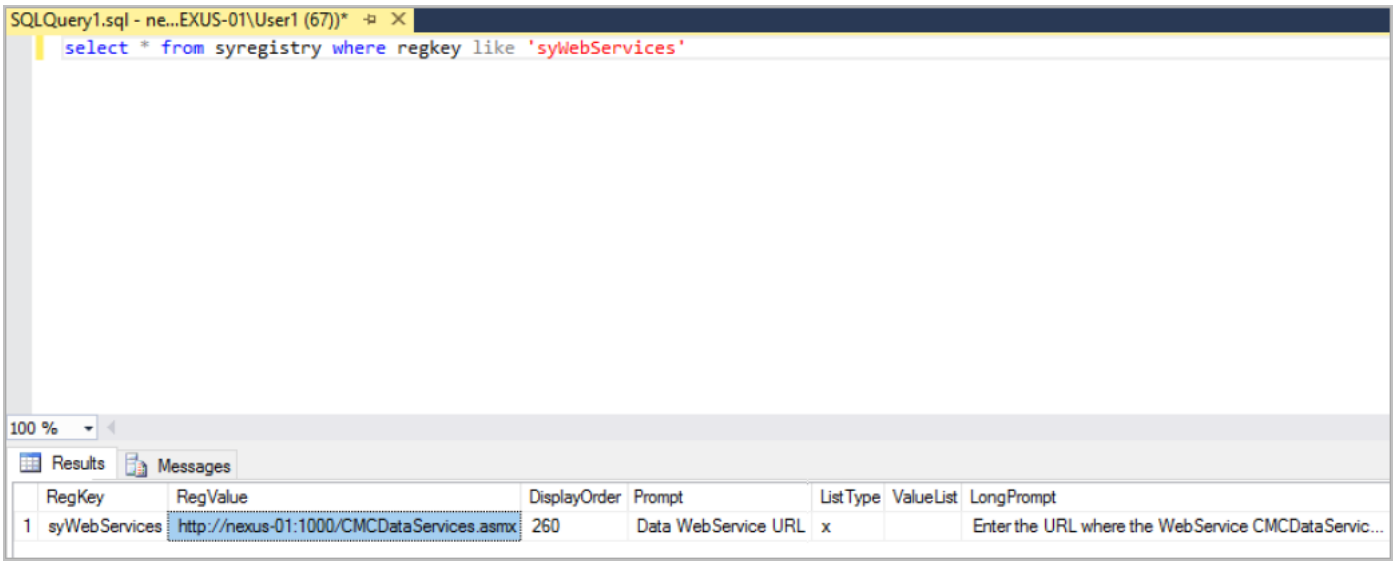




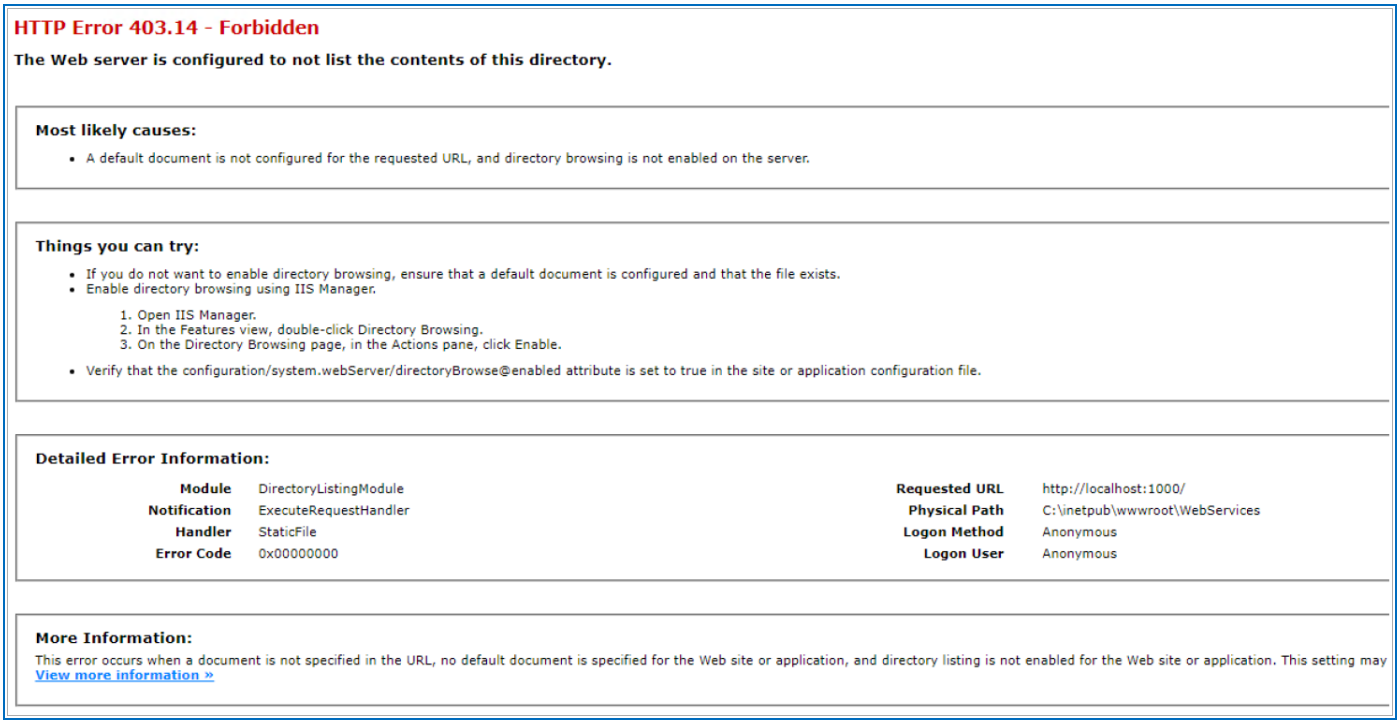
CMCDataServices URL

If getting an error when launching a Forms Builder authenticated form and being redirected to Portal STS or trying to launch portal, the URL in syregistry for syWebServices needs to be updated to point to the CMCDataServices.asmx URL. Below is the query and the errors that may occur for Forms Builder and Portal.

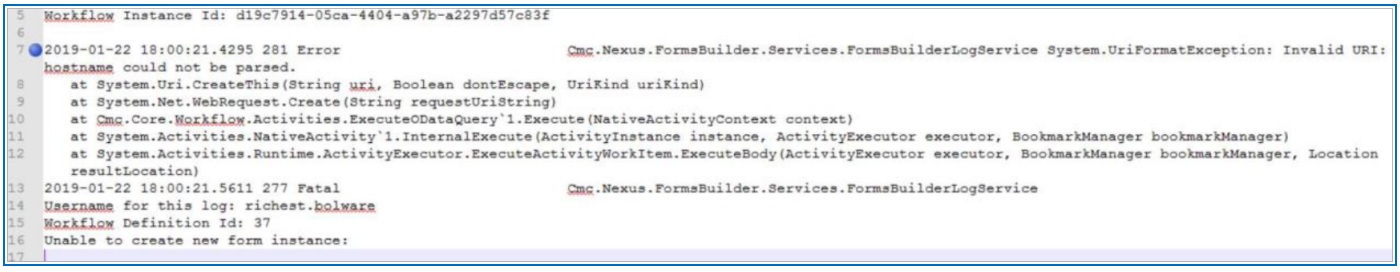
Syregistry query



Portal Error



Forms Builder Error



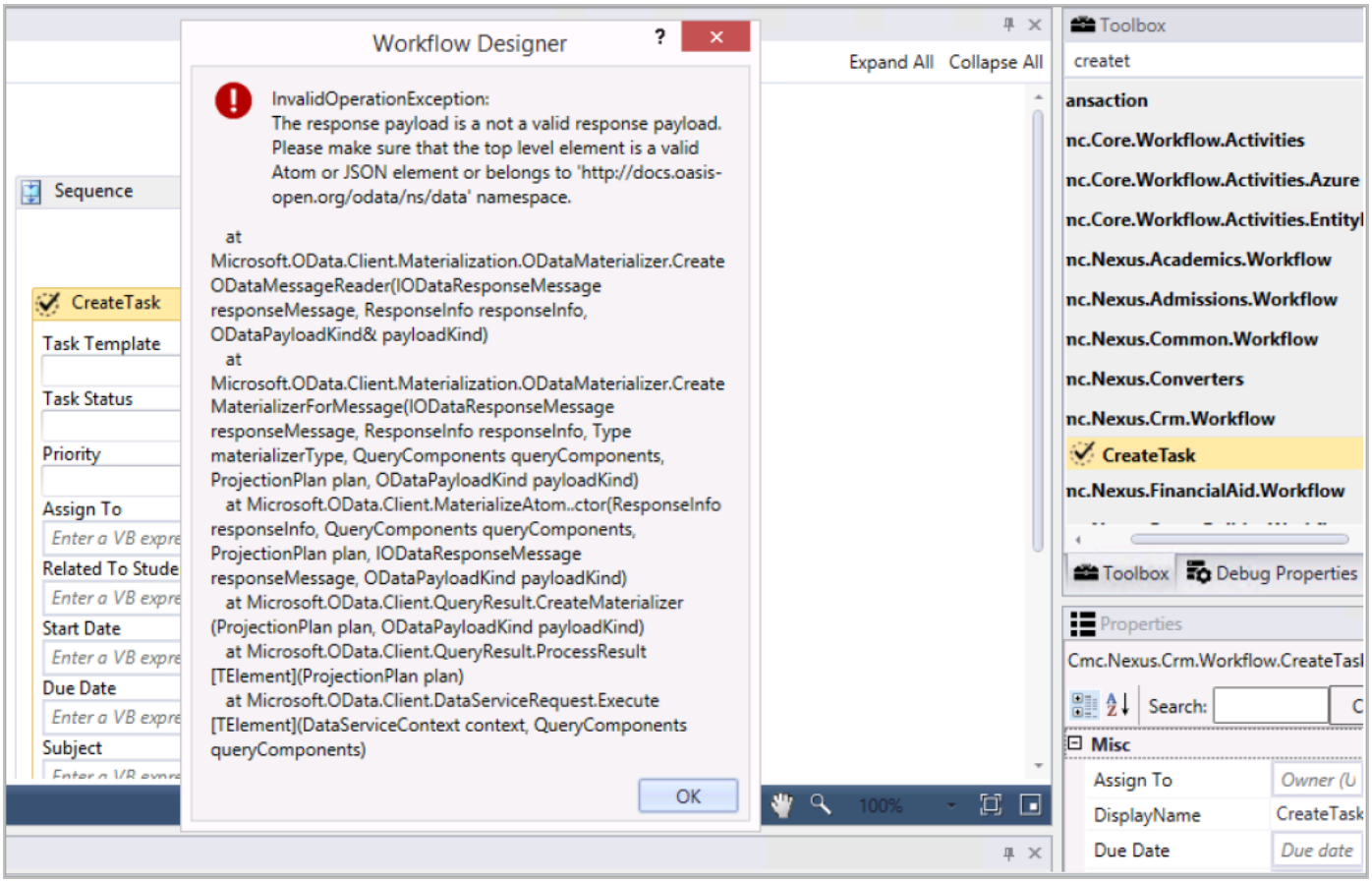
Activity Errors

If receiving errors when dragging activities into Workflow Composer, check the following:

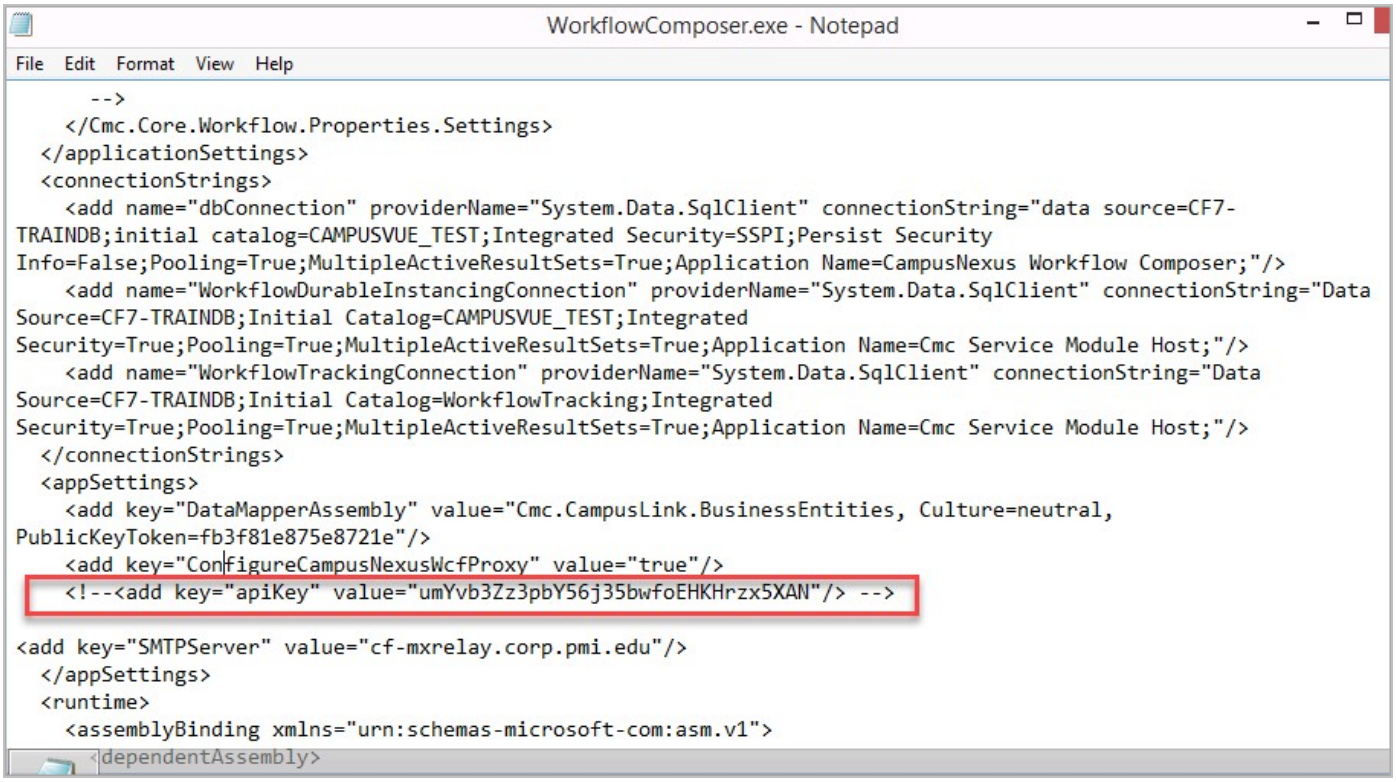
- API Key Accuracy across all products. If customer is on prior version of products that do not have API Key but new version of Workflow that does have API Key, comment the API Key out in Workflow Composer config file.
- Base URL for Web Client in SyRegistry is correct.
- Make sure the API Password is correct.
- Make sure the API User has correct Activity and Document Policy and correct permissions in Anthology Student.

Example

Error when dragging V2 activity CreateTask in Designer of Workflow Composer



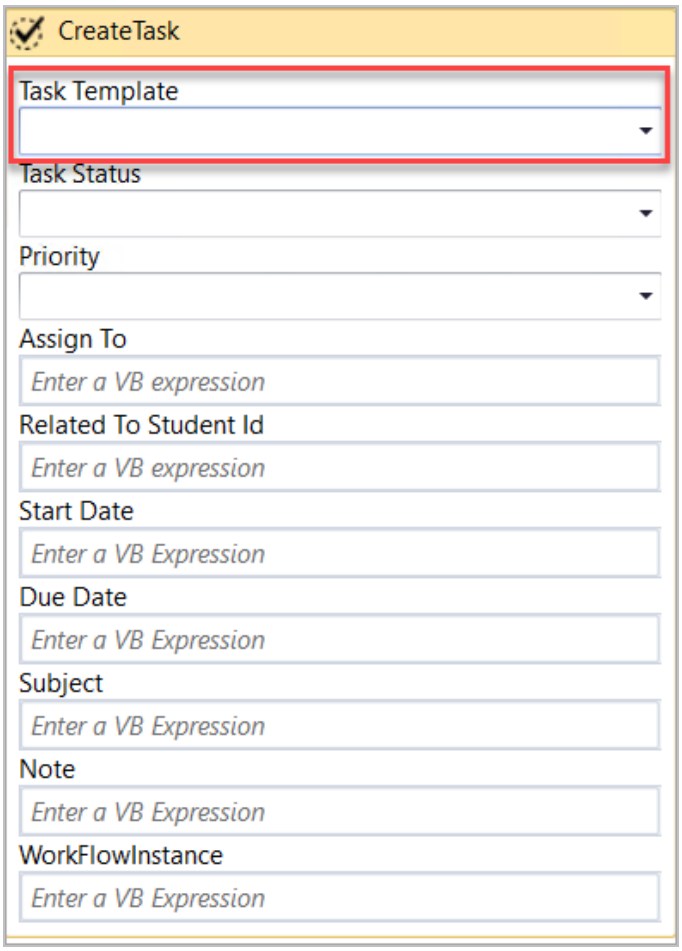
Comment out the API Key in the Workflow Composer Config File. If Workflow Composer is on a version that has the API Key in the Workflow Composer Config File but Anthology Student is on a version below 19.0, then this error occurs:



Example

Error when dragging a V2 activity, such as CreateTask, in Designer of Workflow Composer and there are no values in the drop-down for the Task Template

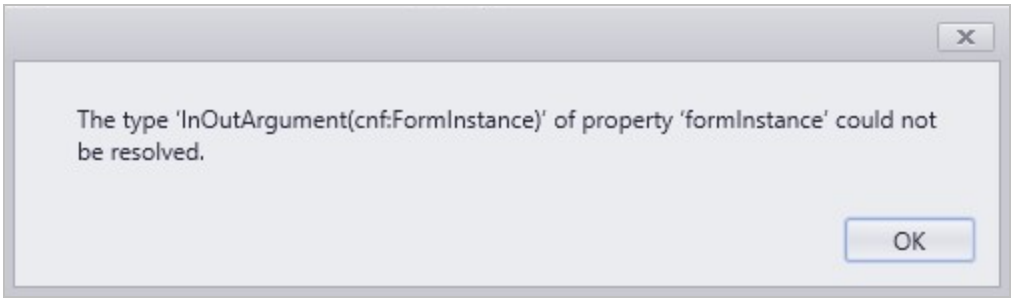
Ensure the API user has the proper Activity policy assigned in Anthology Student:


CreateTask
Task Template
Task Status
Priority
Assign To
Enter a VB expression
Related To Student Id
Enter a VB expression
Start Date
Enter a VB Expression
Due Date
Enter a VB Expression
Subject
Enter a VB Expression
Note
Enter a VB Expression
WorkFlowInstance
Enter a VB Expression

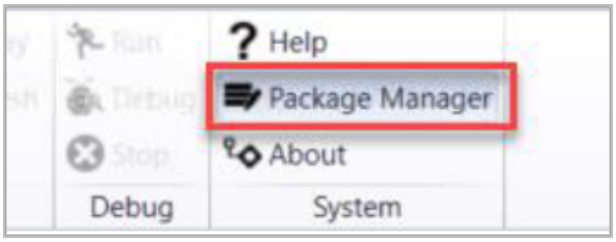
Configuration Issues

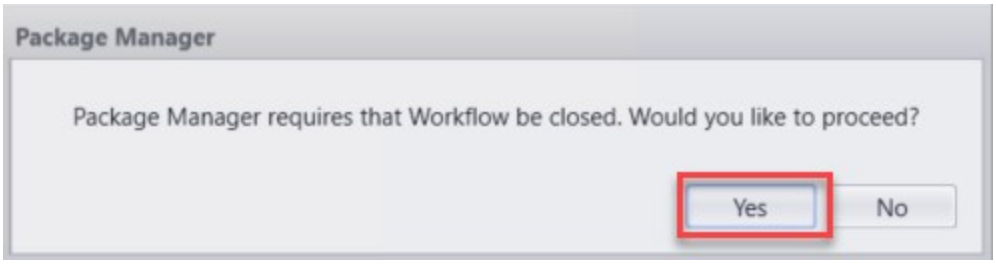
Packages

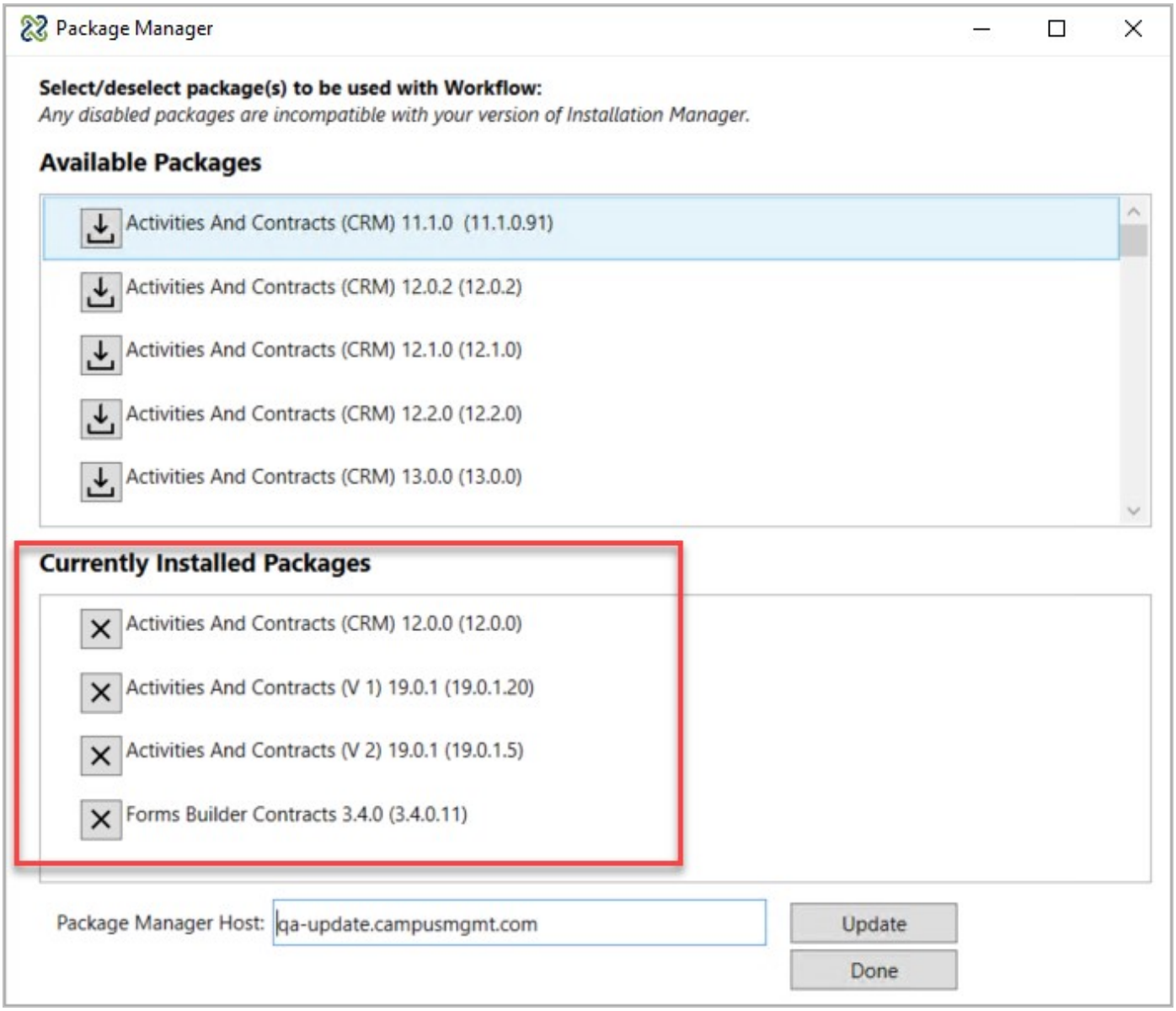
If you try to open the workflow definition for a sequence and a message similar to the following is displayed, the Forms Builder Contracts packages have not been installed in Workflow Composer. All current packages need to be imported into the Workflow Composer for Anthology Student, CampusNexus CRM, and Forms Builder based on the version the customer is on.



Import Packages







Make sure the correct versions of the packages are installed. Install V1 and V2 Activities and Contracts for Anthology Student.

Connection Strings

If querying other products from within Workflow, make sure the Connection String Information exists and/or is correct in the following config files:

- Renderer web.config file
- CampusNexus Service Module Host config file
- Workflow Composer config file

```
<connectionStrings>

  <!--CampusVue Database connection string needed for ServiceModule and Composer to access Service Broker queues AND for Csla Access to DB -->
  <add name="dbConnection" providerName="System.Data.SqlClient" connectionString="Data Source=Nexus-01,1433;Initial Catalog=C2000Help;Integrate

      <!--CampusVue Database connection string needed for ServiceModule and Composer to access Service Broker queues AND for Csla Access to DB
  <add name="PortalConnection" providerName="System.Data.SqlClient" connectionString="Data Source=Nexus-01,1433;Initial Catalog=C2000Help_Porta

--CampusVue Database connection string needed for ServiceModule and Composer to access Service Broker queues AND for Csla Access to DB -->
  <add name="CRMConnection" providerName="System.Data.SqlClient" connectionString="Data Source=Nexus-01,1433;Initial Catalog=tlMain;Integrated

      <!-- Connection String for Database containing the Workflow Persistence tables (may be the same as the CVue database, but could be a diff
      <add name="WorkflowDurableInstancingConnection" providerName="System.Data.SqlClient" connectionString="Data Source=Nexus-01,1433;Initial

  <!-- Workflow Tracking Database.  Should not be the same as the CampusVue database. -->
  <add name="WorkflowTrackingConnection" providerName="System.Data.SqlClient" connectionString="Data Source=Nexus-01,1433;Initial Catalog=Workf

</connectionStrings>
```

Email Configuration

If using the Email Activity within a Workflow, the Email Configuration in the config files needs to be accurate or an error of "Unable to send email" will be received in the log file.

```
    <mailSettings>
      <smtp deliveryMethod="Network">
        <network clientDomain="" host="smtpout.campusmgmt.com" userName="" password="" port="25" enableSsl="FALSE"
defaultCredentials="FALSE" />
      </smtp>
    </mailSettings>
```

This setting is in the Workflow Composer config file, Renderer config file, and Service Module Host config file.

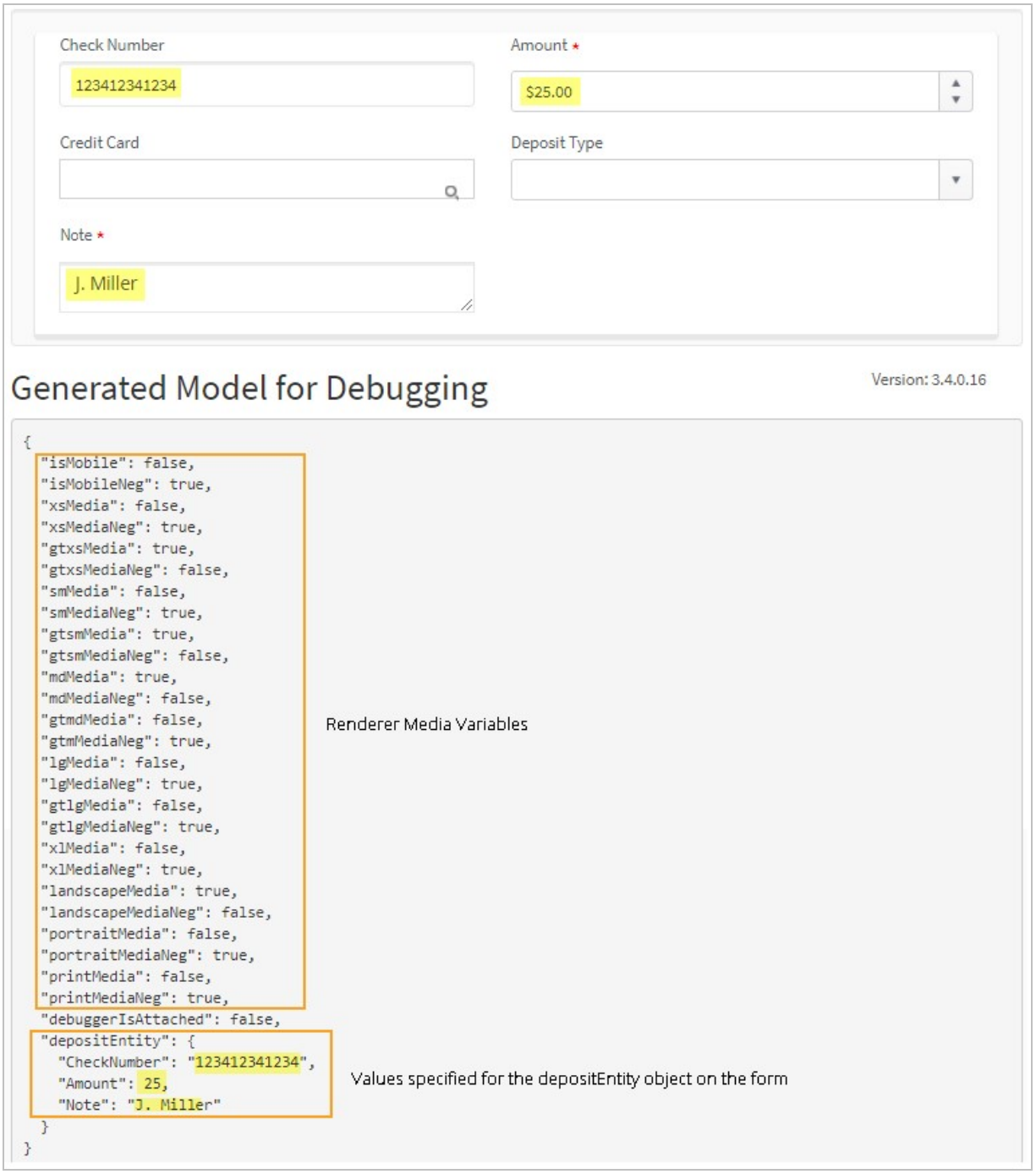
JSON Debug

When the "Debug - Show Generated JSON Model" option is enabled in the Forms Builder Settings workspace, additional data that shows the values for objects on the form will be shown at the bottom of each rendered form. This data can be helpful for troubleshooting, especially for complex components on a page where knowing the data that is available to a workflow during a transition will aid in debugging a workflow.

On form load, the Generated Model for Debugging section shows the Renderer Media Variables. As the form fields are populated with values, the debugging section displays the values associated with each object on the form, i.e., all model entity data on the page and new values entered are displayed in real time.

Notes:

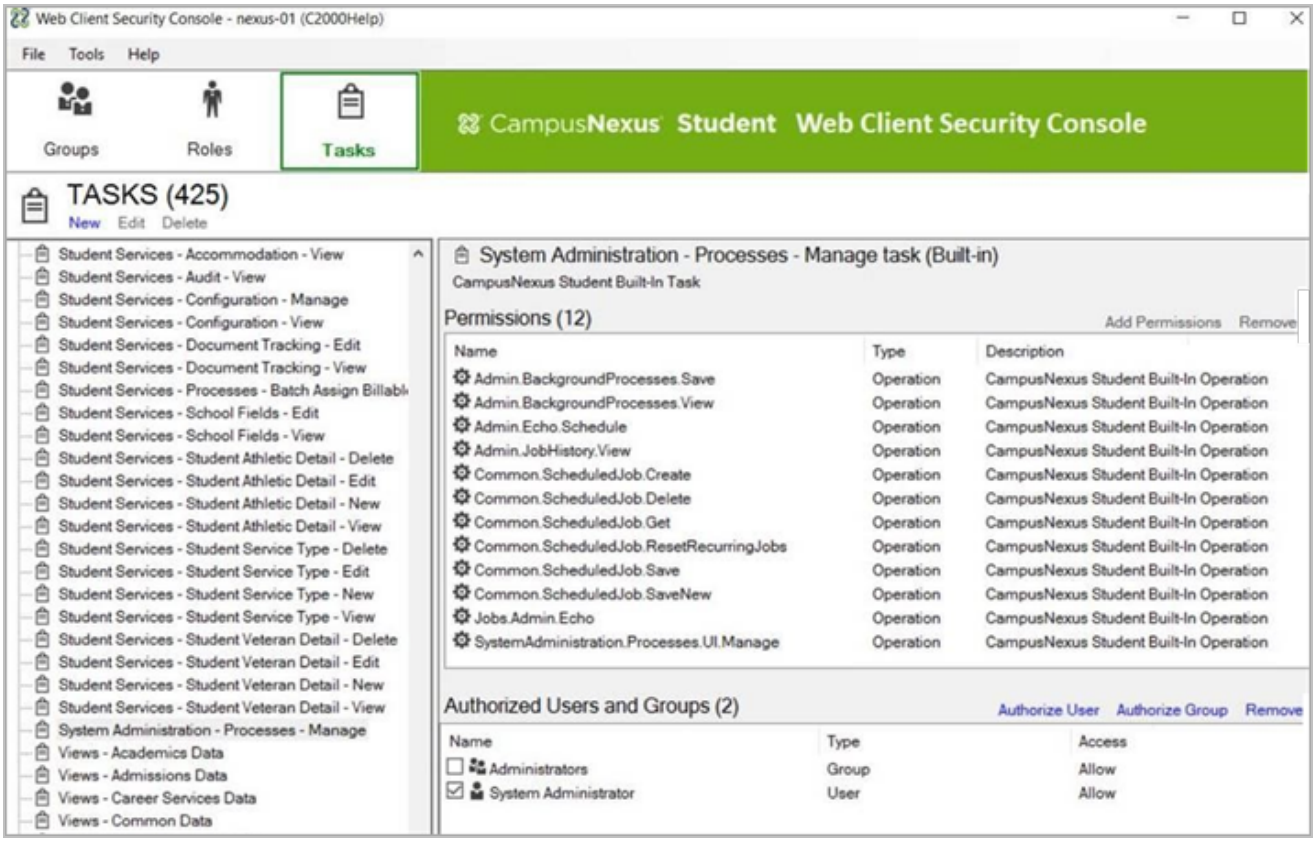
- On form transition, the entity data is serialized, and the debugging section will present all fields (not just the fields completed for the entity in the form).
- A createEntity or getEntity activity in the workflow will also cause all data fields of an entity to be included in the debugging section.


Check Number
123412341234
Amount *
$25.00
Credit Card
Deposit Type
Note *
J. Miller
Generated Model for Debugging
Version: 3.4.0.16
{
"isMobile": false,
"isMobileNeg": true,
"xsMedia": false,
"xsMediaNeg": true,
"gtxsMedia": true,
"gtxsMediaNeg": false,
"smMedia": false,
"smMediaNeg": true,
"gtsmMedia": true,
"gtsmMediaNeg": false,
"mdMedia": true,
"mdMediaNeg": false,
"gtmdMedia": false,
"gtmMediaNeg": true,
"lgMedia": false,
"lgMediaNeg": true,
"gtlgMedia": false,
"gtlgMediaNeg": true,
"xlMedia": false,
"xlMediaNeg": true,
"landscapeMedia": true,
"landscapeMediaNeg": false,
"portraitMedia": false,
"portraitMediaNeg": true,
"printMedia": false,
"printMediaNeg": true,
"debuggerIsAttached": false,
"depositEntity": {
"CheckNumber": "123412341234",
"Amount": 25,
"Note": "J. Miller"
}
}
Renderer Media Variables
Values specified for the depositEntity object on the form

Workflow Execution

Task Scheduler Occurrence Event

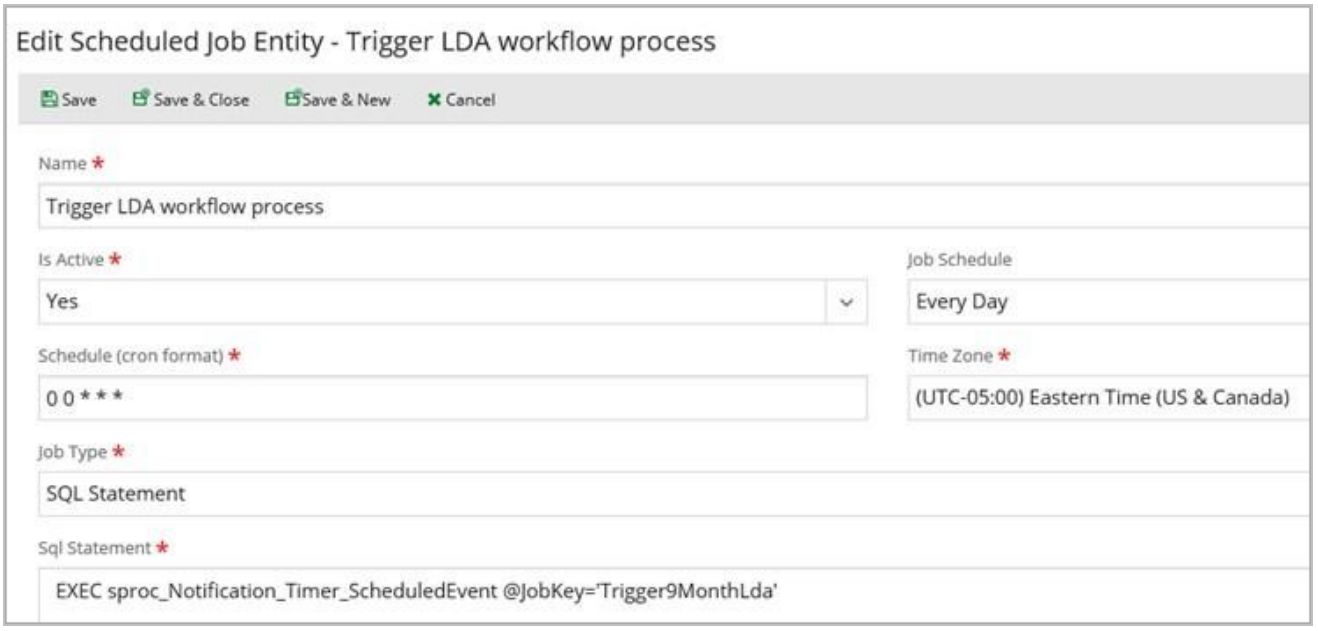
1. Sign into Anthology Student with an Administrator User Id. Specifically, a user with NSA permissions to the **Admin.BackgroundProcesses.View** and **Admin.BackgroundProcesses.Save** operations.



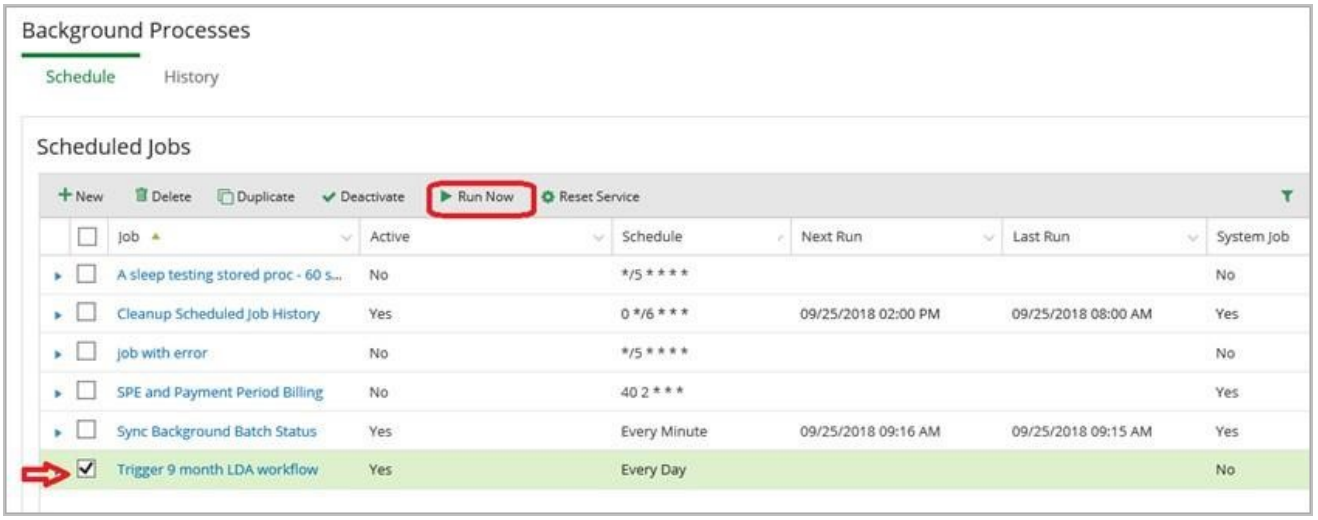
2. Go to: **Processes → [System Administraton] → Background Processes**.
3. Select **New**.
4. Enter the job parameters similar to what is shown below in the screenshot.

   Note the **cron schedule** – you can select from one of the pre-defined "Job Schedule" entries in the drop down, or enter your own, custom cron string – cron string reference.

   The schedule below will execute the job every day at midnight.



5. You can test the job by selecting **Run Now** after the Scheduled Job has been saved, and it is selected in the grid:



6. View the results by expanding the row:

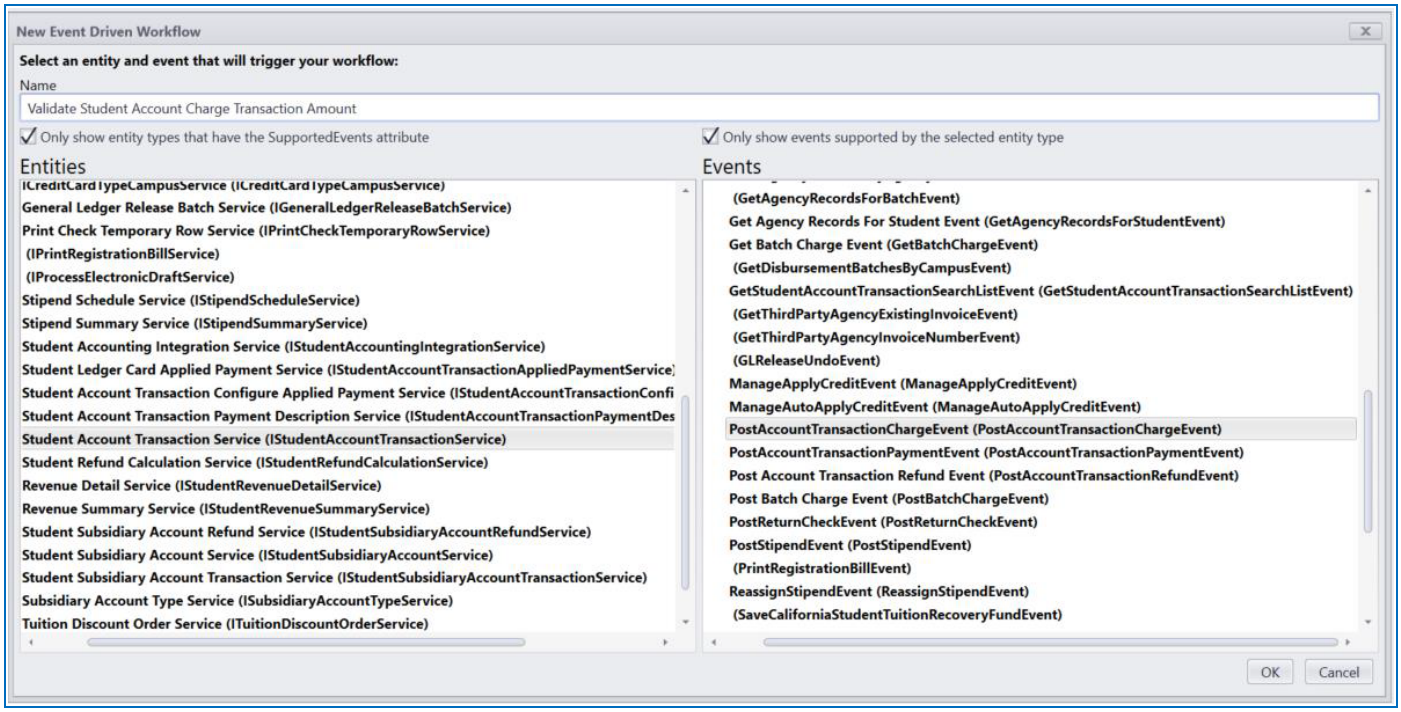


Workflow Validation of Business Process in Web Client

The change from previous Anthology Student versions (prior to 20.0) is that we never allowed a workflow to cancel further business process execution -— or to specify at which phase of the process the workflow will execute.

To test it out: Here's a simple example – School's Business Rule: Limit student account transactions to 10,000 or less – (don't allow to post if amount is greater than 10,000)

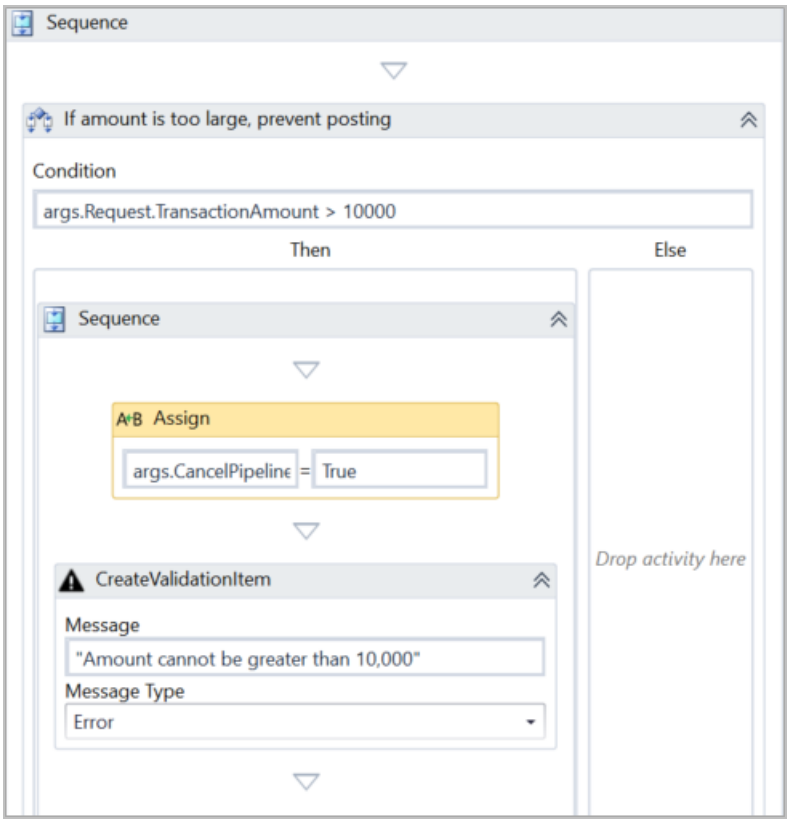
- Add workflow: Validate Student Account Charge Transaction Amount
- Service: Student Account Transaction Service
- Event: PostAccountTransactionChargeEvent



Add a condition – check if the requested TransactionAmount exceeds 10,000:

If true, cancel execution by using the Assign activity, set:
`args.CancelPipelineExecution = true`

Then, add whatever validation text you want to return, e.g., "Amount cannot be greater than 10,000"



Update the WorkflowDefinition so that it runs during the Validation phase (need to do this with a SQL Query for now, but with 21.0, it will be in Workflow Composer):

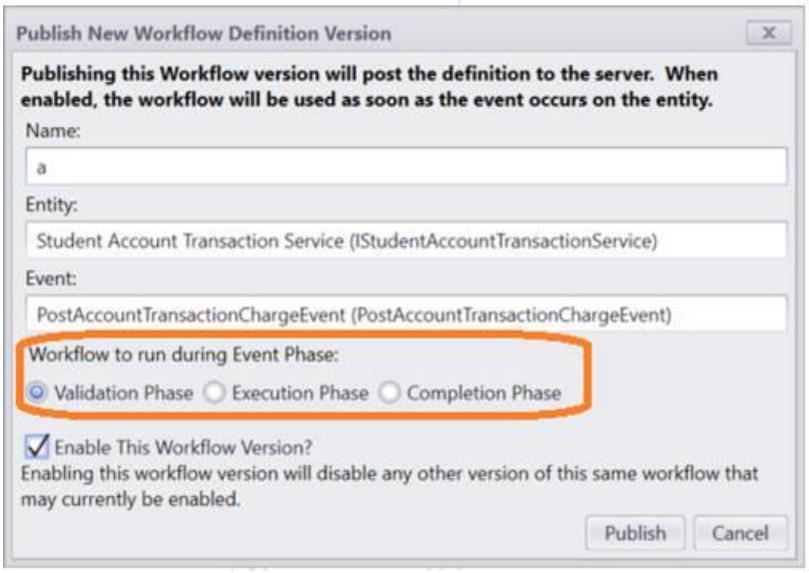
```
update dbo.WorkflowDefinition
set eventphase = 0
where name = 'Validate Student Account Charge Transaction Amount'
```

EventPhase 0 = Validation (before any posting occurs)

EventPhase 1 = Execution (during the posting to the database)

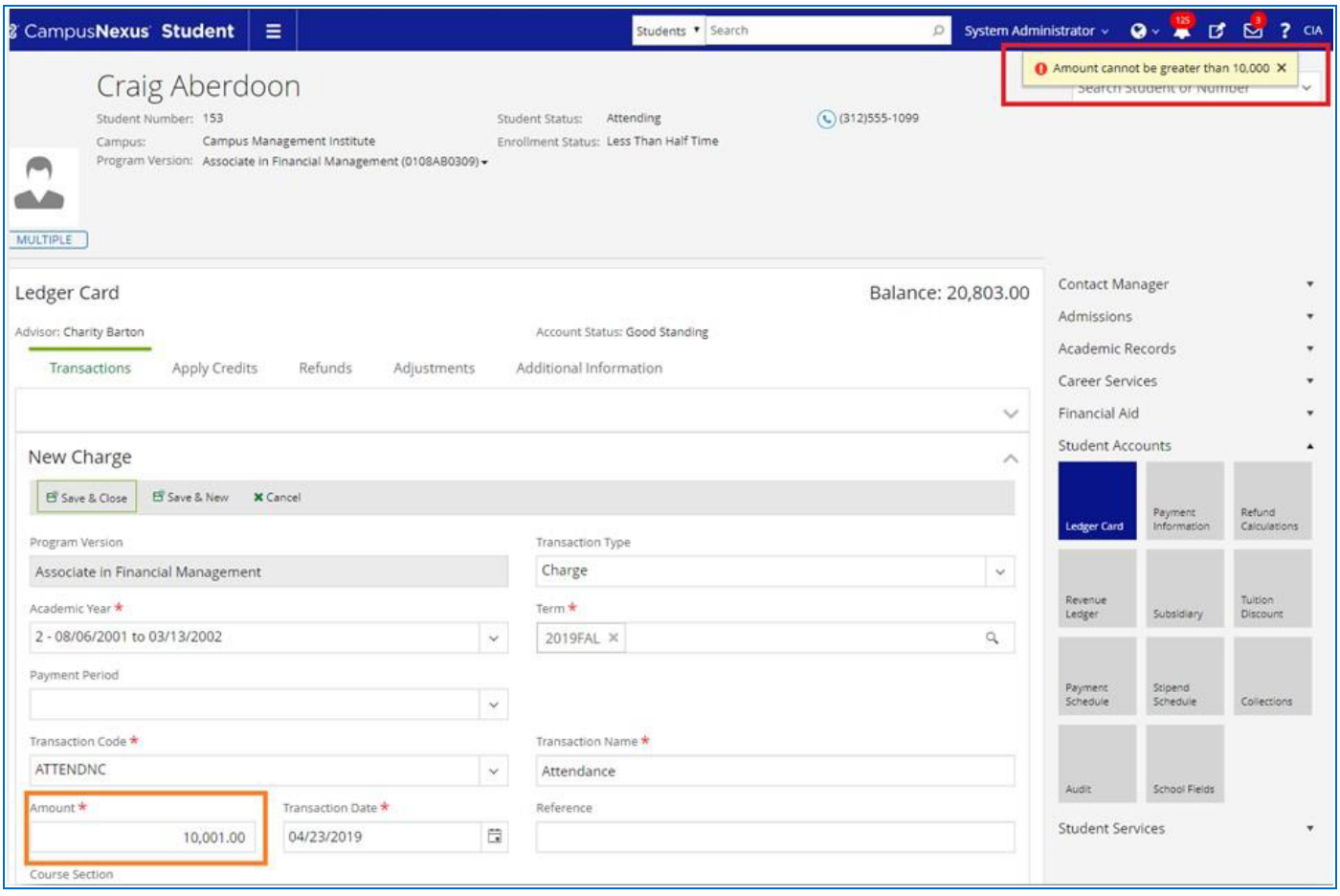
EventPhase 2 = Completion (after all has been committed)

Workflow Composer supports setting the EventPhase. It appears similar to the following, when publishing a workflow:



Although the framework will allow it in other phases; you should only set CancelPipelineExecution during Validation phase workflows. Cancelling later in the process will return unexpected results as you would have no idea what had been posted up until that point.

Test it out: Run the App and try to post a charge > 10,000



Verify that it was not saved:

```
select top 5 SaTransId, SyStudentId, SaBillCode, Amount, PostDate
from satrans
where systudentid = 26777
order by satransid desc
```

SaTransId	SyStudentId	SaBillCode	Amount	PostDate
32057	26777	BOOK	55.00	2018-08-13 11:39:47.487
31629	26777	TUIT	6000.00	2018-06-13 00:00:00.000
31628	26777	INS	1200.00	2018-06-13 00:00:00.000
31627	26777	BOOK	1200.00	2018-06-13 00:00:00.000
30963	26777	ARTSUP	25.00	2017-05-03 08:13:55.527

Then, save an amount of 10,000 or less and verify that it did post:

SaTransId	SyStudentId	SaBillCode	Amount	PostDate
32328	26777	ATTENDNC	10000.00	2019-04-23 12:48:51.377
32057	26777	BOOK	55.00	2018-08-13 11:39:47.487
31629	26777	TUIT	6000.00	2018-06-13 00:00:00.000
31628	26777	INS	1200.00	2018-06-13 00:00:00.000
31627	26777	BOOK	1200.00	2018-06-13 00:00:00.000

Resources

This section contains reference material that may assist you when designing and testing forms and sequences.

For additional resources:

- Refer to the Anthology Student Documentation Suite on the Product Information site.
- Visit the Community Portal to view product updates and log in to the Anthology Idea Exchange site to share, like, vote, and comment on your favorite ideas.
- Visit the Anthology Academy site (login required) for training classes and videos.

Analytics Tracker - Quick Reference

Forms Builder supports integration with Zuko (formerly Formisimo), a third-party analytics tool designed to track and analyze user interactions with forms. By specifying a Zuko slug in the Sequence Designer, you can monitor how users interact with your forms, including metrics on field engagement, abandonment, and completion rates.

Request a Partner Referral

To request form analytics integration, please contact Anthology Support at https://support.anthology.com/.

Provide the following information as part of a new partner request:

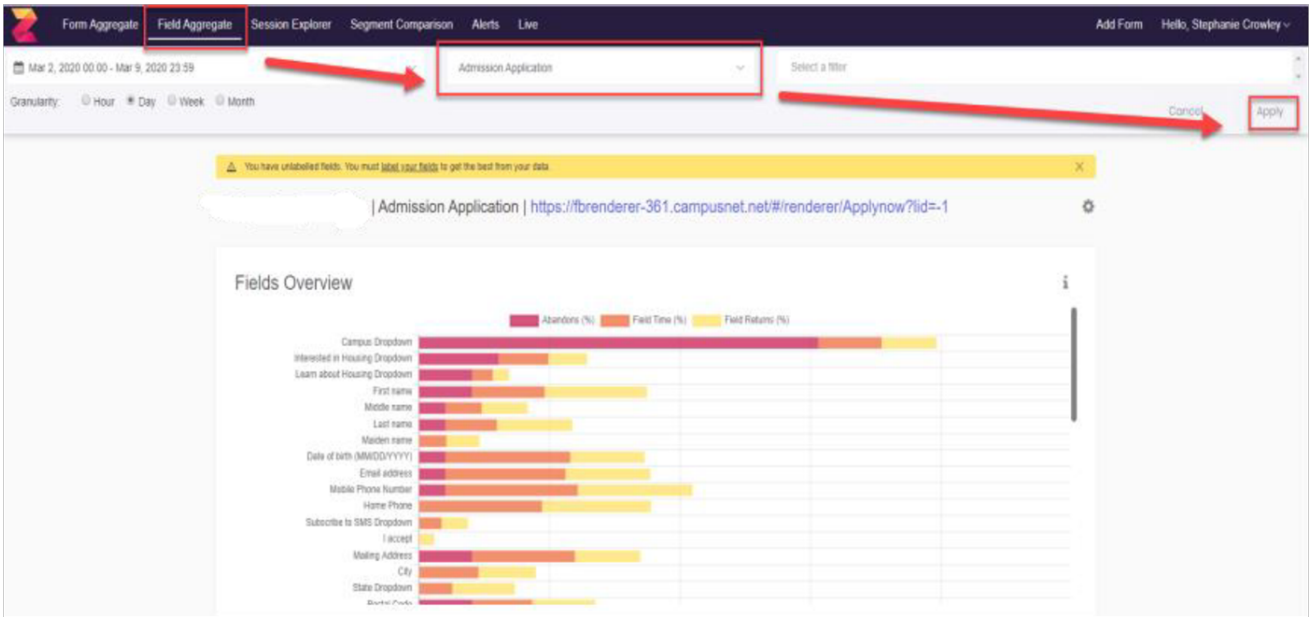
- Product: Forms Builder
- Partner: Formisimo/Zuko
- Additional Comments: Message of Request and Form Sequence Names/URLs to be tracked

Confirm an email acknowledgment for the request.

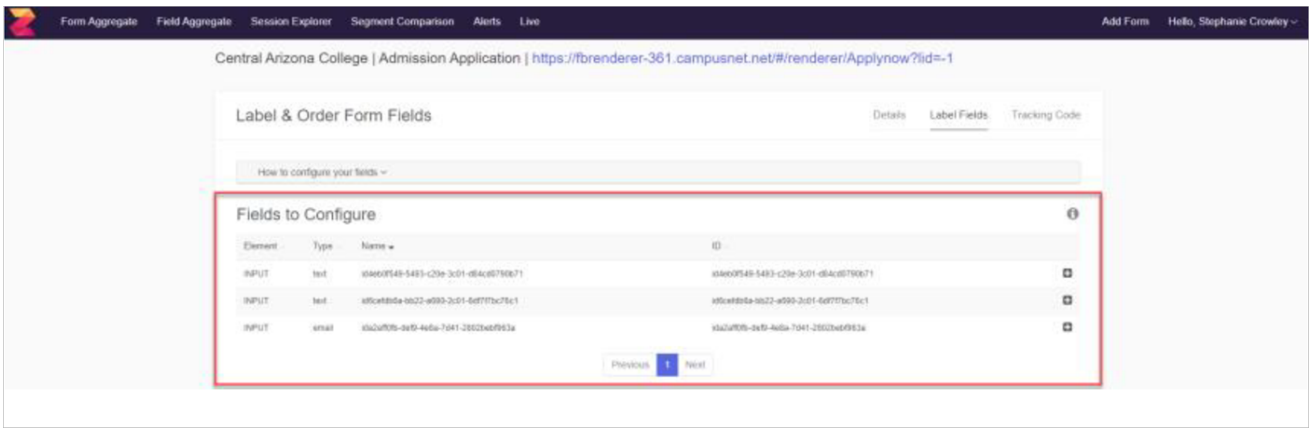
Map Fields

Perform the following steps to configure the field mapping between Forms Builder and Zuko.

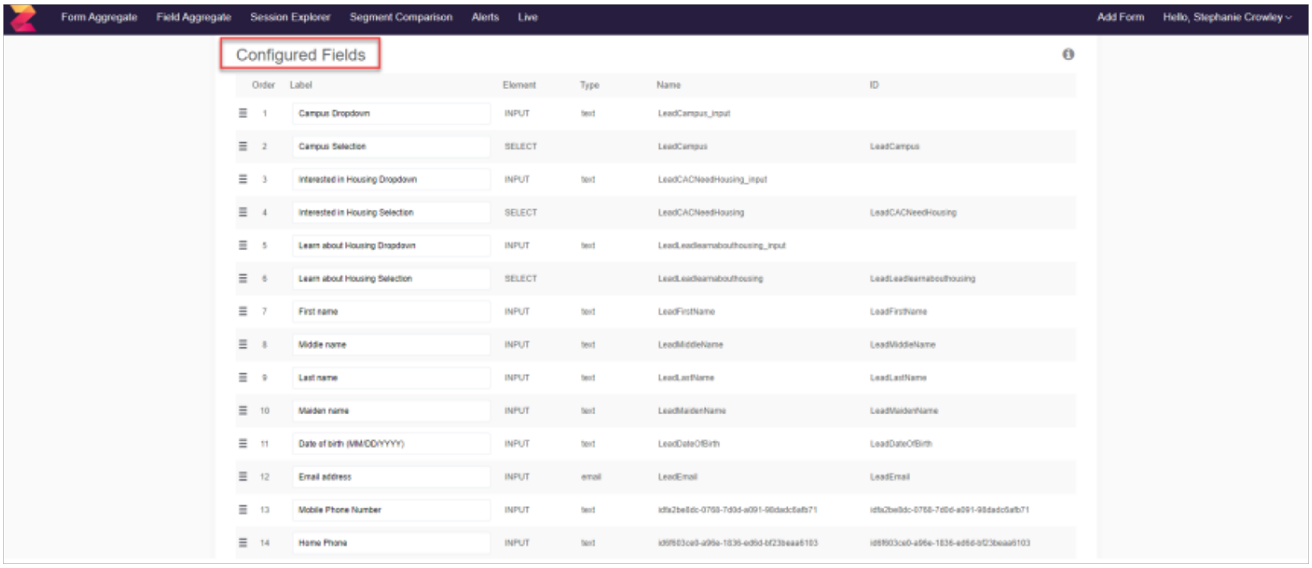
1. Log into the Zuko dashboard at www.zuko.io/login.
2. Select the **Field Aggregate** tab. Ensure the form that you want to map fields for is selected from the middle drop-down and select **Apply**.



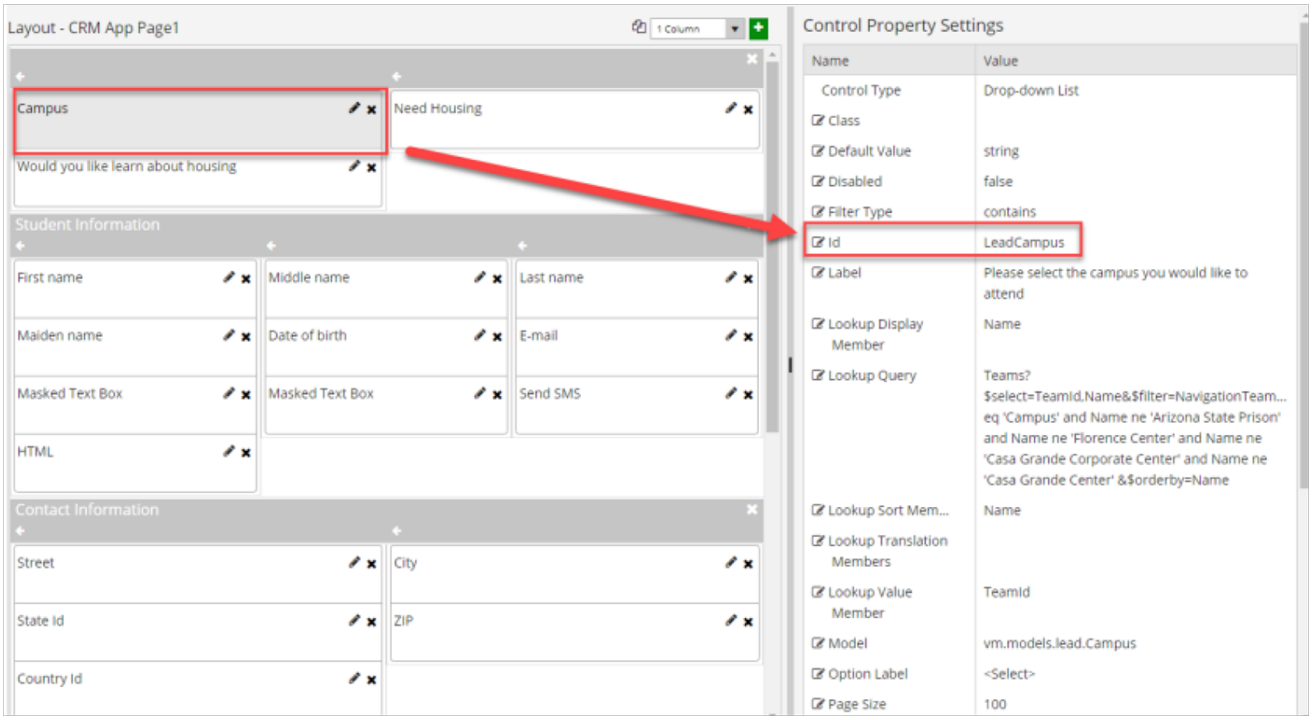
3. The initial setup will require you to map all fields for your forms that you want to track information against. The **Fields to Configure** section will list the fields available for mapping on your form. You will be able to see the **Element** (Field Type on the form), **Type** (Input Type on the form for that Element), **Name**, and **ID**.

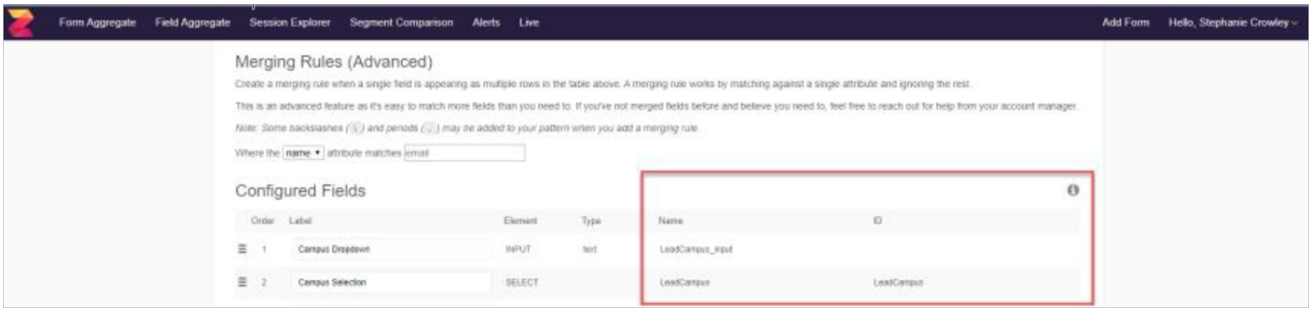


The fields that you have already been mapped will be listed under the Configured Fields section.

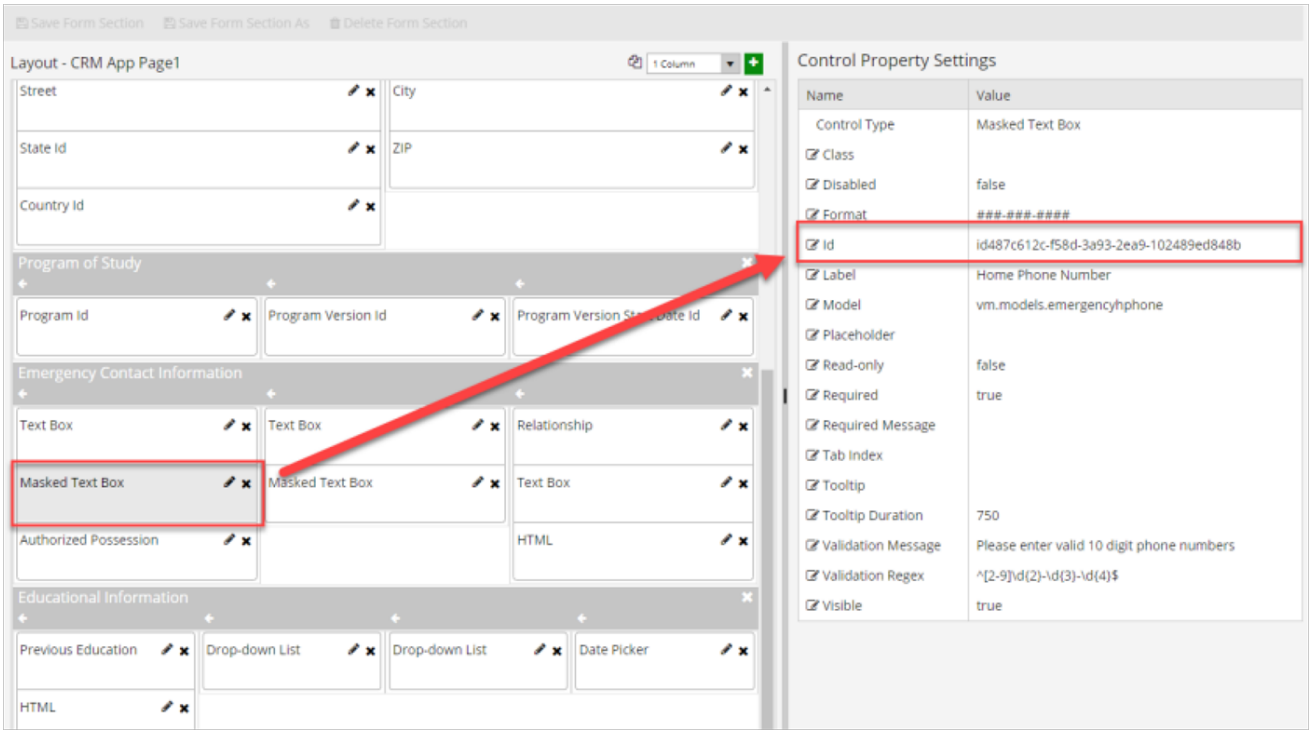


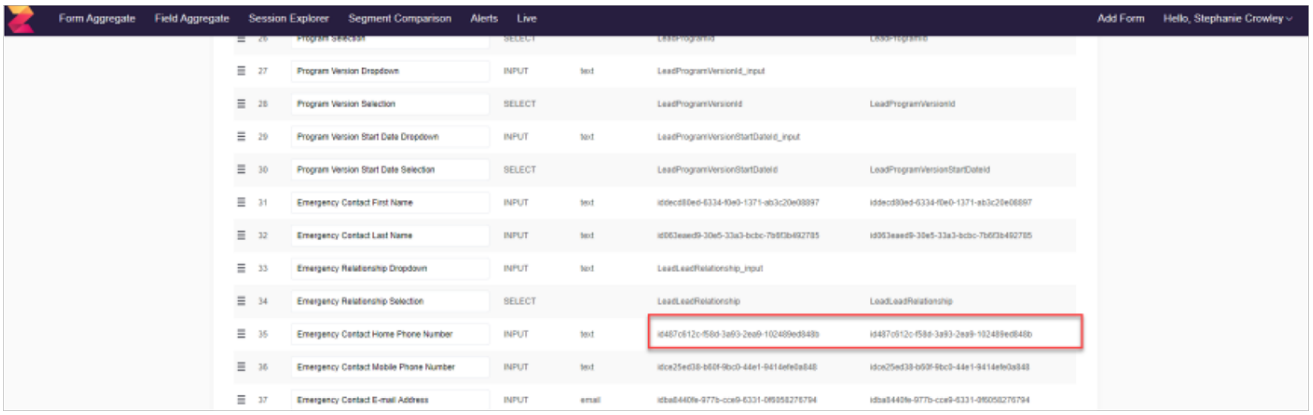
In **Forms Builder**, fields that are used on your form from an Entity should have an ID property that can help with identifying the field being used on the form when mapping in Zuko.



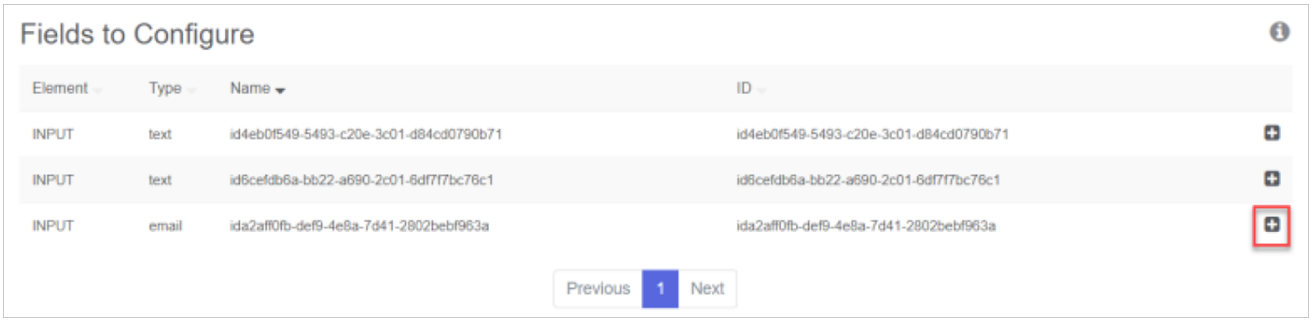


When using Custom Components on your form in Forms Builder, if the ID is not updated, it will pull the default ID as the Name/ID in Zuko which is harder to identify when performing the mapping. When mapping the field in Zuko, the easiest way to identify which field is being brought over to the Zuko Dashboard is to log in to Forms Builder Designer and review the properties for your Fields/Components on the form to match the ID values to what you see in Zuko.

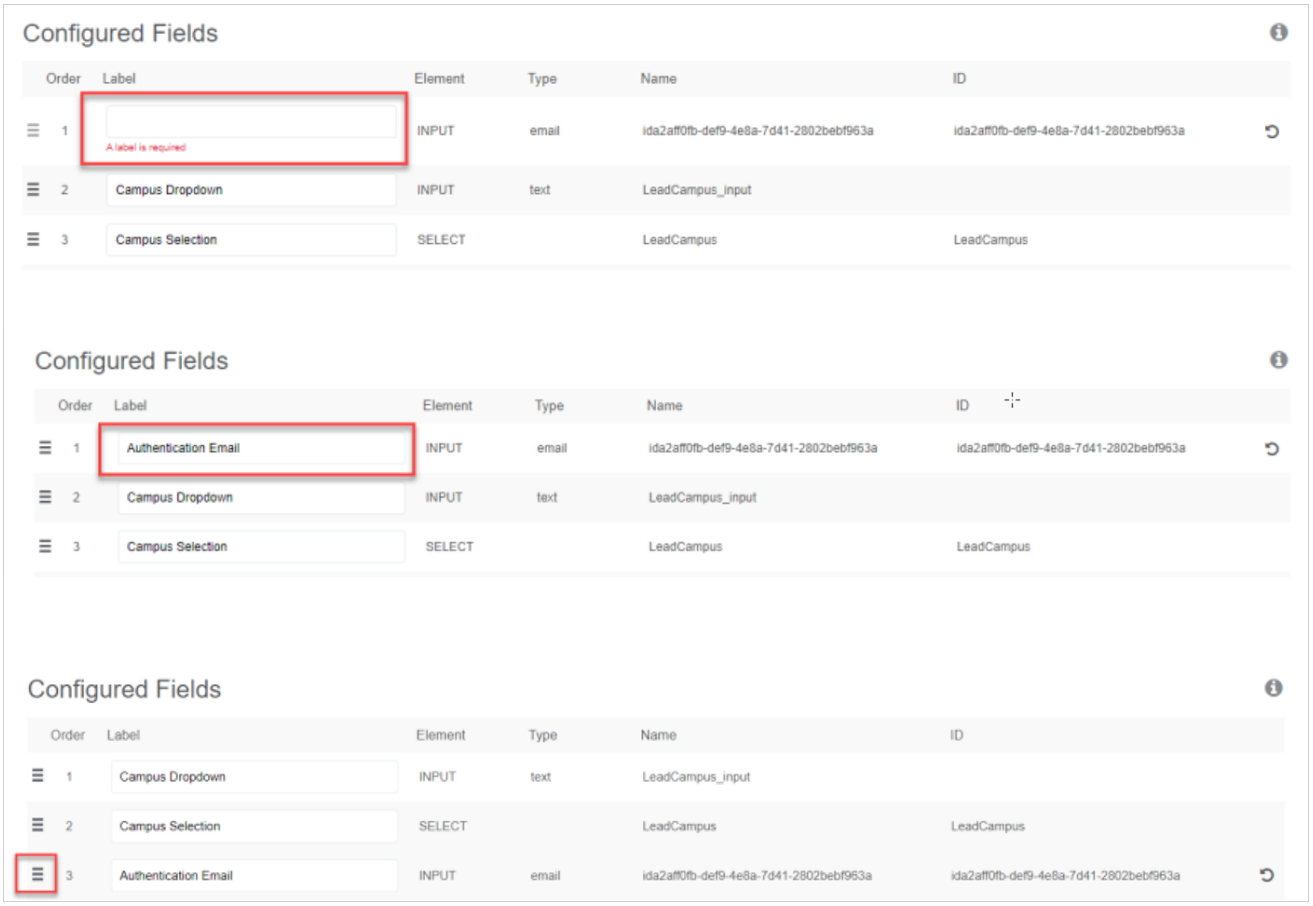




4. To configure a field in Zuko, select the **+** to the right of the field under the Fields to Configure section.



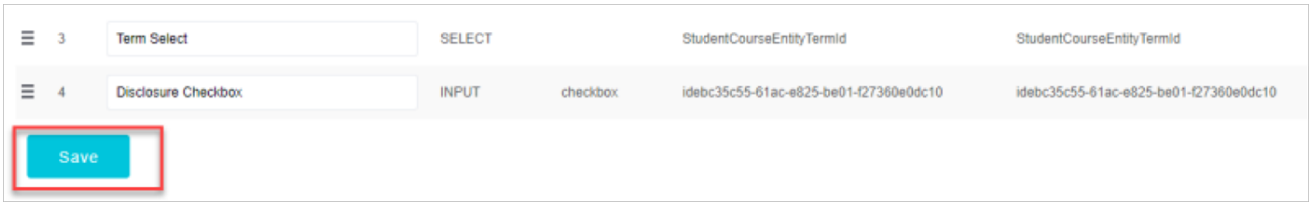
5. The field will be added to the **Configured Fields** section. In this section you will Label your field and order in the order that you want to view in the Dashboard.



Note: Best practice would be to order based on the order the fields are accessed when the Form Sequence is Rendered.

To reorder the fields, drag the and move the row accordingly to where you want to view it in the list of fields.

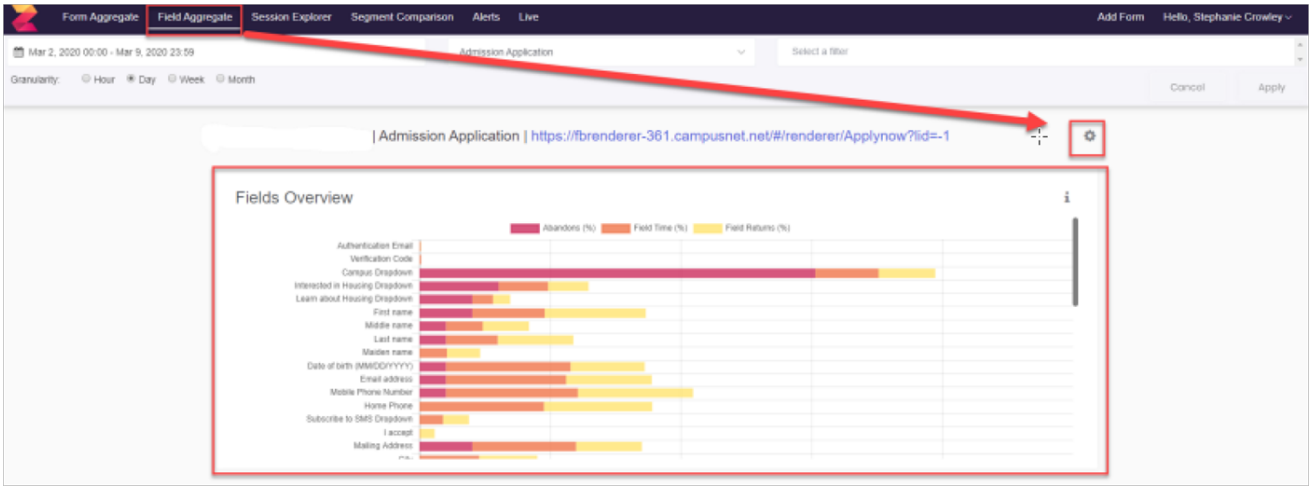
6. Once all your fields have been mapped, select **Save** at the bottom of the Configured Fields section.



Note: If you have used fields on your form that have been sent to the Zuko Dashboard and then were later removed from your form, just label those fields as **Ignore** if you do not want to monitor the data any longer.

For drop-down list fields/components, you will always have to map the Input and the **Select** values in Zuko.

7. If you access the **Field Aggregate** tab after you have mapped all fields, you will now find the fields in your Zuko Dashboards and will no longer get the message stating there are unmapped fields. To edit any of the field mapping labels, access the icon to the right of the URL.



CampusNexus CRM Entities

CampusNexus CRM entities are objects that can contain system-defined properties and properties that are manually configured by the customer. The custom properties are not addressed by the documentation.

Forms Builder supports the same list objects/entities as those that are described in the CampusNexus CRM Events topic in Workflow Help.

Limitations:

- For the Event object, only the Get operation is supported.
- For the Participant object, only the Get and Update operations are supported.
- For all other objects, the Get, Create, and Update operations are supported.
- The Delete operation is not supported in all objects.
- For external properties in all objects, only the Get activity is supported.

The first 1024 properties are published per entity for Forms Builder use. System-defined properties are published first and then the custom properties.

Refer to the Managing Objects section in Business Administrator Help to identify the system-defined properties of CampusNexus CRM objects.

The Show All Fields check box in Form Designer controls whether the entire list of CampusNexus CRM entities is exposed or only a subset. When the Show All check box is cleared, only the Contact and Lead entities are exposed.

Contact

The following table lists the properties of the system-defined fields in the Contact entity. Any additional fields displayed in Form Designer are customer-specific and are not included here.

Contact

Field	Required	Type	Description
First name	Yes	SystemString	The first name of the Contact you want to create in CampusNexus CRM.
Last name	Yes	SystemString	The last name of the Contact you want to create in CampusNexus CRM.
Middle name	No	SystemString	The middle name of the Contact you want to create in CampusNexus CRM.
Name	Yes	SystemString	The name of the Contact you want to create in CampusNexus CRM.

Lead

The following table lists the properties of the system-defined fields in the Lead entity. Any additional fields displayed in Form Designer are customer-specific and are not included here.

Lead

Field	Required	Type	Description
First name	Yes	SystemString	The first name of the Lead you want to create in CampusNexus CRM.
Last name	Yes	SystemString	The last name of the Lead you want to create in CampusNexus CRM.
Lead Name	Yes	SystemString	The name of the Lead you want to create in CampusNexus CRM.
Middle name	No	SystemString	The middle name of the Lead you want to create in CampusNexus CRM.
Team	Yes	SystemInt64	A unique ID used to identify a Team or Campus in CampusNexus CRM.

Anthology Student Entities

The following tables list the properties of fields in Anthology Student entities that are exposed in Form Designer when the "Select All" filter is cleared. If you need to reference properties of other entities and fields (when the "Select All" filter is selected), please refer to the Anthology Student Object Library.

Academic Year

When the this entity is selected, Form Designer exposes the following properties of the AcademicYearEntity class.

AcademicYearEntity

Field	Table : Column	Required	Type	Description	Example
Active	FaAcademicYear : Active	No	Boolean	Indicates whether the record is active.	0
Code	FaAcademicYear : Code	Yes	String	Indicates the code associated with the record.	AY1NPLN
Date Added	FaAcademicYear : DateAdded	No	DateTime	Date and time when the record was added.	2019-09-12 08:21:11.000
Date Modified	FaAcademicYear : DateLstMod	No	DateTime	Date and time when the record was last modified.	2019-12-01 09:22:35.423
Default Months in Academic Year	FaAcademicYear : DefaultMonthsinAY	No	Nullable<Int16>	Default number of months in an academic year.	11
Default Terms in Academic Year	FaAcademicYear : DefaultTermsInAY	No	Int16	Default number of terms in an academic year.	2
Grant Disb 1 Pct – Grant Disb 6 Pct	FaAcademicYear : Disb1PctGrant – Disb6PctGrant	No	Nullable<Decimal>	Indicates the percentage of a packaged grant per disbursement.	50.00
Id	FaAcademicYear : FaAcademicYearID	No	Int32	Identifier of the academic year entity record.	55
Loan Disb 1 Pct – Loan Disb 20 Pct	FaAcademicYear : Disb1PctLoan – Disb20PctLoan	No	Nullable<Decimal>	Indicates the percentage of a packaged loan per disbursement.	50.00
Modified By	FaAcademicYear : UserID	No	Int32	Identifier of the person who modified the record.	89
Name	FaAcademicYear : Descrip	No	String	Description of the record.	AY1

Field	Table : Column	Required	Type	Description	Example
Number Grant Disb	FaAcademicYear : NumDisbGrant	No	Nullable<Int16>	Sequence number of the grant disbursement.	2
Number Loan Disb	FaAcademicYear : NumDisbLoan	No	Nullable<Int16>	Sequence number of the loan disbursement.	2
Number Other Disb	FaAcademicYear : NumDisbOther	No	Nullable<Int16>	Sequence number of the other fund source disbursement.	2
Number Pell Disb	FaAcademicYear : NumDisbPell	No	Nullable<Int16>	Sequence number of the Pell disbursement.	2
Number Perkins Disb	FaAcademicYear : NumDisbPerkins	No	Nullable<Int16>	Sequence number of the Perkins disbursement.	2
Number Work Study Disb	FaAcademicYear : NumDisbWS	No	Nullable<Int16>	Sequence number of the work study disbursement.	1
Other Disb 1 Pct – Other Disb 6 Pct	FaAcademicYear : Disb1PctOther – Disb6PctOther	No	Nullable<Decimal>	Indicates the percentage of a packaged other fund source disbursement.	20.00
Packaging Method Id	FaAcademicYear : FaPackMethId	No	Nullable<Int32>	Identifier for the packaging method.	5
Pell Disb 1 Pct – Pell Disb 15 Pct	FaAcademicYear : Disb1PctPell – Disb15PctPell	No	Nullable<Decimal>	Indicates the percentage of a packaged Pell fund source disbursement.	50.00
Perkins Disb 1 Pct – Perkins Disb 1 Pct 6	FaAcademicYear : Disb1PctPerkins – Disb6PctPerkins	No	Nullable<Decimal>	Indicates the percentage of a packaged Perkins fund source per disbursement.	50.00
Row Version	FaAcademicYear : RowVersion	No	Byte[]	Row version associated with the AcademicYearEntity record.	0068
Weeks in Academic Year	FaAcademicYear : DefaultWeeksInAY	No	Nullable<Int16>	Default number of weeks in an academic year.	48

Field	Table : Column	Required	Type	Description	Example
Weeks Non-Enrolled	FaAcademicYear : DefaultWeeksNonEnroll	No	Nullable<Int16>	Default period of non-enrollment in an academic year.	0
Work Study Disb 1 Pct – Work Study Disb 6 Pct	FaAcademicYear : Disb1PctWS – Disb6PctWS	No	Nullable<Decimal>	Indicates the percentage of a packaged work study per disbursement.	50.00

Account Status

When the this entity is selected, Form Designer exposes the following properties of the AccountStatusEntity class.

AccountStatusEntity

Field	Table : Column	Required	Type	Description	Example
Code	SaAcctStatus : Code	Yes	String	Indicates the code associated with the record.	QWY
Date Added	SaAcctStatus : DateAdded	No	DateTime	Date and time when the record was added.	2019-09-12 08:21:11.000
Date Modified	SaAcctStatus : DateLstMod	No	DateTime	Date and time when the record was last modified.	2019-12-01 09:22:35.423
Name	SaAcctStatus : Descrip	No	String	Description of the record.	Deposit Received

Address Type

When the this entity is selected, Form Designer exposes the following properties of the AddressTypeEntity class.

AddressTypeEntity

Field	Table : Column	Required	Type	Description	Example
Active	syAddrType : Active	No	Boolean	Indicates whether the record is active.	1
Code	syAddrType : Code	Yes	String	Code for the address type.	PARENT
Date Added	syAddrType : DateAdded	No	DateTime	Date and time when the record was added.	2019-09-12 08:21:11.000
Date Modified	syAddrType : DateLstMod	No	DateTime	Date and time when the record was last modified.	2019-12-01 09:22:35.423
FERPA Address Type	syAddrType : showFERPA	No	Boolean	Indicates whether the record is a FERPA address type.	1
Is Excluded Crm Integration	syAddrType : CrmSuppressFlag	No	Boolean	Indicates whether the record is excluded from the CRM integration via Connector.	1
Modified By	syAddrType : UserID	No	Int32	Identifier of the person who modified the record.	89
Name	syAddrType : Descrip	No	String	Description of the record.	Parents
System	syAddrType : SystemCode	No	Boolean	Indicates whether the address type is a system code.	1
Use Effective Dates	syAddrType : UseEffectiveDates	No	Boolean	Indicates whether the address type is associated with an effective date.	1

Admissions Deposit

When the this entity is selected, Form Designer exposes the following properties of the DepositEntity class.

DepositEntity

Field	Table : Column	Required	Type	Description	Example
Academic Year Sequence	AmDeposit : AySeq	No	Nullable<Int16>	Academic year sequence number	1
Amount	AmDeposit : Amount	Yes	Decimal	Deposit amount	65.00
Check Number	AmDeposit : CheckNo	Conditional	String	Check number; MaxLength(20); required if payment type is H.	464635890900
Credit Card	AmDeposit : SaCCID	No	Nullable<Int32>	Student credit card identifier	3
Deposit Type	AmDeposit : DepositType	No	Nullable<Int16>	Valid values are : • 1 - Housing Deposit • 2 - Other Deposits	2
Enrollment	AmDeposit : AdEnrollID	No	Nullable<Int32>	Student enrollment period identifier	112309
Name	AmDeposit : Descrip	No	String	Description of the deposit; MaxLength (30)	Deposit Received
Paid By	AmDeposit : SyAddressID	No	Nullable<Int32>	Identifier of the person who posted the deposit (Student Relationship Address Id)	0
Payment Date	AmDeposit : DateReceived	Yes	DateTime	Deposit received date	10/31/2016
Payment Type	AmDeposit : PaymentType	No	String	Valid values are : • C - Cash • H - Check • R- Credit Card	H
Receipt Number	AmDeposit : ReceiptNo	No	String	Receipt number; MaxLength(16)	16-2113-006

Field	Table : Column	Required	Type	Description	Example
Term	AmDeposit : AdTermID	No	Nullable<Int32>	Term identifier	12891
Transaction Code	AmDeposit : SaBillCodeID	No	Nullable<Int32>	Billing transaction code identifier	0

Applicant Areas Of Study

When the this entity is selected, Form Designer exposes the following properties of the ApplicantAreaOfStudyEntity class.

ApplicantAreaOfStudyEntity

Field	Table : Column	Required	Type	Description	Example
Area Of Study Id	AdConcentrationByEnrollment : AdConcentrationID	Yes	Int32	Identifier for the concentration within an area of study	167

Applicants

When the this entity is selected, Form Designer exposes the following properties of the ApplicantEntity class.

ApplicantEntity

Field	Table : Column	Required	Type	Description	Example
Application Received Date	AdEnroll : AppRecDate	No	Nullable<DateTime>	Application received date	10/31/2016
Application Type	AdEnroll : AmApplicantTypeID	No	Nullable<Int32>	Application type identifier	2
Area Of Study Id	Refer to Applicant Areas Of Study.				
Campus	AdEnroll : SyCampusID	Yes	Int32	Campus identifier	4
Enrollment Date	AdEnroll : EnrollDate	No	Nullable<DateTime>	Enrollment date	10/31/2016
Expected Graduation Date	AdEnroll : GradDate	No	Nullable<DateTime>	Expected graduation date	10/31/2018
Expected Start Date	AdEnroll : ExpStartDate	No	Nullable<DateTime>	Expected start date	10/31/2016
Expected Start Term	AdEnroll : AdTermID	No	Nullable<Int32>	Expected start term identifier	12907
Externship Start Date	AdEnroll : ExternBeginDate	No	Nullable<Int32>	Externship start date	10/31/2017
Financial Aid Entrance Date	AdEnroll : FaEntrDate	No	Nullable<Int32>	Financial aid entrance interview date	10/31/2016
Pending Enrollment Number	AdEnroll : StuNum	No	String	Student enrollment number	100234
Previous Education	AdEnroll : AmPrevEducID	No	Nullable<Int32>	Previous education identifier	1
Program	AdEnroll : AdProgramID	No	Nullable<Int32>	Program identifier	2
Program Version	AdEnroll : AdProgramVersionID	No	Nullable<Int32>	Program version identifier	55
Shift	AdEnroll : AdShiftID	No	Nullable<Int32>	Shift identifier	1

Field	Table : Column	Required	Type	Description	Example
Version Start Date	AdEnroll : AdStartDateID	No	Nullable<Int32>	Version start date identifier	10/31/2016
Zone	AdEnroll : AmZoneId	No	Nullable<Int32>	Region/distance zone identifier	32

Area Of Study

When the this entity is selected, Form Designer exposes the following properties of the AreaOfStudyEntity class.

AreaOfStudyEntity

Field	Table : Column	Required	Type	Description	Example
Area Of Study Type Id	AdConcentration : AdConcentrationID	No	Int32	Identifier for the area of study type.	2
Campus Group Id	AdConcentration : SyCampusGrpID	No	Nullable<Int32>	Identifier for the campus group.	1311
CIP Code	AdConcentration : CipCode	No	String	CIP code for the area of study.	01.0106
Code	AdConcentration : Code	Yes	String	Code for the area of study.	CRIMLAW
Credit Hours Required	AdConcentration : CreditsReq	No	Nullable<Decimal>	Number of credits required for the area of study.	30.00
Date Added	AdConcentration : DateAdded	No	DateTime	Date and time when the record was added.	2019-09-12 08:21:11.000
Date Modified	AdConcentration : DateLstMod	No	DateTime	Date and time when the record was last modified.	2019-12-01 09:22:35.423
Id	AdConcentration : AdConcentrationID	Yes	Int32	Identifier for the area of study record.	32
IsActive	AdConcentration : Active	No	Boolean	Indicates whether the record is active.	1
Is Excluded Crm Integration	AdConcentration : CrmSuppressFlag	No	Boolean	Indicates whether the record is excluded from the CRM integration via Connector.	1
Minimum GPA Required	AdConcentration : MinGPA	No	Nullable<Decimal>	Minimum required GPA for the area of study.	12.00
Modified By	AdConcentration : UserID	No	Int32	Identifier of the person who modified the record.	89
Name	AdConcentration : Descrip	No	String	Description of the record.	Criminal Law

Field	Table : Column	Required	Type	Description	Example
Row Version	AdConcentration : RowVersion	No	Byte[]	Row version associated with the AreaOfStudyEntity record.	0068

Area Of Study Requirement Rule

When the this entity is selected, Form Designer exposes the following properties of the AreaOfStudyRequirementRuleEntity class.

AreaOfStudyRequirementRuleEntity

Field	Table : Column	Required	Type	Description	Example
Area Of Study Id	AdConcentrationRequirementRule : AdConcentrationID	No	Int32	Identifier for the area of study.	2
Created Date Time	AdConcentrationRequirementRule : DateAdded	No	DateTime	Date and time when the record was added.	2019-09-12 08:21:11.000
Last Modified Date Time	AdConcentrationRequirementRule : DateLstMod	No	DateTime	Date and time when the record was last modified.	2019-12-01 09:22:35.423
Last Modified User Id	AdConcentrationRequirementRule : UserID	No	Int32	Identifier of the person who modified the record.	89
Requirement Rule Id	AdConcentrationRequirementRule : AdRequirementRuleId	Yes	Int32	Identifier of the requirement rule.	32
Row Version	AdConcentrationRequirementRule : RowVersion	No	Byte[]	Row version associated with the AreaOfStudyRequirementRuleEntity record.	0068

Area Of Study Type

When the this entity is selected, Form Designer exposes the following properties of the AreaOfStudyTypeEntity class.

AreaOfStudyTypeEntity

Field	Table : Column	Required	Type	Description	Example
Active	AdConcentrationType : Active	No	Boolean	Indicates whether the record is active.	1
Code	AdConcentrationType : Code	Yes	String	Code for the area of study type.	MAJOR
Date Added	AdConcentrationType : DateAdded	No	DateTime	Date and time when the record was added.	2019-09-12 08:21:11.000
Date Modified	AdConcentrationType : DateLstMod	No	DateTime	Date and time when the record was last modified.	2019-12-01 09:22:35.423
Id	AdConcentrationType : AdConcentrationTypeID	Yes	Int32	Identifier for the area of study type record.	3
Is Excluded Crm Integration	AdConcentrationType : CrmSuppressFlag	No	Boolean	Indicates whether the record is excluded from the CRM integration via Connector.	1
Modified By	AdConcentrationType : UserID	No	Int32	Identifier of the person who modified the record.	89
Name	AdConcentrationType : Descrip	No	String	Description of the record.	Major
Row Version	AdConcentrationType : RowVersion	No	Byte[]	Row version associated with the AreaOfStudyTypeEntity record.	0068
System	AdConcentrationType : SystemCode	No	Boolean	Indicates whether the area of study type is a system code.	1

Document

When the this entity is selected, Form Designer exposes the following properties of the DocumentEntity class.

DocumentEntity

Field	Table : Column	Required	Type	Description	Example
Award Year	CmDocument : AwardYear	No	String	Award year; MaxLength(7)	2016-01
Date Requested	CmDocument : DateReq	No	Nullable<DateTime>	Requested date	10/31/2016
Date Sent	CmDocument : DateSent	No	Nullable<DateTime>	Sent date	10/31/2016
Document	CmDocument : DocumentImage	No	Byte[]	The document image associated with this DocumentEntity	0x255042..
Document Status	CmDocument : CmDocStatusID	Yes	Int32	Document status identifier	3
Document Transcript Request	Refer to Document Transcript Request.				
Document Type	CmDocument : CmDocTypeID	Yes	Int32	Document type identifier	23
Due Date	CmDocument : DateDue	No	Nullable<DateTime>	Document due date	10/31/2016
Enrollment	CmDocument : AdEnrollID	No	Nullable<Int32>	Student enrollment period identifier	5
Expiration Date	CmDocument : DateExpires	No	Nullable<DateTime>	Document expiration date	10/31/2016
Image Type	CmDocument : ImageType	No	String	Document image type; MaxLength (3)]	2
Module	CmDocument : SyModuleID	Yes	Int32	Module identifier	2
Note	CmDocument : Comments	No	String	Notes about the document	Reinstated student
Permit	CmDocument : IsPermit	No	Boolean	Is this a permit document?	0

Field	Table : Column	Required	Type	Description	Example
Received Date	CmDocument : DateRecv	No	Nullable<DateTime>	Date the document was received	10/31/2016
Student Placement	CmDocument : PlStudentPlacementID	No	Nullable<Int32>	Student placement identifier	1
Transcript Request	CmDocument : CmDocumentTranscriptID	No	Nullable<Int32>	Document transcript request identifier	1

Document Transcript Request

When the this entity is selected, Form Designer exposes the following properties of the DocumentTranscriptRequestEntity class.

DocumentTranscriptRequestEntity

Field	Table : Column	Required	Type	Description	Example
Attendance Dates: From	CmDocumentTranscript : AttendBeginDate	No	Nullable<DateTime>	Attendance begin date	10/31/2014
Fee	CmDocumentTranscript : Fee	No	Nullable<Decimal>	Document transcript request fee	35.00
Institution	CmDocumentTranscript : AmCollegeID	No	Nullable<Int32>	College identifier	1
Institution	CmDocumentTranscript : AmHighSchoolID	No	Nullable<Int32>	High school identifier	9
Note	CmDocumentTranscript : Comment	No	String	Comment or note; MaxLength (4000)	Incomplete
Program of Study	CmDocumentTranscript : ProgramOfStudy	No	String	Program of study; MaxLength (255)	Specialist in Education - Distance Learning
Request Type	CmDocumentTranscript : RequestIndicator	Yes	Int16	Request type	1
To	CmDocumentTranscript : AttendEndDate	No	Nullable<DateTime>	Attendance end date	10/31/2016

ISIR Verification

When the this entity is selected, Form Designer exposes the following properties of the IsirVerificationEntity class.

IsirVerificationEntity

Field	Table : Column	Required	Type	Description	Example
Field Number	FaISIRVerification : FieldNumber	No	String	Field number; MaxLength (3)	053
New Value	FaISIRVerification : VerificationValue	No	String	Verification value; MaxLength(50)	6016

Pending Applicant

When the this entity is selected, Form Designer exposes the following properties of the PendingApplicantEntity class.

PendingApplicantEntity

Field	Table : Column	Required	Type	Description	Example
Alien Number	AmOnlineApplicant : AlienNo	No	String	Alien number; MaxLength(9)	123456789
Applicant Type Id	AmOnlineApplicant : AmApplicantTypeID	No	Nullable<Int32>	Applicant type identifier	25
Areas of Study	Refer to Pending Applicant Area of Study.				
Campus Id	AmOnlineApplicant : CampusID	No	Nullable<Int32>	Campus identifier	10
Citizen Id	AmOnlineApplicant : CitizenID	No	Nullable<Int32>	Citizen identifier	1
City	AmOnlineApplicant : City	No	String	City; MaxLength(25)	Chicago
Country Id	AmOnlineApplicant : SyCountryID	No	Int32	Country identifier	1
Credit Card Expiration Date	AmOnlineApplicant : CardExpiration	No	<Nullable>DateTime	Card expiration date	10/31/2020
Credit Card Holder City	AmOnlineApplicant : CardHolderCity	No	String	Card holder's city; MaxLength(30)	Chicago
Credit Card Holder First Name	AmOnlineApplicant : CardHolderFirstname	No	String	Card holder's first name; MaxLength (50)	Mindy
Credit Card Holder Last Name	AmOnlineApplicant : CardHolderLastname	No	String	Card holder's last name; MaxLength (50)	Mayer
Credit Card Holder Name	AmOnlineApplicant : CardHolderName	No	String	Card holder's name; MaxLength(50)	

Field	Table : Column	Required	Type	Description	Example
Credit Card Holder Postal Code	AmOnlineApplicant : CardHolderZip	No	String	Card holder's postal code; MaxLength (15)	33099
Credit Card Holder State	AmOnlineApplicant : CardHolderState	No	String	Card holder's state; MaxLength(2)	HI
Credit Card Holder Street Address	AmOnlineApplicant : CardHolderAddr	No	String	Card holder's street address; MaxLength (50)	21 Main Street
Credit Card Number	AmOnlineApplicant : CardNumber	No	String	Card number; MaxLength(500)	1212-3234-2131-3232
Credit Card Type Id	AmOnlineApplicant : CardTypeId	No	Nullable<Int32>	Card type identifier	1
Date Of Birth	AmOnlineApplicant : DOB	No	Nullable<DateTime>	Date of birth	10/31/1995
Date Added	AmOnlineApplicant : DateAdded	No	DateTime	Date and time when the record was added.	2019-09-12 08:21:11.000
Date Modified	AmOnlineApplicant : DateLstMod	No	DateTime	Date and time when the record was last modified.	2019-12-01 09:22:35.423
Degree Id	AmOnlineApplicant : DegreeID	No	Nullable<Int32>	Degree identifier	2
Driver License Number	AmOnlineApplicant : DrivLic	No	String	Driver's license number; MaxLength(20)	R360-555-87-752-0
Driver License State	AmOnlineApplicant : DrivLicState	No	String	Driver's license state; MaxLength(2)	IN
Email Address	AmOnlineApplicant : Email	No	String	Email address; MaxLength(50)	tester@anthology.com
Ethnicities	Refer to Pending Applicant Ethnicity.				

Field	Table : Column	Required	Type	Description	Example
Final School Status Id	AmOnlineApplicant : FinalStatusID	No	Nullable<Int32>	Final school status identifier	62
First Name	AmOnlineApplicant : FirstName	No	String	First name; MaxLength(25)	Mario
Ged Awarded Date	AmOnlineApplicant : GEDAwardedDate	No	Nullable<DateTime>	GED awarded date	10/31/2002
Gender Id	AmOnlineApplicant : SexID	No	Nullable<Int32>	Gender identifier	6
Hispanic Latino	AmOnlineApplicant : IsHispanic	No	String	Is Hispanic/Latino; MaxLength(1)	N
Id	AmOnlineApplicant : AmOnlineApplicantID	Yes	Int32	Identifier for the PendingApplicantEntity record.	560
Is Subscribed To Sms	AmOnlineApplicant : SubscribeToSMS	No	Boolean	Indicates whether the applicant subscribed to SMS	true
Last Name	AmOnlineApplicant : LastName	No	String	Last name; MaxLength(25)	Romanov
Lead Source Id	AmOnlineApplicant : AmLeadSrcID	No	Int32	Lead source identifier	699
Lead Type Id	AmOnlineApplicant : AmLeadTypeID	No	Nullable<Int32>	Lead type identifier	4
Marital Status Id	AmOnlineApplicant : MaritalID	No	Nullable<Int32>	Marital status identifier	1
Middle Name	AmOnlineApplicant : MiddleName	No	String	Middle name; MaxLength(100)	Thomas
Mobile Phone Number	AmOnlineApplicant : MobileNumber	No	String	Mobile phone number; MaxLength(16)	954-333-1212
Other Email Address	AmOnlineApplicant : OtherEmail	No	String	Other email address; MaxLength(50)	mario@anthology.com
Other Phone Number	AmOnlineApplicant : OtherPhone	No	String	Other phone number; MaxLength(16)	305-333-1212
Payment Amount	AmOnlineApplicant : PaymentAmount	No	Nullable<Decimal>	Payment amount	235.00

Field	Table : Column	Required	Type	Description	Example
Permanent City	AmOnlineApplicant : PermCity	No	String	Permanent city; MaxLength(25)	Jolie
Permanent Country Id	AmOnlineApplicant : PermSyCountryID	No	Nullable<Int32>	Permanent country identifier	0
Permanent Phone Number	AmOnlineApplicant : PermPhone	No	String	Permanent phone number; MaxLength (16)	954-333-1212
Permanent Postal Code	AmOnlineApplicant : PermZip	No	String	Permanent postal code; MaxLength (10)	33071
Permanent State	AmOnlineApplicant : PermState	No	String	Permanent state; MaxLength(2)	FL
Permanent Street Address	AmOnlineApplicant : PermAddress	No	String	Permanent street address; MaxLength (255)	200 Broadway
Phone Number	AmOnlineApplicant : Phone	No	String	Phone number; MaxLength(16)	954-333-1212
Postal Code	AmOnlineApplicant : Zip	No	String	Postal code; MaxLength(10)	30071
Previous Education History	Refer to Pending Applicant Previous Education.				
Previous Education Id	AmOnlineApplicant : AmPrevEducID	No	Nullable<Int32>	Previous education identifier	3
Program Id	AmOnlineApplicant : ProgramID	No	Nullable<Int32>	Program identifier	2
Program Version Id	AmOnlineApplicant : ProgramVersionID	No	Nullable<Int32>	Program version identifier	1
Sms Service Provider Id	AmOnlineApplicant : CMSMSServiceProviderID	No	Nullable<Int32>	SMS service provider identifier	2
Ssn	AmOnlineApplicant : SSN	No	String	Social Security Number; MaxLength(16)	444-77-9999

Field	Table : Column	Required	Type	Description	Example
Start Date	AmOnlineApplicant : StartDate	No	Nullable<DateTime>	Start date	10/31/2016
Start Date Id	AmOnlineApplicant : StartDateID	No	Nullable<Int32>	Start date identifier	1
State	AmOnlineApplicant : State	No	String	State; MaxLength(2)	GA
Street Address	AmOnlineApplicant : Address	No	String	Street address; MaxLength(255)	6585 Turkey Trail
Title Id	AmOnlineApplicant : AmTitleID	No	Nullable<Int32>	Title identifier	1
Work Phone Number	AmOnlineApplicant : WorkPhone	No	String	Work phone number; MaxLength(16)	678-543-3330
Work Phone Number Extension	AmOnlineApplicant : WorkExt	No	String	Work phone number extension; MaxLength(10)	2345

Pending Applicant Area of Study

When the this entity is selected, Form Designer exposes the following properties of the PendingApplicantAreaOfStudyEntity class.

PendingApplicantAreaOfStudyEntity

Field	Table : Column	Required	Type	Description	Example
Area Of Study Id	AmOnlineApplicantConcentration : AdConcentrationId	Yes	Int32	Identifier for the concentration within an area of study	2
Is Active	AmOnlineApplicantConcentration : Active	No	Nullable Boolean	Indicates whether the area of study is active	true

Pending Applicant Ethnicity

When the this entity is selected, Form Designer exposes the following properties of the PendingApplicantEthnicityEntity class.

PendingApplicantEthnicityEntity

Field	Table : Column	Required	Type	Description	Example
Ethnicity Id	AmOnlineApplicantAmRace : AmRaceID	No	Int32	Ethnicity identifier	8

Pending Applicant Previous Education

When the this entity is selected, Form Designer exposes the following properties of the PendingApplicantPreviousEducationEntity class.

PendingApplicantPreviousEducationEntity

Field	Table : Column	Required	Type	Description	Example
College Id	AmOnlineApplicantPrevEducation : AmCollegeID	No	Nullable Int32	College identifier	6
Degree Id	AmOnlineApplicantPrevEducation : DegreeID	No	Nullable Int32	Degree identifier	4
Enrollment Date	AmOnlineApplicantPrevEducation : DateOfEnrollment	No	Nullable DateTime	Enrollment date	10/31/2016
Gpa	AmOnlineApplicantPrevEducation : GPA	No	Decimal	Grade point average	3.2
Graduation Date	AmOnlineApplicantPrevEducation : GradDate	No	Nullable DateTime	Graduation date	10/31/2018
Last Attended Date	AmOnlineApplicantPrevEducation : LastDateAttended	No	Nullable DateTime	Last attended date	10/31/2014
Major	AmOnlineApplicantPrevEducation : Major	No	String	Major; MaxLength (100)	Biology
Other College	AmOnlineApplicantPrevEducation : OtherCollege	No	String	Other college; MaxLength (60)	Campus Institute

Pending Prospect Inquiry

When the this entity is selected, Form Designer exposes the following properties of the PendingProspectInquiryEntity class.

PendingProspectInquiryEntity

Field	Table : Column	Required	Type	Description	Example
Best Time to Contact	AmElectronicLeads : BestTimeToContact	No	String	Best time to contact; MaxLength (10)	Evening
Birth Date	AmElectronicLeads : DOB	No	Nullable DateTime	Birth date	10/31/1995
Campus	AmElectronicLeads : SyCampusID	No	Nullable Int32	Campus identifier	1
Citizenship	AmElectronicLeads : AmCitizenID	No	Nullable Int32	Citizenship identifier	2
City	AmElectronicLeads : City	No	String	City; MaxLength (25)	Chicago
College	AmElectronicLeads : AmCollegeID	No	Nullable Int32	College identifier	5
Country	AmElectronicLeads : SyCountryID	No	Nullable Int32	Country identifier	3
Disabled	AmElectronicLeads : Disabled	No	String	Disabled; MaxLength(1)	N
Email Address	AmElectronicLeads : Email	No	String	Email address; MaxLength(50)	tester@anthology.com
Ethnicities	Refer to Pending Prospect Inquiry Ethnicity.				
Expected Start Date	AmElectronicLeads : StartDate	No	Nullable DateTime	Expected start date	10/31/2016
Extra Curricular Activity	AmElectronicLeads : AmExtraCurrID	No	Nullable Int32	Extra curricular activity identifier	8
First Name	AmElectronicLeads : FirstName	No	String	First name; MaxLength(25)	Mario
Gender	AmElectronicLeads : AmSexID	No	Nullable Int32	Gender identifier	2
High School	AmElectronicLeads : AmHighSchoolID	No	Nullable Int32	High school identifier	554

Field	Table : Column	Required	Type	Description	Example
High School Grad Date	AmElectronicLeads : HSGradDate	No	Nullable DateTime	High school graduation date	10/31/2014
High School Grad Year	AmElectronicLeads : HSGradYear	No	String	High school graduation year; MaxLength(4)	2014
Hispanic	AmElectronicLeads : IsHispanic	No	String	Is Hispanic; MaxLength(1)	N
Last Name	AmElectronicLeads : LastName	No	String	Last name; MaxLength(25)	Romanov
Lead Source	AmElectronicLeads : AmLeadSrcID	No	Nullable Int32	Lead source identifier	44
Lead Type	AmElectronicLeads : AmLeadTypeID	No	Nullable Int32	Lead type identifier	3
Marital Status	AmElectronicLeads : AmMaritalID	No	Nullable Int32	Marital status identifier	1
Middle Initial	AmElectronicLeads : MI	No	String	Middle initial; MaxLength(1)	M
Middle Name	AmElectronicLeads : MiddleName	No	String	Middle name; MaxLength(100)	Merlin
Mobile Phone Number	AmElectronicLeads : MobileNumber	No	String	Mobile phone number; MaxLength(16)	954-222-9090
Nationality	AmElectronicLeads : AmNationalityID	No	Nullable Int32	Nationality identifier	8
Note	AmElectronicLeads : Comment	No	String	Note; MaxLength (2000)	Needs financial aid
Other Email Address	AmElectronicLeads : OtherEmail	No	String	Other email address; MaxLength(50)	mario@anthology.com
Other Phone Number	AmElectronicLeads : OtherPhone	No	String	Other phone number; MaxLength(16)	954-202-8765
Phone Number	AmElectronicLeads : Phone	No	String	Phone number; MaxLength(16)	954-202-8888
Postal Code	AmElectronicLeads : Zip	No	String	Postal code; MaxLength(10)	33088
Previous Education	AmElectronicLeads : AmPrevEducID	No	Nullable Int32	Previous education identifier	3

Field	Table : Column	Required	Type	Description	Example
Program	AmElectronicLeads : AdProgramID	No	Nullable Int32	Program identifier	2
Program Group	AmElectronicLeads : AdProgramGroupID	No	Nullable Int32	Program group identifier	2
Shift	AmElectronicLeads : AdShiftID	No	Nullable Int32	Shift identifier	1
Start Term	AmElectronicLeads : AdTermID	No	Nullable Int32	Start term identifier	2
State	AmElectronicLeads : State	No	String	State; MaxLength(2)	TX
Street Address	AmElectronicLeads : Addr	No	String	Street address; MaxLength(255)	98 Track Trail
Suffix	AmElectronicLeads : AmSuffixID	No	Nullable Int32	Suffix identifier	1
Title	AmElectronicLeads : AmTitleID	No	Nullable Int32	Title identifier	2
Veteran	AmElectronicLeads : Vet	No	String	Veteran; MaxLength(1)	N
Work Phone Number	AmElectronicLeads : WorkPhone	No	String	Work phone number; MaxLength(16)	954-666-1212

Pending Prospect Inquiry Ethnicity

When the this entity is selected, Form Designer exposes the following properties of the PendingProspectInquiryEthnicityEntity class.

PendingProspectInquiryEthnicityEntity

Field	Table : Column	Required	Type	Description	Example
Ethnicity Id	AmElectronicLeadsAmRace: AmRaceID	No	Int32	Ethnicity identifier	8

Refer to Multiselect for Single Property Collections.

Prospect Inquiry

When the this entity is selected, Form Designer exposes the following properties of the ProspectInquiryEntity class.

ProspectInquiryEntity

Field	Table : Column	Required	Type	Description	Example
Campus	SyStudentInquiry : SyCampusID	Yes	Int32	Campus identifier	32
Interest	SyStudentInquiry : AdProgramGroupID	No	Nullable Int32	Program group identifier	6
Primary Prospect Source	SyStudentInquiry : AmLeadSrcID	No	Nullable Int32	Lead source identifier	7
Program	SyStudentInquiry : AdProgramID	No	Nullable Int32	Program identifier	12
Programs	Refer to Prospect Inquiry Program of Interest.				
Prospect Date	SyStudentInquiry : LeadDate	Yes	DateTime	Lead date	10/31/2016
Prospect Sources	Refer to Prospect Inquiry Lead Source.				
Prospect Type	SyStudentInquiry : AmLeadTypeID	No	Nullable Int32	Lead type identifier	43
Region	SyStudentInquiry : AmZoneID	No	Nullable Int32	Region identifier	65
Student	Refer to Student.				

Prospect Inquiry Lead Source

When the this entity is selected, Form Designer exposes the following properties of the ProspectInquiryLeadSourceEntity class.

ProspectInquiryLeadSourceEntity

Field	Table : Column	Required	Type	Description	Example
Lead Date	amProspectLeadSrc : SourceDate	No	Nullable DateTime	Lead date	10/31/2016
Lead Source	amProspectLeadSrc : AmLeadSrcID	Yes	Int32	Lead source identifier	1
Primary Lead Source	amProspectLeadSrc : PrimarySource	No	Boolean	Is this a primary lead source?	true

Prospect Inquiry Program of Interest

When the this entity is selected, Form Designer exposes the following properties of the ProspectInquiryProgramEntity class.

ProspectInquiryProgramEntity

Field	Table : Column	Required	Type	Description	Example
Is Primary	AmProspectProgram : PrimaryProgram	No	Boolean	Is this a primary program of interest?	true
Program	AmProspectProgram : AdProgramID	Yes	Int32	Program identifier	1

Refer to Multiselect for Single Property Collections.

Student

When the this entity is selected, Form Designer exposes the following properties of the StudentEntity class.

StudentEntity

Field	Table : Column	Required	Type	Description	Example
Agency Sponsor	syStudent : AmAgencyID	No	Nullable<Int32>	Agency identifier	9
Alien Number	syStudent : AlienNo	No	String	Alien number; MaxLength(9)	123456789
Best Time to Contact	syStudent : BestTimeToContact	No	String	Athletic identifier; MaxLength(10)	Morning
Campus	syStudent : SyCampusID	Yes	Int32	Campus identifier	10
Citizenship	syStudent : AmCitizenID	No	Nullable<Int32>	Citizenship status identifier	1
City	syStudent : City	No	String	City; MaxLength(25)	Chicago
Country	syStudent : SyCountryID	No	Nullable<Int32>	Country identifier	1
County	syStudent : SyCountyID	No	Nullable<Int32>	County identifier	87
Current Employer	syStudent : PlEmployerID	No	Nullable<Int32>	Current employer identifier	62
Date of Birth	syStudent : DOB	No	Nullable DateTime	Date of Birth	1995-08-21 00:00:00.000
Disabled	syStudent : Disabled	No	String	Disabled? MaxLength(1)	N
Driver License Number	syStudent : DrivLic	No	String	Driver's license number; MaxLength(20)	R360-555-87-752-0
Driver License State	syStudent : DrivLicState	No	String	Driver's license state; MaxLength(2)	IN
Email Address	syStudent : email	No	String	Email address; MaxLength(50)	tester@anthology.com

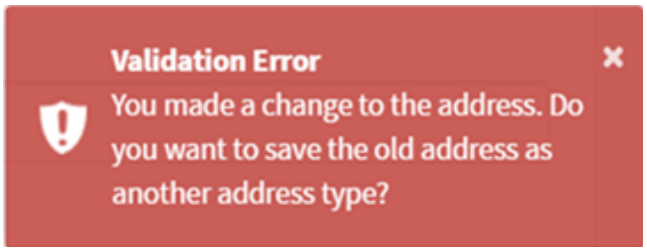
Field	Table : Column	Required	Type	Description	Example
Employment Status	syStudent : plEmpStatusID	No	Nullable<Int32>	Employment status identifier	3
Expected Start	syStudent : StartDate	No	Nullable<DateTime>	Expected start date	2019-08-21 00:00:00.000
First Name	syStudent : FirstName	No	String	First name; MaxLength(25)	Mario
Gender	syStudent : AmSexID	No	Nullable<Int32>	Genderidentifier	1
Hispanic/Latino	syStudent : IsHispanic	No	String	Is the student Hispanic/Latino?; MaxLength(1)	N
Interest	syStudent : AdProgramGroupID	No	Nullable<Int32>	Program group of interest	0
Last Name	syStudent : LastName	No	String	Last name; MaxLength(25)	Romanov
Maiden Name	syStudent : MaidenName	No	String	Maiden name; MaxLength(19)	Jolie
Marital Status	syStudent : AmMaritalID	No	Nullable<Int32>	Marital status identifier	1
Middle Initial	syStudent : MI	No	String	Middle initial; MaxLength(1)	T
Middle Name	syStudent : MiddleName	No	String	Middle name; MaxLength (100)	Thomas
Mobile Phone Number	syStudent : MobileNumber	No	String	Mobile phone number; MaxLength(16)	954-333-1212
Nationality	syStudent : AmNationalityID	No	Nullable<Int32>	Nationality identifier	21
Nickname	syStudent : NickName	No	String	Nickname; MaxLength(14)	Tom
Other Email Address	syStudent : OtherEmail	No	String	Other email address; MaxLength(50)	mario@anthology.com
Other Phone Number	syStudent : OtherPhone	No	String	Other phone number; MaxLength(16)	305-333-1212
Phone Number	syStudent : Phone	No	String	Phone number; MaxLength(16)	954-333-1212

Field	Table : Column	Required	Type	Description	Example
Previous Education Level	syStudent : AmPrevEducID	No	Nullable<Int32>	Previous education level identifier	4
Program	syStudent : AdProgramID	No	Nullable<Int32>	Program identifier	8
Shift	syStudent : AdShiftID	No	Nullable<Int32>	Shift identifier	1
SMS Provider	syStudent : CMSMSServiceProviderID	No	Nullable<Int32>	SMS service provider identifier	2
SSN	syStudent : SSN	No	String	Social security number; MaxLength(30)	444-77-9999
State	syStudent : State	No	String	State; MaxLength(2)	FL
Street Address	syStudent : Addr1	No	String	Street address; MaxLength (255)	200 Broadway
Student Number	syStudent : StuNum	No	String	Student number; MaxLength (10)	9333212
Subscribe to SMS	syStudent : SubscribeToSMS	No	Boolean	Subscribe to SMS	true
Suffix	syStudent : AmSuffixID	No	Nullable<Int32>	Suffix identifier	1
Title	syStudent : AmTitleID	No	Nullable<Int32>	Title identifier	2
Veteran	syStudent : Vet	No	String	Veteran? MaxLength(1)	N
Work Phone Number	syStudent : WorkPhone	No	String	Work phone number; MaxLength(16)	678-543-3330
Work Phone Number Ext	syStudent : Workext	No	String	Work phone number extension; MaxLength(10)	2345
ZIP Code/Postal Code	syStudent : Zip	No	String	Postal code; MaxLength(10)	33071

Student Address Changes

The message *"You made a change to the address. Do you want to save the old address as another address type?"* is displayed when a student's address is modified including any of the following fields of the StudentEntity:

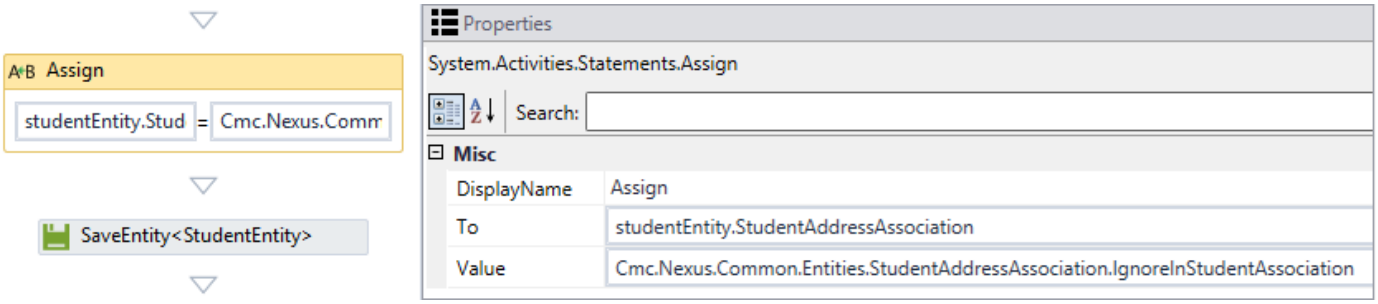
- FirstName
- LastName
- MiddleName
- StreetAddress
- City
- State
- PostalCode
- CountryId
- PhoneNumber
- WorkPhoneNumber
- EmailAddress



The message is not a validation message that can be handled in Forms Builder or in Workflow Composer during a SaveEntity activity for the StudentEntity. To avoid this message after updating a student's address, include an **Assign** activity in the workflow with the following assignment:

```
studentEntity.StudentAddressAssociation = Cmc.Nexus.Common.Entities.StudentAddressAssociation.IgnoreInStudentAssociation
```

Insert the Assign activity just prior to the **SaveEntity<StudentEntity>** activity.



If you change an address from Forms Renderer, it will not reflect as the Related Address in the Student Profile of a student in Anthology Student.

Student Advisor

When the this entity is selected, Form Designer exposes the following properties of the StudentAdvisorEntity class.

StudentAdvisorEntity

Field	Table : Column	Required	Type	Description	Example
Advisor Type	SyAdvisorByEnroll : AdvisorModule	No	String	Advisor type; MaxLength (8)	SA
Enrollment	SyAdvisorByEnroll : AdEnrollID	Yes	Int32	Enrollment identifier	9
Staff	SyAdvisorByEnroll : SyStaffID	Yes	Int32	Staff identifier	2
Staff Group	SyAdvisorByEnroll : SyStaffGroupID	Yes	Int32	Staff group identifier	3

Student Agency Branch

When the this entity is selected, Form Designer exposes the following properties of the StudentAgencyBranchEntity class.

StudentAgencyBranchEntity

Field	Table : Column	Required	Type	Description	Example
Branch	SyStudentAgency : AmAgencyBranchID	No	Int32	Agency branch identifier	9
End Date	SyStudentAgency : DateEnd	No	Nullable DateTime	End date	10/31/2017
Function	SyStudentAgency : StudentFunction	No	String	Student function; MaxLength(128)	Misc. office chores
Location	SyStudentAgency : Location	No	String	Location; MaxLength(60)	Visitor Center
Note	SyStudentAgency : Comments	No	String	Notes; MaxLength (600)	Added during schedule setup
Primary Billing Affiliate	SyStudentAgency : PrimaryBillingAffiliate	No	Boolean	Is this a primary billing affiliate?	true
Start Date	SyStudentAgency : DateStart	No	Nullable DateTime	Start Date	10/31/2016
Title	SyStudentAgency : Title	No	String	Title; MaxLength(40)	

Student Area of Study

When the this entity is selected, Form Designer exposes the following properties of the StudentAreaOfStudyEntity class.

StudentAreaOfStudyEntity

Field	Table : Column	Required	Type	Description	Example
Active	AdConcentrationByEnrollment : Active	Yes	Boolean	Is this area of study active?	true
Area Of Study	AdConcentrationByEnrollment : AdConcentrationID	Yes	Int32	Area of study identifier	7
Drop Date	AdConcentrationByEnrollment : DateDropped	No	Nullable DateTime	Date dropped	10/31/2016
GPA	AdConcentrationByEnrollment : GPA	No	Nullable Decimal	Grade point average	3.5

Student Athletic Detail

When the this entity is selected, Form Designer exposes the following properties of the StudentAthleticDetailEntity class.

StudentAthleticDetailEntity

Field	Table : Column	Required	Type	Description	Example
Athletic Status Id	SsAthleticDetail : SsAthleticStatusID	No	Int32	Athletic status identifier	5
Last Active Term Id	SsAthleticDetail : AdTermID	No	Int32	Last active term identifier	7
Recruitment Type Id	SsAthleticDetail : SsRecruitmentTypeID	No	Int32	Recruitment type identifier	3
Remaining Eligibility	SsAthleticDetail : RemainingEligibility	No	Int32	Remaining eligibility identifier	22
Sport Id	SsAthleticDetail : SsSportsID	No	Int32	Sport identifier	105

Student Course

When the this entity is selected, Form Designer exposes the following properties of the StudentCourseEntity class.

StudentCourseEntity

Field	Table : Column	Required	Type	Description	Example
Class Section Id	AdEnrollSched : AdClassSchedID	No	Nullable Int32	Class section identifier	15
Course	AdEnrollSched : AdCourseID	Yes	Int32	Course identifier	7
Course Status	AdEnrollSched : Status	No	String	Course status; MaxLength(1)	F
Drop Date	AdEnrollSched : DropDate	No	Nullable DateTime	Drop date	12/31/2016
Letter Grade	AdEnrollSched : AdGradeLetterCode	No	String	Letter grade; MaxLength(8)	A+
Numeric Grade	AdEnrollSched : NumericGrade	No	Nullable Decimal	Numeric grade	75.00
Start Date	AdEnrollSched : StartDate	No	Nullable DateTime	Start date	10/31/2016
Term	AdEnrollSched : AdTermID	No	Nullable Int32	Term identifier	1

Student Credit Card

When the this entity is selected, Form Designer exposes the following properties of the StudentCreditCardEntity class.

StudentCreditCardEntity

Field	Table : Column	Required	Type	Description	Example
Card Holder Apartment Number	SaCC : AptNumber	No	String	Card holder apartment number; MaxLength (10)	E-240
Card Holder Name	SaCC : CardHolderName	No	String	Card holder name; MaxLength(50)	Mario Romanov
Card Number	SaCC : Number	No	String	Credit card number; MaxLength(20)	1111-1111-2222-2222
Card Type	SaCC : SaCCTypeID	No	Nullable<Int32>	Credit card type identifier	3
City	SaCC : CardHolderCity	No	String	Card holder city; MaxLength(30)	Miami
Default	SaCC : DefaultCC	No	Boolean	Is this the default card?	true
Expiration Date	SaCC : Expire	No	DateTime	Expiration date	10/31/2020
First Name	SaCC : CardHolderFirstname	No	String	Card holder first name; MaxLength(50)	Mario
Is Active	SaCC : active	No	Boolean	Is this card active?	true
Is Primary Payment	SaCC : PrimaryPayment	No	Boolean	Is this the primary payment method?	true
Last Name	SaCC : CardHolderLastname	No	String	Card holder last name; MaxLength(50)	Romanov
State	SaCC : CardHolderState	No	String	Card holder state; MaxLength(2)	FL

Field	Table : Column	Required	Type	Description	Example
Street Address	SaCC : CardHolderAddr	No	String	Card holder street address; MaxLength(50)	12 Turkey Lane
Verification Number	SaCC : VerificationNumber	No	String	Verification number; MaxLength(4)	4321
ZIP Code/Postal Code	SaCC : CardHolderZip	No	String	Card holder ZIP/postal code; MaxLength(15)	33071

Student Disability Detail

When the this entity is selected, Form Designer exposes the following properties of the StudentDisabilityDetailEntity class.

StudentDisabilityDetailEntity

Field	Table : Column	Required	Type	Description	Example
Disability Status	SsStudentDisabilityDetail : SsDisabilityStatusID	No	Nullable Int32	Disability status identifier	5
Is Disabled	SsStudentDisabilityDetail : Disabled	No	String	Is this student disabled? MaxLength (1)	Y
Needs Registration Assistance	SsStudentDisabilityDetail : RegistrationAssistance	No	String	Does this student require registration assistance?	Y
Note	SsStudentDisabilityDetail : Comments	No	String	Note; MaxLength (2000)	Needs transportation assistance
Priority Registration	SsStudentDisabilityDetail : PriorityRegistration	No	String	Is this a priority registration? MaxLength (1)	Y

Student Enrollment Period

When the this entity is selected, Form Designer exposes the following properties of the StudentEnrollmentPeriodEntity class.

StudentEnrollmentPeriodEntity

Field	Table : Column	Required	Type	Description	Example
Academic Advisor	AdEnroll : AdAdvisorID	No	Nullable Int32	Academic advisor identifier	8
Applicant Type	AdEnroll : AmApplicantTypeID	No	Nullable Int32	Applicant type identifier	2
Application Date	AdEnroll : AppRecDate	No	Nullable DateTime	Application received date	10/31/2016
Campus	AdEnroll : SyCampusID	Yes	Int32	Campus identifier	1
Enrollment Date	AdEnroll : EnrollDate	No	Nullable DateTime	Enrollment date	10/31/2016
Enrollment Number	AdEnroll : StuNum	No	String	Enrollment number; MaxLength(10)	3214498
Expected Start Date	AdEnroll : ExpStartDate	No	Nullable DateTime	Expected start date	10/31/2016
Externship Start Date	AdEnroll : ExternBeginDate	No	Nullable DateTime	Externship start date	10/31/2016
FA Entrance Interview Date	AdEnroll : FaEntrDate	No	Nullable DateTime	Financial aid entrance Int32erview date	10/31/2016
Graduation Date	AdEnroll : GradDate	No	Nullable DateTime	Graduation date	10/31/2018
Previous Education	AdEnroll : amPrevEducID	No	Nullable Int32	Previous education identifier	2
Program	AdEnroll : AdProgramID	No	Nullable Int32	Program identifier	23
Program Version	AdEnroll : adProgramVersionID	No	Nullable Int32	Program version identifier	10
Program Version Start Date	AdEnroll : StartDate	No	Nullable DateTime	Program version start date (actual start date)	10/31/2016
Reentry Date	AdEnroll : ReEntryDate	No	Nullable DateTime	Re-entry date	10/31/2016
Shift	AdEnroll : adShiftID	No	Nullable Int32	Shift identifier	1

Field	Table : Column	Required	Type	Description	Example
Start Date	AdEnroll : AdStartDateID	No	Nullable DateTime	Start date identifier	10
Start Term	AdEnroll : adTermID	No	Nullable Int32	Start term identifier	1
Student	Refer to Student.				

Student Ethnicity

When the this entity is selected, Form Designer exposes the following properties of the StudentEthnicityEntity class.

StudentEthnicityEntity

Field	Table : Column	Required	Type	Description	Example
Ethnicity	SyStudentAmRace : EthnicityId	Yes	Int32	Ethnicity identifier	4

Refer to Multiselect for Single Property Collections.

Student Extra Curricular Activity

When the this entity is selected, Form Designer exposes the following properties of the StudentExtraCurricularActivityEntity class.

StudentExtraCurricularActivityEntity

Field	Table : Column	Required	Type	Description	Example
Extra Curricular Activity Id	AmProspectExtraCurr : AmExtraCurrID	No	Int32	Extra curricular activity identifier	11
Primary	AmProspectExtraCurr : PrimaryExtraCurr	No	Boolean	Is this a primary extra curricular activity?	true

Student Ledger Card Transaction

When the this entity is selected, Form Designer exposes the following properties of the StudentAccountTransactionEntity class.

StudentAccountTransactionEntity

Field	Table : Column	Required	Type	Description	Example
Academic Year	SaTrans : AcademicYear	No	Decimal	Academic year sequence	1
Amount	SaTrans : Amount	No	Decimal	Transaction amount	120.00
Campus Id	SaTrans : SyCampusID	No	Int32	Campus identifier	1
Check Number	SaTrans : CheckNo	No	String	Check number; MaxLength (20)	123412341234
Credit Card	SaTrans : SaCCID	No	Nullable<Int32>	Student credit card identifier	0
Payment Period	SaTrans : FaStudentAyPaymentPeriodId	No	Nullable<Int32>	Student academic year payment period identifier	1
Payment Type	SaTrans : PaymentType	No	String	Payment type; MaxLength (1)	H
Student Enrollment Period Id	SaTrans : AdEnrollID	No	Int32	Student enrollment period identifier	1
Term	SaTrans : AdTermID	No	Int32	Term identifier	2
Transaction Code	SaTrans : SaBillCode	No	String	Billing transaction code; MaxLength (8)	12341234
Transaction Date	SaTrans : Date	No	DateTime	Transaction date	10/31/2016

Field	Table : Column	Required	Type	Description	Example
Transaction Name	SaTrans : Descrip	No	String	Description; MaxLength (100)	Housing fee

Student Previous Education

When the this entity is selected, Form Designer exposes the following properties of the StudentPreviousEducationEntity class.

StudentPreviousEducationEntity

Field	Table : Column	Required	Type	Description	Example
Degree	AmProspectPrevEduc : DegreeID	No	Nullable Int32	Degree identifier	67
Enrollment Date	AmProspectPrevEduc : DateOfEnrollment	No	Nullable DateTime	Enrollment date	10/31/2016
GED Awarded Date	AmProspectPrevEduc : GEDAwardedDate	No	Nullable DateTime	GED awarded date	10/31/2016
GPA	AmProspectPrevEduc : GPA	No	Nullable Decimal	Grade point average	4.0
Grade Level	AmProspectPrevEduc : AmGradeLevelID	No	Nullable Int32	Grade level identifier	4
Graduated	AmProspectPrevEduc : Graduated	No	Boolean	Did this student graduate?	true
Graduation Date	AmProspectPrevEduc : GraduationDate	No	Nullable DateTime	Graduation date	10/31/2015
Graduation Session	AmProspectPrevEduc : GraduationSession	No	String	Graduation session; MaxLength(10)	Spring
Graduation Year	AmProspectPrevEduc : HSGradYear	No	String	Graduation year; MaxLength(4)	2015
High School	AmProspectPrevEduc : AmHighSchoolID	No	Nullable Int32	Institution (high school) identifier	3216
Institution	AmProspectPrevEduc : AmCollegeID	No	Nullable Int32	Institution (college) identifier	341
Last Date Attended	AmProspectPrevEduc : LastDateAttended	No	Nullable DateTime	Last date attended	10/31/2015
Major	AmProspectPrevEduc : Major	No	String	Major; MaxLength (100)	Biology
Note	AmProspectPrevEduc : Comments	No	String	Note; MaxLength (2000)	Potential candidate
Online Application Other College	AmProspectPrevEduc : OtherCollege	No	String	Online application other college; MaxLength(60)	FIU

Field	Table : Column	Required	Type	Description	Example
Rank	AmProspectPrevEduc : StudRank	No	Nullable Int32	Student rank	45
School Size	AmProspectPrevEduc : SchoolSize	No	Nullable Int32	School Size	900
Total Credits Attempted	AmProspectPrevEduc : CreditsAttempt	No	Nullable Decimal	Total credit hours attempted	120
Total Grade Points	AmProspectPrevEduc : GradePoints	No	Nullable Decimal	Total grade points	3.5
Total Hours Attempted	AmProspectPrevEduc : HoursAttempt	No	Decimal	Total clock hours attempted	30.5
Total Number of Courses	AmProspectPrevEduc : CoursesTaken	No	Nullable Int32	Total number of courses taken	4
Total Quality Points	AmProspectPrevEduc : QualityPoints	No	Nullable Decimal	Total quality points	4.0
Transcript Type	AmProspectPrevEduc : TranscriptID	No	Nullable Byte	Transcript Type	0

Student Relationship Address

When the this entity is selected, Form Designer exposes the following properties of the StudentRelationshipAddressEntity class.

StudentRelationshipAddressEntity

Field	Table : Column	Required	Type	Description	Example
Address Type	SyAddress : SyAddrTypeID	No	Int32	Address type identifier	2
City	SyAddress : City	No	String	City; MaxLength (25)	Chicago
Country	SyAddress : SyCountryID	No	Nullable Int32	Country identifier	1
County	SyAddress : SyCountyID	No	Nullable Int32	County identifier	88
Email Address	SyAddress : Email	No	String	Email address; MaxLength(50)	mario@anthology.com
Employer Name	SyAddress : Employer	No	String	Employer name; MaxLength(50)	Anthology Inc.
End Date	SyAddress : EndDate	No	Nullable DateTime	Address end date	10/31/2014
First Name	SyAddress : FirstName	No	String	First name; MaxLength(25)	Mario
Is Permanent Address	SyAddress : Yearly	No	Boolean	Is this a permanent address?	true
Last Name	SyAddress : LastName	No	String	Last name; MaxLength(25)	Romanov
Note	SyAddress : Comments	No	String	Note	Verify employer
Phone Number	SyAddress : Phone	No	String	Phone number; MaxLength(16)	123-666-8989
Relation To Student	SyAddress : RelationToStudent	No	String	Relation to student; MaxLength(30)	Father
Second Phone Number	SyAddress : SecondPhone	No	String	Second phone number; MaxLength(16)	123-666-8990

Field	Table : Column	Required	Type	Description	Example
Ssn	SyAddress : SSN	No	String	Social security number; MaxLength(11)	111-22-3333
Start Date	SyAddress : BeginDate	No	Nullable DateTime	Address start date	10/31/2002
State	SyAddress : State	No	String	State; MaxLength(2)	FL
Street Address	SyAddress : Addr1	No	String	Street address; MaxLength (255)	123 Turkey Lane
Student Enrollment Period Id	SyAddress : AdEnrollID	No	Int32	Student enrollment period identifier	1
Title	SyAddress : AmTitleID	No	Nullable Int32	Title identifier	1
Work Phone Number	SyAddress : WorkPhone	No	String	Work phone number; MaxLength(16)	954-333-9090
Years At Address	SyAddress : YearsAtAddress	No	Nullable Decimal	Years at address	14.5
Years Known Student	SyAddress : YearsKnownStudent	No	Nullable Decimal	Years known student	22
ZIP Code / Postal Code	SyAddress : Zip	No	String	ZIP code / postal code; MaxLength(20)	33075

Student Service Type

When the this entity is selected, Form Designer exposes the following properties of the StudentServiceTypeEntity class.

StudentServiceTypeEntity

Field	Table : Column	Required	Type	Description	Example
Cost	SsStudentServiceAssociation : Amount	No	Nullable Decimal	Cost of service	1500.00
Service Begin Date	SsStudentServiceAssociation : BeginDate	No	Nullable DateTime	Service begin date	10/31/2016
Service End Date	SsStudentServiceAssociation : EndDate	No	Nullable DateTime	Service end date	10/31/2017
Service Type	SsStudentServiceAssociation : SsServiceID	Yes	Int32	Service type identifier	10
Status	SsStudentServiceAssociation : Status	No	Boolean	Status	true
Student Enrollment Period Id	SsStudentServiceAssociation : AdEnrollID	No	Nullable Int32	Student enrollment period identifier>	1
Term	SsStudentServiceAssociation : AdTermID	No	Nullable Int32	Term identifier	1

Student Transfer Credit

When the this entity is selected, Form Designer exposes the following properties of the StudentTransferCreditEntity class.

StudentTransferCreditEntity

Field	Table : Column	Required	Type	Description	Example
Approved Date	AmCollegeTransfer : DateApproved	No	Nullable DateTime	Approved date	10/31/2016
Campus Course	AmCollegeTransfer : AdCourseID	No	Nullable Int32	Campus course identifier	12
College Course Code	AmCollegeTransfer : CollegeCourseCode	No	String	College course code; MaxLength (16)	ENG101
College Course Description	AmCollegeTransfer : CollegeCourseDescrip	No	String	College course description; MaxLength (75)	English composition
College Name	AmCollegeTransfer : CollegeName	No	String	College name; MaxLength (60)	FIU
Course	AmCollegeTransfer : AdCourseID	No	Nullable<Int32>	Course identifier	13
Credits	AmCollegeTransfer : CollegeCourseCredits	No	Decimal	Credits	4.00
Credits Earned	AmCollegeTransfer : CollegeCourseCreditsEarned	No	Nullable<Decimal>	Credits earned	4.00
Credits to Transfer	AmCollegeTransfer : CreditsEarned	No	Decimal	Credits to transfer	4.00
Date Completed	AmCollegeTransfer : CompletionDate	No	DateTime	College course date completed	10/31/2015
Date Started	AmCollegeTransfer : StartDate	No	Nullable<DateTime>	College course date started	10/31/2014

Field	Table : Column	Required	Type	Description	Example
Grade	AmCollegeTransfer : Grade	No	String	Letter Grade; MaxLength (8)	A-plus
Grade Points	AmCollegeTransfer : CollegeCourseGradePoints	No	Nullable<Decimal>	Grade points	4.00
Grade Received	AmCollegeTransfer : CollegeCourseGrade	No	String	Grade received; MaxLength (3)	B
Hours	AmCollegeTransfer : ClockHours	No	Nullable<Decimal>	Clock hours attempted	50.00
Hours Attempted	AmCollegeTransfer : CollegeCourseHoursAttempt	No	Nullable<Decimal>	Hours attempted	36.00
Hours Earned	AmCollegeTransfer : CollegeCourseHoursEarned	No	Nullable<Decimal>	Hours earned	30.00
Hours to Transfer	AmCollegeTransfer : HoursEarned	No	Nullable<Decimal>	Hours to transfer	30.00
Institution	AmCollegeTransfer : AmCollegeID	No	Nullable<Int32>	Institution identifier	4
Term	AmCollegeTransfer : AdTermID	No	Nullable Int32	Term identifier	1
Transfer Credit Status	AmCollegeTransfer : AmCollegeTransferStatusID	No	Nullable Int32	Transfer credit status identifier	2
Transfer Credit Type	AmCollegeTransfer : AmTransferTypeID	No	Nullable Int32	Transfer credit type identifier	1

Student Veteran Detail

When the this entity is selected, Form Designer exposes the following properties of the StudentVeteranDetailEntity class.

StudentVeteranDetailEntity

Field	Table : Column	Required	Type	Description	Example
Certification Type	SsStudentVeteranDetail : SsVeteranCertificationTypeID	No	Nullable Int32	Veteran Affairs certification type identifier	4
Last Certified Term	SsStudentVeteranDetail : AdTermID	No	Nullable Int32	Last certified term identifier	1

Student_Staff Picture

When the this entity is selected, Form Designer exposes the following properties of the PersonPictureEntity class.

PersonPictureEntity

Field	Table : Column	Required	Type	Description	Example
Is Student	CmStudentPicture : Student	No	Boolean	Is this a student?	true

Cheat Sheets

Cascading Style Sheets:	CSS Cheat Sheet
HTML5:	HTML5 Visual Cheat Sheet
AngularJS expressions:	https://docs.angularjs.org/guide/expression
Comparison and logical operators:	https://www.w3schools.com/js/js_comparisons.asp
Regular expressions:	http://regexlib.com/CheatSheet.aspx

Exposed Events

Forms Builder was enhanced to expose specific events from rendered components outside of the Renderer application domain. There are multiple uses to this capability. For example, analytics tools can access this data to provide metrics on how users of the rendered sequences are completing the forms within the sequence.

This enhancement also gives developers access to the raised events. It makes it easier to write custom JavaScript code by constructing a JavaScript function with the name of the event handler plus parameter.

Form Events

The form events occur at the beginning of a sequence when navigating to a sequence and at the end of a sequence when navigating to a new URL, i.e., away from the current sequence.

Since form events are very sensitive to timing, we recommend placing the HTML component with the event handlers in the Welcome page (or at least the page before the form where the events occur). This ensures that the event handlers are defined before the events occur.

```
vmEventHandlersRef.cmcFormViewContentLoaded = function(payload){
Occurs when form view is completely loaded
     payload object contains:
       models - reference to the models passed to and from a workflow
       rootScope - angular $rootScope
}

vmEventHandlersRef.cmcFormRouteChangeStart = function(payload){
     payload object contains:
       models - reference to the models passed to and from a workflow (may not be defined for certain
routes)
       rootScope - angular $rootScope
       fromRoute - route from which we are coming
       toRoute - route to which we are going
       event - event object for routeChangeStart
}

vmEventHandlersRef.cmcFormRouteChangeSuccess = function(payload){
     payload object contains:
       models - reference to the models passed to and from a workflow
       rootScope - angular $rootScope
       event - event object for routeChangeSuccess
       current - current route
       previous - previous route
}
```

Component Events

Event Scheduler (Calendar/Scheduler Component)

```
vmEventHandlersRef.cmcEventSchedulerInitialized = function(payload){
    payload object contains:
```

```
       key - used to address data about pending OData query
       control - reference to the control itself
       id - id of the control
       isOdataQuery - true or false
       element - reference to the DOM element for the control
       query - value of the OData query
       models - reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcEventSchedulerDataBound = function(payload){
    payload object contains:
       key - used to address data about pending OData query
       control - reference to the control itself
       id - id of the control
       isOdataQuery - true or false
       element - reference to the DOM element for the control
       query - value of the OData query
       models - reference to the model values passed to and from workflow
       data - data bound to the scheduler
       firstBind - true or false
}
```

Checkbox

```
vmEventHandlersRef.cmcCheckBoxChange = function(payload){
     payload object contains:
       element - reference to the DOM element of the control
       id - id of the control
       value - the value will be true if the checkbox selected; false if unselected
       models - reference to the model values passed to and from workflow
}
```

Date Picker

```
vmEventHandlersRef.cmcDatePickerChange = function(payload){

If "Ignore Time" is false:
     payload object contains:
       value - Model value. Datetime object with offset. This value is weighted by its offset. Since
the object of this component is to pick a date only, this is undesirable if the time will cause
the date to shift either one day back or one day forward depending on the time zone the browser
is located in compared to the actual time. To avoid this, "Ignore Time" should be set to true.
       formattedDate - ISO 8601 format date, time, and offset string
       fullDateTime - culture dependent value of the date and time
       isoDateTime - ISO 8601 format date represented as UTC
       control - reference to the control itself
       validator - reference to a validator which will contain a validate() method
       id - id of the control
       models - reference to the model values passed to and from workflow

If "Ignore Time" is true (recommended for most situations):
     payload object contains:
       control - reference to the control itself
       validator - reference to a validator which will contain a validate() method
       value - ISO 8601 format date. This will both take and produce an ISO 8601 date only string
value. A value set in a workflow will cause the ISO string returned to this control to have no
offset. This means that it will be the same date in any browser in the world, regardless of time
zone.
```

```
       id - id of the control
       models - reference to the model values passed to and from workflow
}
```

Date Time Picker

```
vmEventHandlersRef.cmcDateTimePickerInitialized = function(payload){
     payload object contains:
       control - reference to the control itself
       id - id of the control
       element - reference to the DOM element for the control
       validator - reference to a validator which will contain a validate() method
       models - reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcDateTimePickerChange = function(payload){
     payload object contains:
       control - reference to the control itself
       id - id of the control
       validator - reference to a validator which will contain a validate() method
       value - Model value. ISO 8601 format date, time, and offset string.
       models - reference to the model values passed to and from workflow
}
```

Drop-down List

```
vmEventHandlersRef.cmcDropDownListInitialized = function(payload){
     payload object contains:
       control - reference to the control itself
       id - id of the control
       element - reference to the DOM element for the control
       validator - reference to a validator which will contain a validate() method
       reload - function to reload the control
       component - reference to component that encapsulates control
       models - reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcDropDownListSelect = function(payload){
     payload object contains:
       sender - reference to the drop-down list control itself
       text - the text field before selection
       value - the value field before selection
       validator - reference to a validator which will contain a validate() method
       id - id of the control
       models - reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcDropDownListChange = function(payload){
     payload object contains:
       sender - reference to the control itself
       isInList - true or false
       text - the text field for the selection
       value - the value field for the selection
       validator - reference to a validator which will contain a validate() method
       id - id of the control
       models - reference to the model values passed to and from workflow
}
```

```
vmEventHandlersRef.cmcDropDownListDataBound = function(payload){
     payload object contains:
       sender – reference to the drop-down list control itself
       id – id of the control
       models – reference to the model values passed to and from workflow
}
```

Grid

```
vmEventHandlersRef.cmcGridInitialized = function(payload){
     payload object contains:
       key – used to address data about pending OData query. Can be main query or column query.
       control – reference to the control itself
       queryType – "mainOData" or "dropdownlist". Type of query
       id – id of the control
       isOdataQuery – true or false
       element – reference to the DOM element for the control
       query – value of the OData query
       models – reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcGridDataBound = function(payload){
     payload object contains:
       key – used to address data about pending OData query. Can be main query or column query.
       id – id of control
        data – data bound to the grid
       queryType – "mainOData" or "dropdownlist". Type of query
       control – reference to the control itself
       element – reference to the DOM element for the control
       isOdataQuery – true or false
       query – value of the OData query
       firstBind - true or false, whether this is the first dataBound call
       models – reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcGridAngularOnChanges = function(payload){
     payload object contains:
       changes – object containing information about what changed in the control directives, e.g.,
fbColumns and fbOdataQuery are two changes. isFirstChange can be checked to ignore ini-
tialization.
       control – reference to the control itself
       id – id of the control
       element – reference to the DOM element for the control
       models – reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcGridSyncModel = function(payload){
Occurs when model is updated
     payload object contains:
       control – reference to the control itself
       id – id of the control
       element – reference to the DOM element for the control
       data – data used to update the model
       models – reference to the model values passed to and from workflow
       keys – model properties updated
       model – model property updated
}
```

```
vmEventHandlersRef.cmcGridEdit = function(payload){
Occurs when data is edited
     payload object contains:
       control - reference to the control itself
       id - id of the control
       element - reference to the DOM element for the control
       data - data used to update the model
       models - reference to the model values passed to and from workflow
       event - event parameter document for Kendo Grid edit event
}

vmEventHandlersRef.cmcGridRemove = function(payload){
Occurs when data is removed
     payload object contains:
       control - reference to the control itself
       id - id of the control
       element - reference to the DOM element for the control
       data - data used to update the model
       models - reference to the model values passed to and from workflow
       event - see event parameter documentation for Kendo Grid remove event
}

vmEventHandlersRef.cmcGridSave = function(payload){
Occurs when data is saved
     payload object contains:
       control - reference to the control itself
       id - id of the control
       element - reference to the DOM element for the control
       data - data used to update the model
       models - reference to the model values passed to and from workflow
       event - see event parameter documentation for Kendo Grid save event
}
```

Hyperlink

```
vmEventHandlersRef.cmcHyperlinkClick = function(payload){
     payload object contains:
       id - id of the control
       href - url for the hyperlink
       text - text of the hyperlink
       models - reference to the models passed to and from a workflow
       thisObj - reference to "this", the angular scope object
       models - reference to the model values passed to and from workflow
}
```

Masked Text Box

```
vmEventHandlersRef.cmcMaskedTextBoxBlur = function(payload){
     payload object contains:
       control - reference to the control itself
       id - id of the control
       value - value of the control
       element - reference to the DOM element for the control
       validator - reference to a validator which will contain a validate() method
       models - reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcMaskedTextBoxChange = function(payload){
```

```
     payload object contains:
       control - reference to the control itself
       id - id of the control
       value - value of the control
       element - reference to the DOM element for the control
       validator - reference to a validator which will contain a validate() method
       models - reference to the model values passed to and from workflow
}
```

Multiselect

```
vmEventHandlersRef.cmcMultiSelectInitialized = function(payload){
     payload object contains:
       control - reference to the control itself
       id - id of the control
       element - reference to the DOM element for the control
       validator - reference to a validator which will contain a validate() method
       models - reference to the model values passed to and from workflow
       reload - function to refresh the control
}

vmEventHandlersRef.cmcMultiSelectDataBound = function(payload){
     payload object contains:
       sender - reference to the drop-down list control itself
       id - id of the control
       models - reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcMultiSelectChange = function(payload){
     payload object contains:
       control - reference to the control itself
       element - reference to the DOM element of the control
       validator - reference to a validator which will contain a validate() method
       values - array of values selected
       id - id of the control
       models - reference to the model values passed to and from workflow
}
```

Numeric Text Box

```
vmEventHandlersRef.cmcNumericTextBoxInitialized = function(payload){
     payload object contains:
       control - reference to the control itself
       id - id of the control
       element - reference to the DOM element for the control
       validator - reference to a validator which will contain a validate() method
       models - reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcNumericTextBoxBlur = function(payload){
     payload object contains:
       control - reference to the control itself
       id - id of the control
       value - value of control before the blur
       element - reference to the DOM element for the control
       models - reference to the model values passed to and from workflow
}
```

```
vmEventHandlersRef.cmcNumericTextBoxChange = function(payload){
Note this event does not fire when the spin buttons increment and decrement values. The spin
event was not implemented in the core-ui component.
     payload object contains:
       control - reference to the control itself
       id - id of the control
       value - value of the control
       element - reference to the DOM element for the control
       validator - reference to a validator which will contain a validate() method
       models - reference to the model values passed to and from workflow
}
```

Repeater

```
vmEventHandlersRef.cmcRepeaterInitialized = function(payload){
Occurs when repeater is initialized
     payload object contains:
         control - reference to the control itself
         id - id of the control
         element - reference to the DOM element for the control
         models - reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcRepeaterRowAdd = function(payload){
Occurs when repeater row is added
     payload object contains:
         control - reference to the control itself
         id - id of the control
         element - reference to the DOM element for the control
         models - reference to the model values passed to and from workflow
         currentIndex - index of last repeater row before add
         event - angular event for add button
}

vmEventHandlersRef.cmcRepeaterRowDelete = function(payload){
Occurs when repeater row is deleted
     payload object contains:
         control - reference to the control itself
         id - id of the control
         element - reference to the DOM element for the control
         models - reference to the model values passed to and from workflow
         deletedIndex - index of row deleted
         currentIndex - index of last row after delete
         event - angular event for delete button
}
```

Radio Button

```
vmEventHandlersRef.cmcRadioButtonChange = function(payload){
     payload object contains:
       element - reference to the DOM element of the control
       id - id of the control
       value - value of the selected radio button in a group of radio buttons
       models - reference to the model values passed to and from workflow
}
```

Single-select Search

Note: The template for the Single-select Search control was updated. To pick up the new template and access the exposed events, the form containing the control needs to be re-saved .

```
vmEventHandlersRef.cmcSingleSelectSearchInitialized = function(payload){
    payload object contains:
       component - reference to the control itself
       id – id of control
       element – reference to the DOM element for the control
       validator – reference to a validator which will contain a validate() method
       odataPromise – promise for OData request - unresolved
       resetOriginal – function to reload the control
       models – reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcSingleSelectSearchDataBound = function(payload){
    payload object contains:
       component - reference to the control itself
       id – id of control
       element – reference to the DOM element for the control
       validator – reference to a validator which will contain a validate() method
       odataPromise – promise for OData request - resolved
       models – reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcSingleSelectSearchChange = function(payload){
    payload object contains:
       id – id of control
       value – value of control
       element – reference to the DOM element for the control
       validator – reference to a validator which will contain a validate() method
       models – reference to the model values passed to and from workflow
}
```

Text Box

```
vmEventHandlersRef.cmcTextBoxChange = function(payload){
     payload object contains:
       id – id of the control
       value – value of control
       element – reference to the DOM element for the control
       models – reference to the model values passed to and from workflow
}

vmEventHandlersRef.cmcTextBoxBlur = function(payload){
     payload object contains:
       id – id of the control
       value – value of control
       element – reference to the DOM element for the control
       models – reference to the model values passed to and from workflow
}
```

Time Picker

```
vmEventHandlersRef.cmcTimePickerChange = function(payload){
    payload object contains:
       value – datetime object. Convert to string with .toString()
       control – reference to the control itself
       formattedValue – formatted per format specification
```

```
        id - id of the control
        validator - reference to a validator which will contain a validate() method
        models - reference to the model values passed to and from workflow
}
```

Examples of External Event Listeners

All events except form events have an id. You can put event listeners in a non-visible HTML component on the page. A test that would verify your code could be executed would be to prove the event occurred:

```
<script>
vmEventHandlersRef.cmcMaskedTextBoxChange = function(payload)  {
        // An alert can be used to pop up a window
        alert("Got cmcMaskedTextBoxChange for id " + payload.id);

        // Or you can simply write a message to the console window
        console.log("Got cmcMaskedTextBoxChange for id " + payload.id);
}
</script>
```

Note that this can be debugged in the debuggers built in to Firefox, Chrome, Edge, IE, etc. Refer to external documentation on using these.

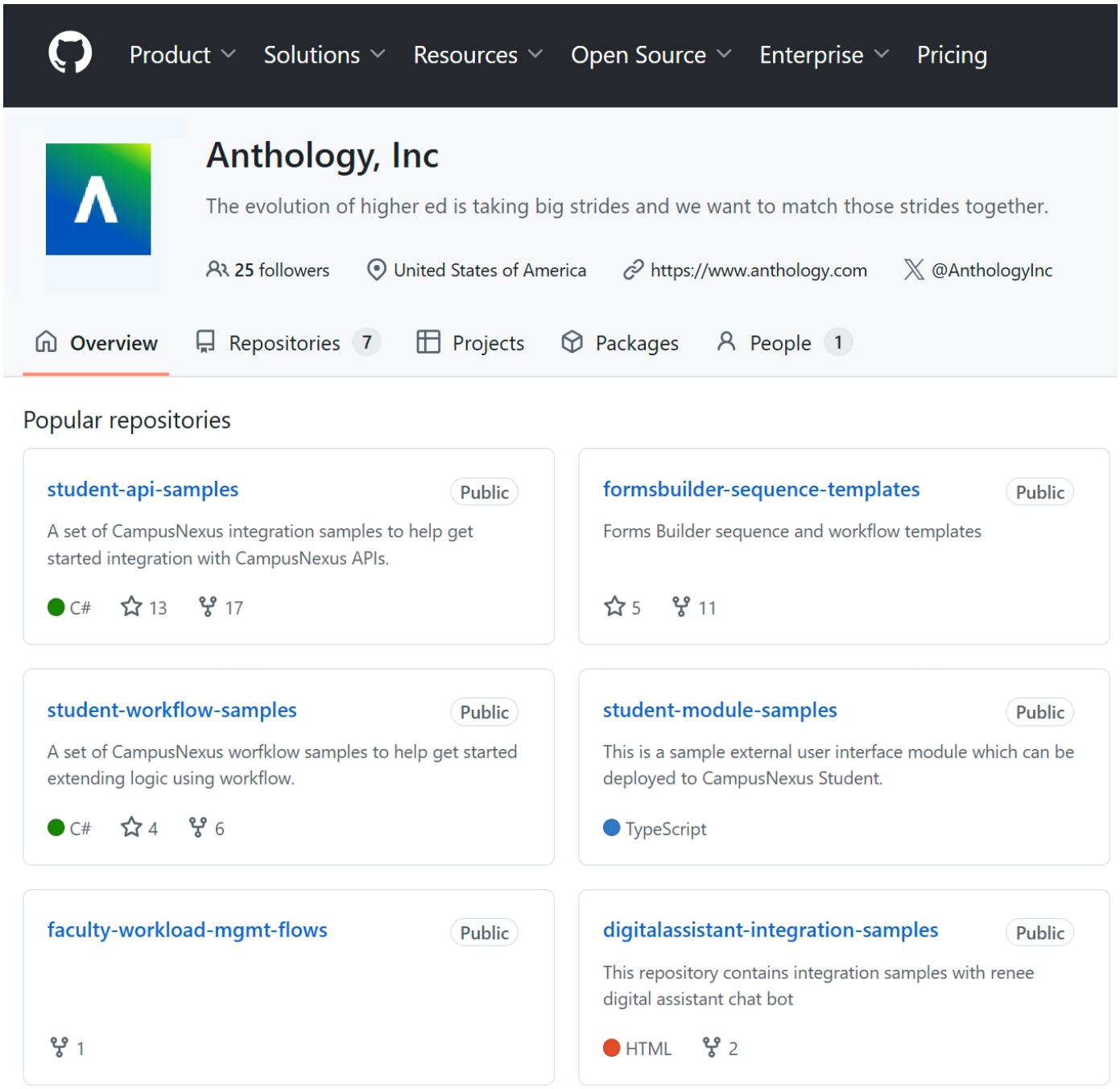
Advanced external events – the sky is the limit in what you could get your own code to do based on an event on a form. Example: call an external website to do something. You want to increment a counter every time a student picks Zimbabwe as a country. This would be something you could capture from the drop-down list and in a cmcDropDownListChange event handler you could send a post to an external website crafted for that purpose, but conditionally, only if the “text” property was “Zimbabwe”. The external website call could be done with a JQuery $ajax call. Some of the libraries already built into Renderer are JQuery, Angularjs, LoDash, w3 (from w3schools.com), moment, kendo, rxjs.

Debugging tip: Add the statement “debugger;” to your external event handler code. This will cause it to stop in debug mode in your browser and you can then examine the values of variables and step forward through your code.

GitHub Repositories

Anthology Inc. has created a set of community-driven GitHub repositories to help share ideas, solutions, and knowledge about Anthology products. We would love for you to contribute to this community as well, so please feel free to fork a project and commit your proposed changes. Submitting a change will write to a new branch so you can send a pull request to Anthology to incorporate changes back into the master branch.

https://github.com/anthology-inc/



Forms Builder Sequence Templates

https://github.com/anthology-inc/fb-sequence-templates

The Forms Builder Sequence Templates repository includes a set of Forms Builder sequence and workflow templates for use with CampusNexus CRM and Anthology Student. These templates are helpful examples to get you started creating form sequences and workflows.

Note: It is important to understand the templates in the GitHub repository were created in one environment and can be shared and re-used in another environment. They are for the sole purpose of providing templates to customers as a starting point for building Student Self-Service processes using Forms Builder/Workflow in their own environments. Almost always, changes and/or updates are needed to these templates to ensure values/IDs are aligned with the customer's environment (database).

Workflow Samples

https://github.com/anthology-inc/workflow-samples

The Workflow Samples repository contains a set of sample workflows and workflow activities that you can easily import into your environment using Workflow Composer.

Integration Samples

https://github.com/anthology-inc/student-api-samples

The Integration Samples repository contains examples of consuming Anthology APIs using various technologies such as .NET, JavaScript, and Azure Logic Apps.

General GitHub Resources

https://guides.github.com/

The link above provides guides for getting started with GitHub, links to GitHub Help, Videos, and More.

OData Queries

Forms Builder supports list controls such as Drop-down, Multiselect, and Typeahead. Each list control has a Lookup Query property that is used to retrieve specific values from the database. Lookup Queries are specified as OData queries.

In Forms Builder 3.1 and later, most list controls include a default OData query in the model metadata which populates the Lookup Query property setting when that property is selected in Form Designer. In Forms Builder 3.0, however, most of the list controls do not have a default query. You can use the View feature in the Web Client of Anthology Student to construct queries or build your own OData queries using the Anthology data model as a reference.

Forms Builder 3.4 and later supports OData queries using Occupation Insight as a data source.

Important
OData calls must be made from **within** Forms Builder or Workflow.
OData calls via ApiProxy outside of Forms Builder or Workflow are **not** supported.

Build Queries Using Views for Anthology Student

Anthology Student provides the Views feature which enables you to create OData queries that can be used to populate the Lookup Query value for list Fields or Components in Form Designer.

To access Anthology Student, open the **About Forms Builder** window, and copy the **Student Base URL** into a browser.

Create a View and Export a Query

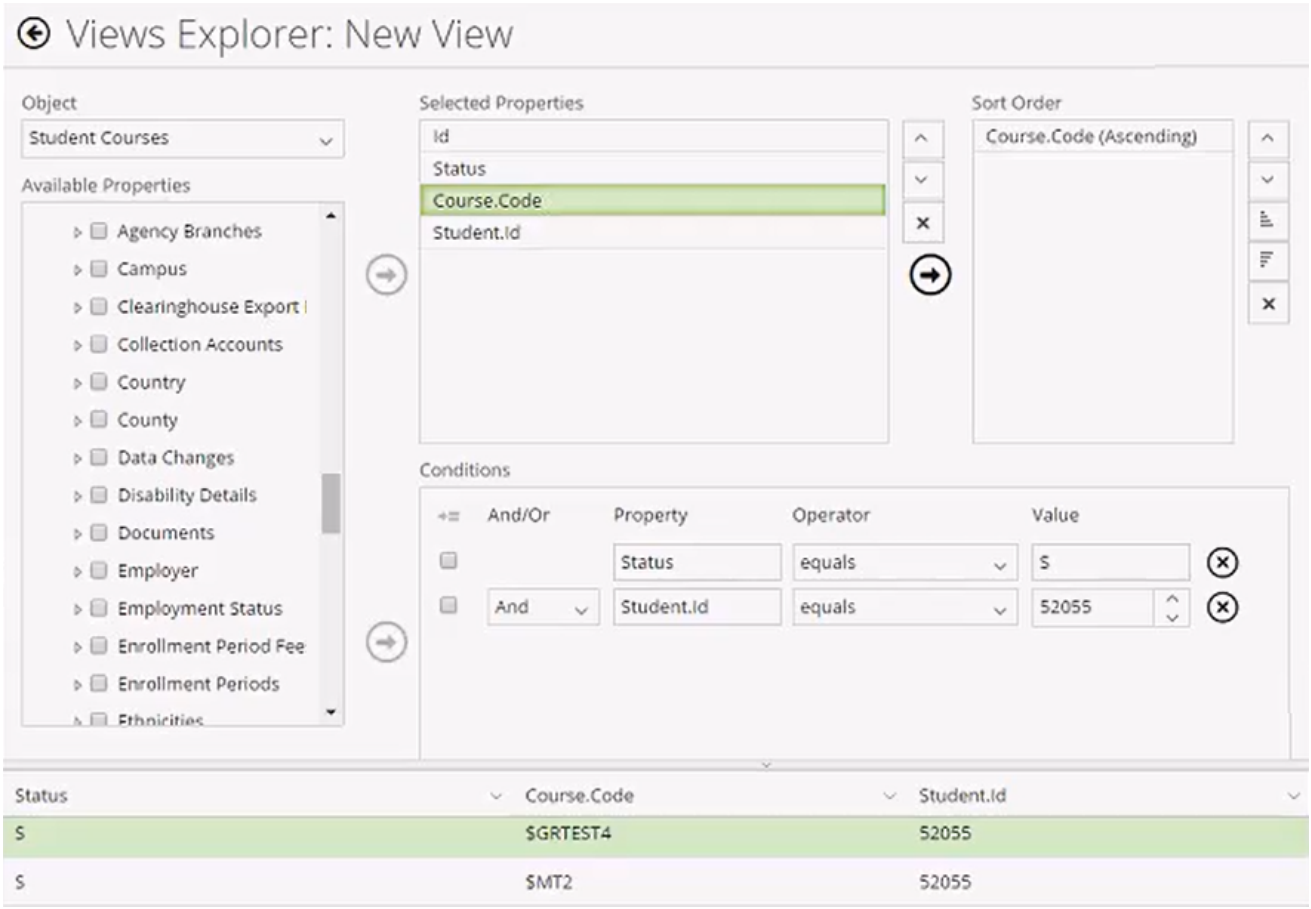
1. Log in to Anthology Student at the Student Base URL.
2. Select the **Views** tile.
3. Select the **New View** button.
4. Add data to the view by selecting appropriate values from the following categories:
   - **Modules** (e.g., Academics, Admissions, Career Services, Common.)
   - **Objects** (e.g., Programs, Buildings)
   - **Properties** (e.g., Active, Number of Rooms, Contact Name)

   Select  to move your selections to the Selected Properties box.
5. In the **Selected Properties** box, select the property you want to sort and select  to move your selection to the Sort Order box.
6. Select appropriate **Conditions** in the query to filter the data in the view (e.g., And/Or, Property, Operator, Value).
7. Select **Run Query** in the toolbar and review the results.

   In our example, the following query was constructed and executed:

   - The StudentCourses object was selected.
   - The Id, Status, Course.Code and Student.Id properties were selected.
   - The Sort Order is by Course.Code.
   - The Conditions filter by Status (S=scheduled) and StudentId.

   The query returns all scheduled student courses.



8. Select  at the top right of the toolbar to access the **Query URL**. The Copy the URL to the clipboard window is displayed.



9. Select **OK** and paste the URL into Notepad.

   In our example, the URL is as follows:

   ```
   http://cltqafb7.campusmgmt.com:80/Cmc.Nexus.Web/ds/campusnexus/StudentCourses?$filter=Status eq 'S' and Student/Id eq 52055&$orderby=Course/Code&$select=Id,Status&$expand=Course($select=Code),Student($select=Id)
   ```

   The URL contains both the Student Base URL and the OData query (highlighted).

   You can validate this query by pasting it in any browser window.

10. Paste the OData query into the Lookup Query field in Form Designer.

Notes:

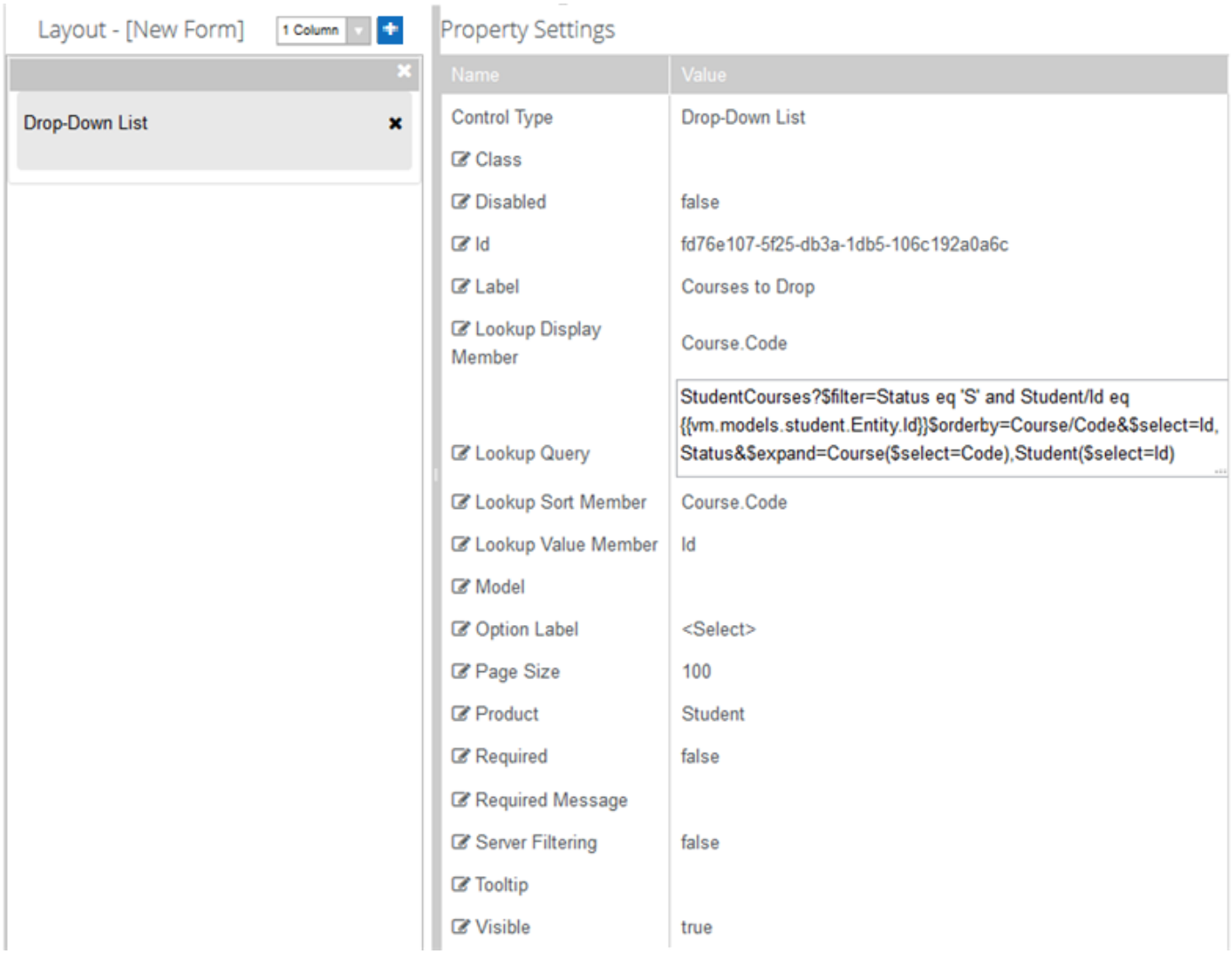
Replace the StudentId value (52055) with an expression representing the `studentEntity.Id` in the data model so that the query for student courses will be applicable to any student.

```
StudentCourses?$filter=Status eq 'S' and Student/Id eq {{vm.-
models.studentEntity.Id}}$orderby=Course/Code&$select=Id,Status&$expand=Course
($select=Code),Student($select=Id)
```

The above OData query assumes that `{{vm.models.studentEntity.Id}}` is populated before this query is being executed. The value of such variable can be populated in Workflow, in a previous form or even in the same form.

Populate the Lookup Query in Form Designer

After testing the OData query, paste it in the Lookup Query field of the list control in the Property Settings pane in Form Designer.



Notes:

- Depending on the columns retrieved by the query, adjust the Lookup Display Member, Lookup Sort Member, and Lookup Value Member properties. The default values "Name" and "Id" are not applicable if the query does not retrieve "Name" and "Id" values.
- The Product specified in the Property Settings pane is the query provider. When you are customizing a Lookup Query, make sure that the Product value (e.g., "Student") matches the database that is being queried.

Lookup Queries for CampusNexus CRM Metadata

For any drop-down or search controls that will be populated via a lookup query, the CampusNexus CRM user needs to enter values for the **Lookup Display Member** and **Lookup Sort Member** attributes. The **Lookup Query** and **Lookup Value Member** property settings should have default values (if applicable for the selected property) as these are currently specified in the metadata.

Build Queries Using the Data Model

The CampusNexusdata model provides most of the information that is displayed in the Property Settings pane in Form Designer. The CampusNexus data model is parsed and its metadata is extracted to populate the Property Settings. The data model is exposed and can be used as a reference to build OData queries in Forms Builder.

Anthology Student 21.2.0 and later requires users to be authorized to execute OData queries. For more information, refer to Security Enhancement for OData Queries.

> **Important**
> OData calls must be made from **within** Forms Builder or Workflow.
> OData calls via ApiProxy outside of Forms Builder or Workflow are **not** supported.

View the Metadata

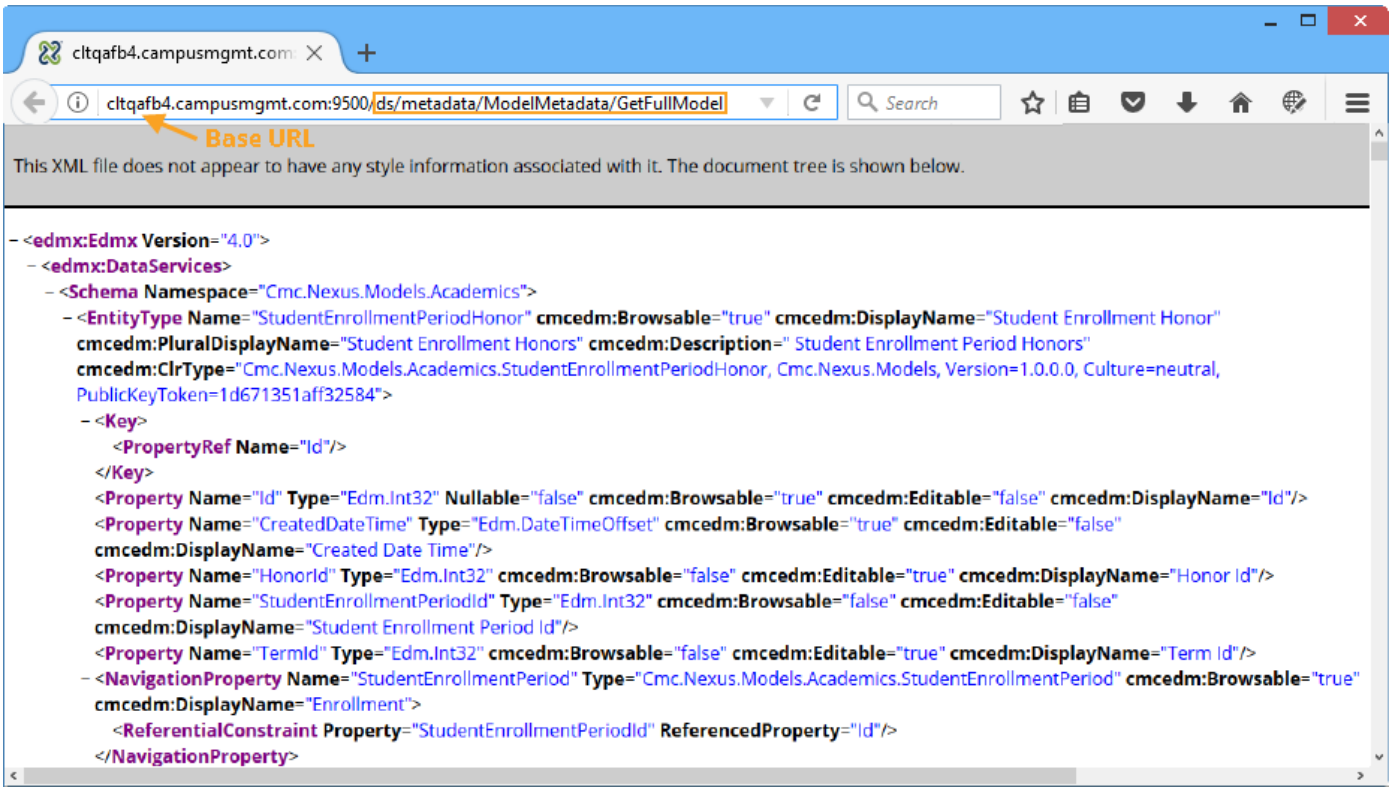
The CampusNexus data model provides a query model and a command model. For the purposes of constructing OData queries, refer to the query model. The query model is available at the following URL:

`<Base URL>/ds/metadata/ModelMetadata/GetFullModel`

Where `<Base URL>` is the Student Base URL displayed in the "About Forms Builder" window.

In this example, the query model is available at:

`http://cltqafb4.campusmgmt.com:9500/ds/metadata/ModelMetadata/GetFullModel`



Search the metadata for the entity you are working, e.g., "Students". The metadata for the "Student" entity provide several prebuilt OData queries. Note that all pre-built queries contain "select" options for the Code, Name, and Id columns and are ordered by Name.

Example: Student Entity Metadata (Excerpt)

-<EntityType cmcedm:Description="Students" cmcedm:PluralDisplayName="Students" cmcedm:DisplayName="Student" cmcedm:Browsable="true" Name="Student">

-<Key>
<PropertyRef Name="Id"/>
</Key>

<Property cmcedm:DisplayName="Id" cmcedm:Browsable="true" Name="Id" cmcedm:Editable="true" Nullable="false" Type="Edm.Int32" cmcedm:Required="true"/>

<Property cmcedm:Description="Campus" cmcedm:DisplayName="Campus" cmcedm:Browsable="false" Name="CampusId" cmcedm:Editable="true" Nullable="false" Type="Edm.Int32" cmcedm:Required="true" cmcedm:Lookup="true" cmcedm:LookupQueryValueColumn="Id" cmcedm:LookupQueryName="Campuses?$select=Code,Name,Id&$filter=IsActive eq true&$orderby=Name"/>

<Property cmcedm:Description="Current Employer" cmcedm:DisplayName="Current Employer" cmcedm:Browsable="false" Name="EmployerId" cmcedm:Editable="true" Type="Edm.Int32" cmcedm:Lookup="true" cmcedm:LookupQueryValueColumn="Id" cmcedm:LookupQueryName="Employers?$select=Code,Name,Id&$filter=IsActive eq true&$orderby=Name"/>

<Property cmcedm:Description="Employment Status" cmcedm:DisplayName="Employment Status" cmcedm:Browsable="false" Name="EmploymentStatusId" cmcedm:Editable="true" Type="Edm.Int32" cmcedm:Lookup="true" cmcedm:LookupQueryValueColumn="Id" cmcedm:LookupQueryName="EmploymentStatuses?$select=Code,Name,Id&$filter=IsActive eq true&$orderby=Name"/>

Execute a Query

You can copy a query from the metadata, append the query to the Student Base URL, and view the query results in a browser.

1. Access the **Student Base URL** in a browser. In our example, the Student Base URL is as follows:

   http://cltqafb3.campusmgmt.com:8080/Cmc.Nexus.Web/ds/campusnexus/$metadata

2. Remove `$metadata` from the URL

3. **Copy a query** from the metadata to the clipboard. In our example, the query is as follows:

   PreviousEducationCodes?$select=Code,Name,Id&$filter=IsActive eq true&$orderby=Name

   This query selects the Code, Name, and ID from the PreviousEducationCode field, filters by Active, and sorts by Name.

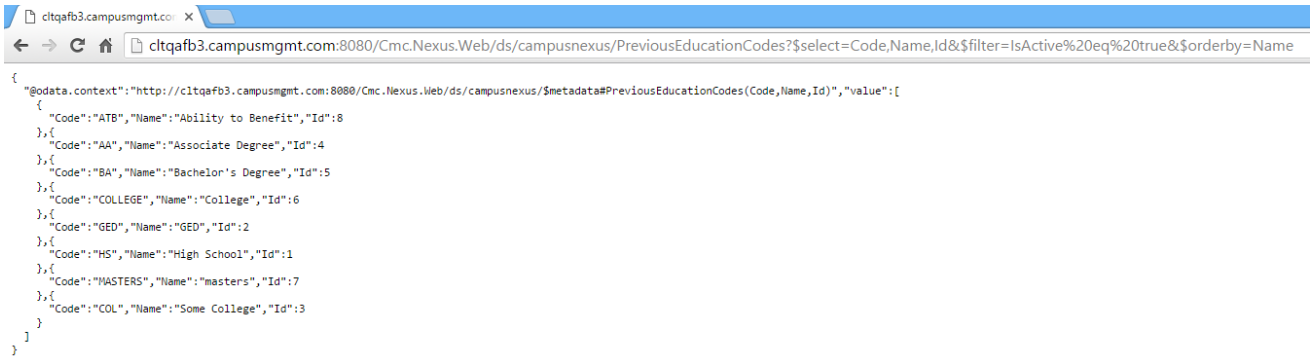
4. **Append the query** to the Student Base URL.

   p>http://cltqafb3.campusmgmt.com:8080/Cmc.Nexus.Web/ds/campusnexus/PreviousEducationCodes?$select=Code,Name,Id&$filter=IsActive eq true&$orderby=Name

5. Press **Enter** to run the query in the browser.

   Use Chrome or Firefox to run the query. Internet Explorer will download the query.

   The browser displays the query results. In our example, the list of active Previous Education Codes is displayed.



Modify a Query

You can modify the query in the URL of the browser to obtain the desired results.

Change the Sort Order

For example, you can change sort order to sort by `Id` instead of `Name`.



Remove the "select" Option

To retrieve the data in all columns of a field, you could remove the "select" option for the columns.

For example, change the query from:

p>PreviousEducationCodes?$select=Code,Name,Id&$filter=IsActive eq true&$orderby=Name

to:

PreviousEducationCodes?$filter=IsActive eq true&$orderby=Name

Use the "$expand" Option for Navigation Properties

For entities that have navigation properties, you can build a query that joins a query for navigation property to the query for the entity.

For example, the Campus entity contains the Buildings navigation property. Navigation properties are related resources for an entity. Use the "$expand" option to expand related resources in line with the retrieved resources.

The following query retrieves all buildings for every campus.

Campuses?$filter=IsActive eq true&$orderby=Id&$expand=Buildings

The basic pattern is as follows: `Entities?$option1= …&$option2=…`

- Use `?` after Entities
- Specify query options such as `filter`, `expand`, and `orderby`. Each option should be prefixed with `$`.
- Combine attributes using `&`.

Change the "$filter" Option

To retrieve the buildings for one campus only, you could filter by Active campuses and by Campus Id as follows:

Campus?filter=IsActive eq true and ID eq 5 orderby=Id $expand=Buildings

Build a Cascading Query Using AngularJS

Forms Builder supports two-way binding using AngularJS expressions. Angular expressions can be part of an OData queries adding flexibility and scope.

Example:

The following query retrieves Buildings in Active status for Campus Id=5.

```
Buildings?filter=IsActive eq true and CampusId eq 5
```

The CampusId is represented by the Model property in the Property Settings. The Model value is `vm.models.studentEntity.CampusId`. The Model value can be added to the LookupQuery using AngularJS syntax (enclosed in double curly braces).

```
Buildings?filter=IsActive eq true and CampusId eq {{vm.models.studentEntity.CampusId}}
```

The resulting query is a cascading filter that retrieves Buildings in Active status for *any* campus that is represented by the Model value.

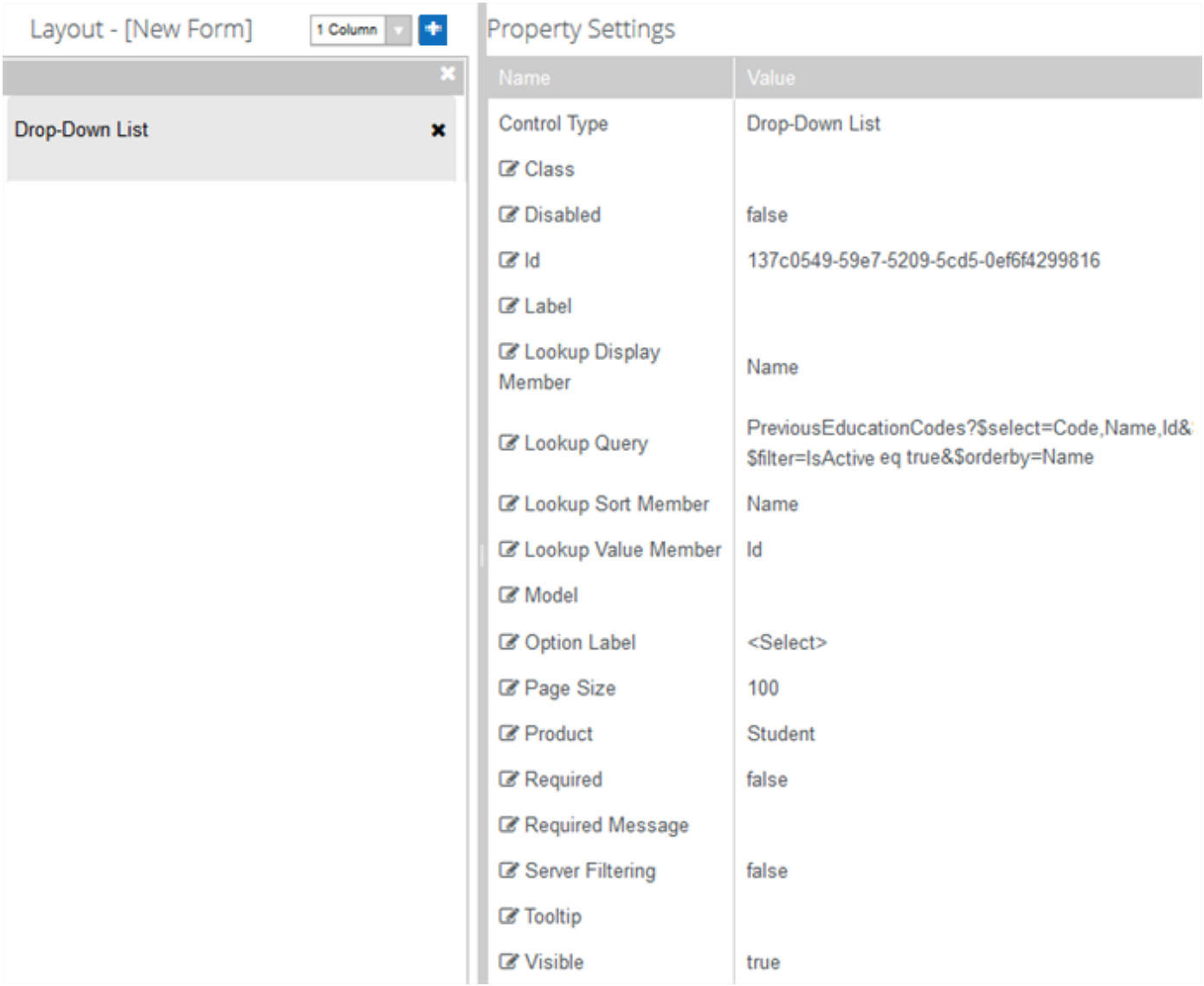
This query is added to a drop-down control for Buildings. The control is added to a form that also contains a drop-down control for Campuses. On the rendered form, the user can select a Campus, and based on the selected Campus, the drop-down for Buildings will list only buildings associated with the selected campus.

OData Syntax Reference

To learn more about the OData query syntax, refer to http://www.odata.org/ and look for “Basic Tutorial” and “Advanced Tutorial”.

Populate the Lookup Query in Form Designer

After testing the OData query in the browser, paste it in the Lookup Query field of the list control in the Property Settings pane of Form Designer.



Notes

- Depending on the columns retrieved by the query, adjust the Lookup Display Member, Lookup Sort Member, and Lookup Value Member properties. The default values "Name" and "Id" are not applicable if the query does not retrieve "Name" and "Id" values.
- The Product specified in the Property Settings pane is the query provider. When you are customizing a Lookup Query, make sure that the Product value (e.g., "Student") matches the database that is being queried.

Lookup Queries for CampusNexus CRM Metadata

For any drop-down or search controls that will be populated via a lookup query, the CampusNexus CRM user needs to enter values for the **Lookup Display Member** and **Lookup Sort Member** attributes. The **Lookup Query** and **Lookup Value Member** property settings should have default values (if applicable for the selected property) as these are currently specified in the metadata.

Run Queries in Web Client for CampusNexus CRM

System integrators can view the results of a lookup query that is available in the Web Client for CampusNexus CRM. Prior to integrating with CampusNexus CRM, this functionality helps an integrator to verify the list of values that will be displayed in their query.

1. Suffix the Web Client URL as follows: **http://<web client url>/nexuscrmodata/$metadata**.

   The webpage that is displayed includes lookup queries that are available by default.

2. Search for the text **"lookup"** and then navigate to the query that you want to run.

   Example

   You want to run the following query to verify the list of available Account types:

   LookupQueryName="EnumAccountAccountTypes?$select=Id,DisplayValue&$filter=IsActive eq 1&$orderby=DisplayOrder"

   a. Copy the following text from the query:

      EnumAccountAccountTypes?$select=Id,DisplayValue&$filter=IsActive eq 1&$orderby=DisplayOrder

   b. Append the copied text to the Web Client URL as follows:

      http://<Web Client URL>/nexuscrmodata/EnumAccountAccountTypes?$select=Id,DisplayValue&$filter=IsActive%20eq%201&$orderby=DisplayOrder

   c. Press ENTER.

3. The list of values available in the Account Type property is displayed.

Build Queries for Occupation Insight

Occupation Insight is a multi-tenant Software as a Service (SaaS) solution. Forms Builder 3.4 and later can be configured to access Occupation Insight as a data source for real-time analytics about the job market. With this configuration, OData queries can be used in Form Designer to retrieve data from the Occupation Insight API.

The configuration for Occupation Insight is done via a name key in the <products> section of the Designer web.-config file. The baseUrl defines the path for Occupation Insight OData queries.

<add name="Occupation Insight" enabled="true" baseUrl="<Your Occupation Insight API URL>" odataPath-h="/odata" requestHeaderKey="apiKey" requestHeaderValue="[]"/>

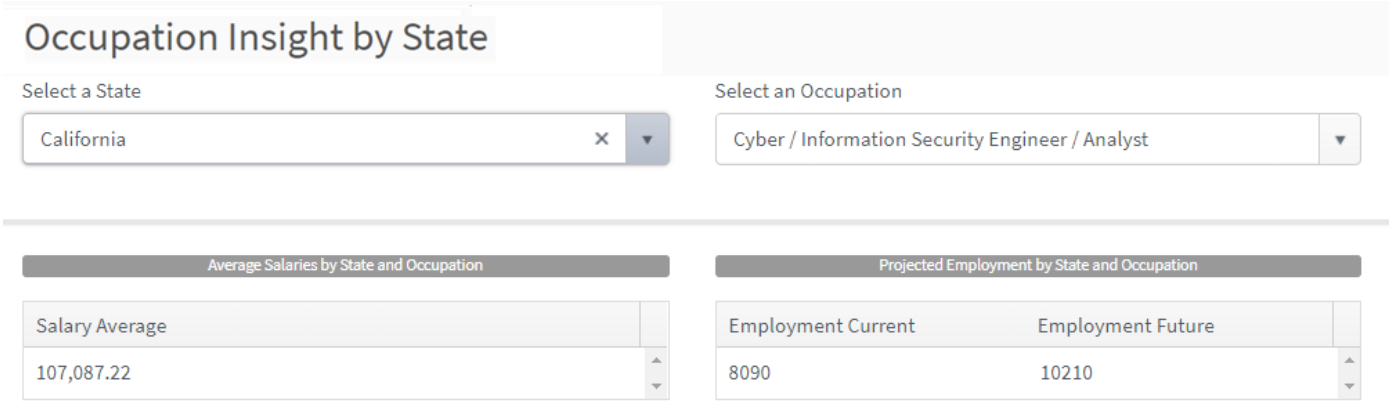
The base URL for your installation of Occupation Insight is displayed in the About window of Forms Builder.

The data returned by the OData queries can be displayed in controls such as Drop-down List, Multiselect, Single-select Search, Typeahead, and Grid. In these controls, users can select "Occupation Insight" as the Product property and construct the applicable OData query to retrieve the desired properties from the Occupation Insight data source.

The Occupation Insight data model is documented in the Anthology Inc. Occupation Insight Power BI API at http://cmc-occupation-insights-bi-api.azurewebsites.net/swagger/ui/index#/.

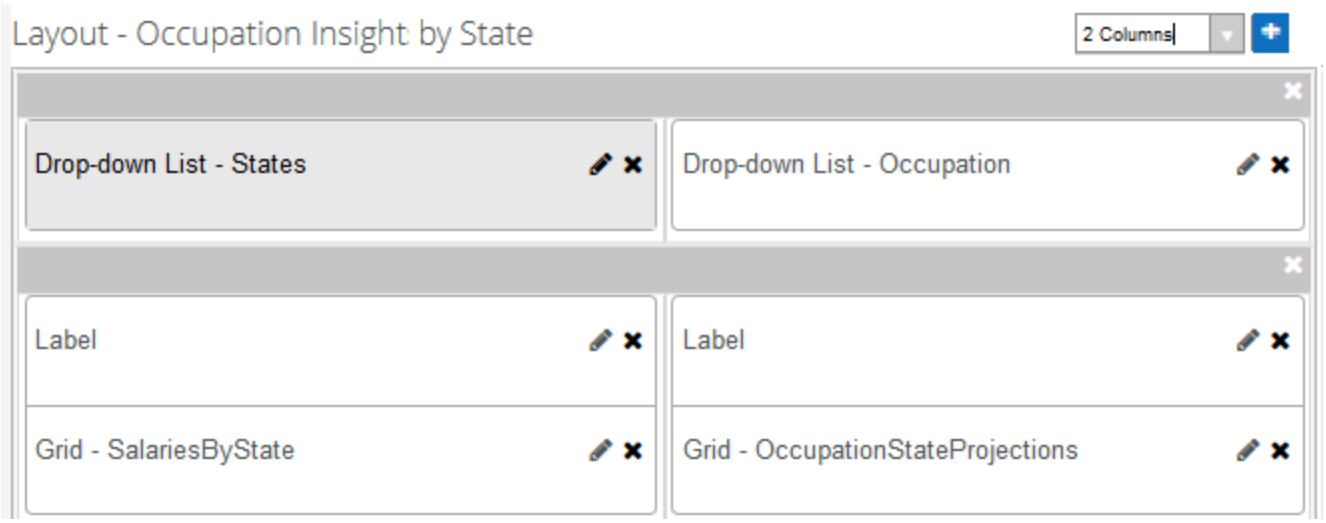
Rendered Form

This form allows the user to view average salaries and projected employment data based on the selected state and occupation.



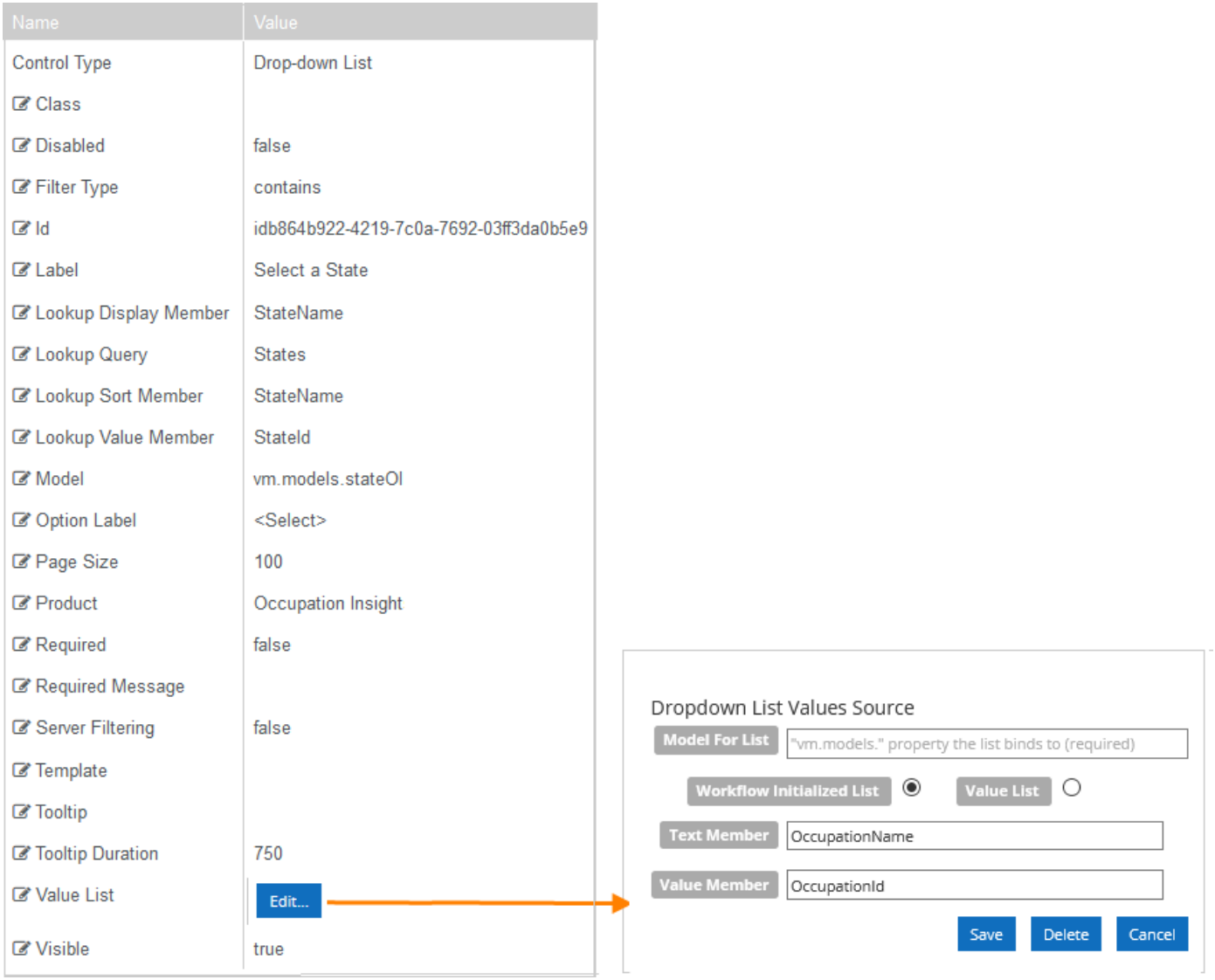
Form Layout

The form layout in Form Designer has two columns with Drop-down List, Label, and Grid controls.



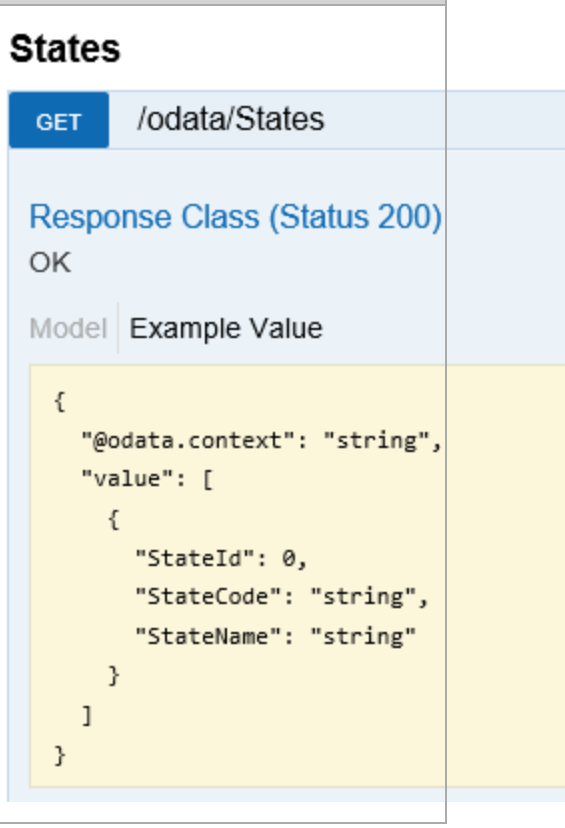
Drop-down List - States

The properties of the Drop-down List for States include the Lookup Query settings and the Product selection.



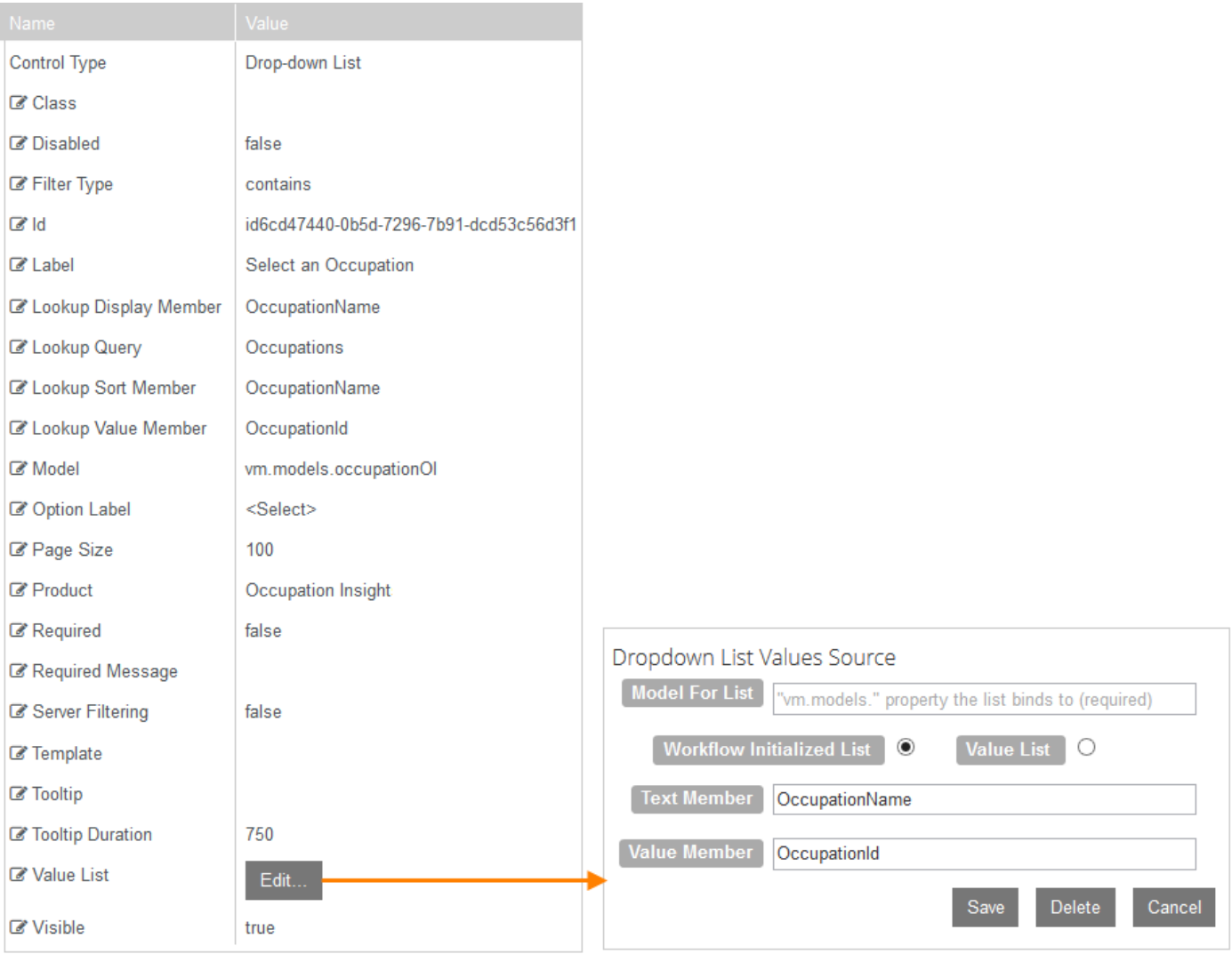
The table below highlights the control properties that are related to the Occupation Insight data source.

Drop-down List - States

Control Property	Property Value	Occupation Insight API
Lookup Display Member	StateName	
Lookup Query	States	
Lookup Sort Member	StateName	
Lookup Value Member	StateId	
Model	vm.models.stateOI	
Product	Occupation Insight	

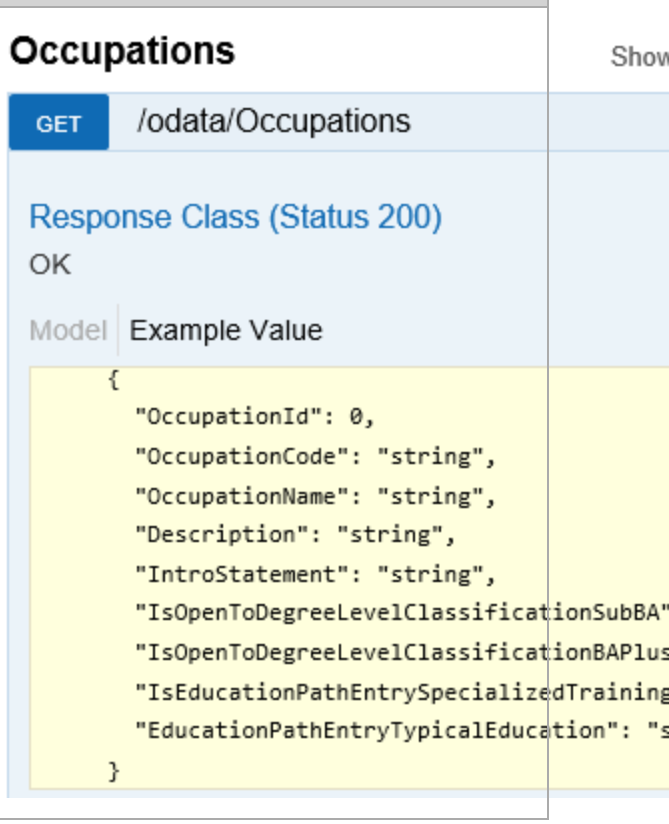
Drop-down List - Occupations

The properties of the Drop-down List for Occupations include the Lookup Query settings and the Product selection.



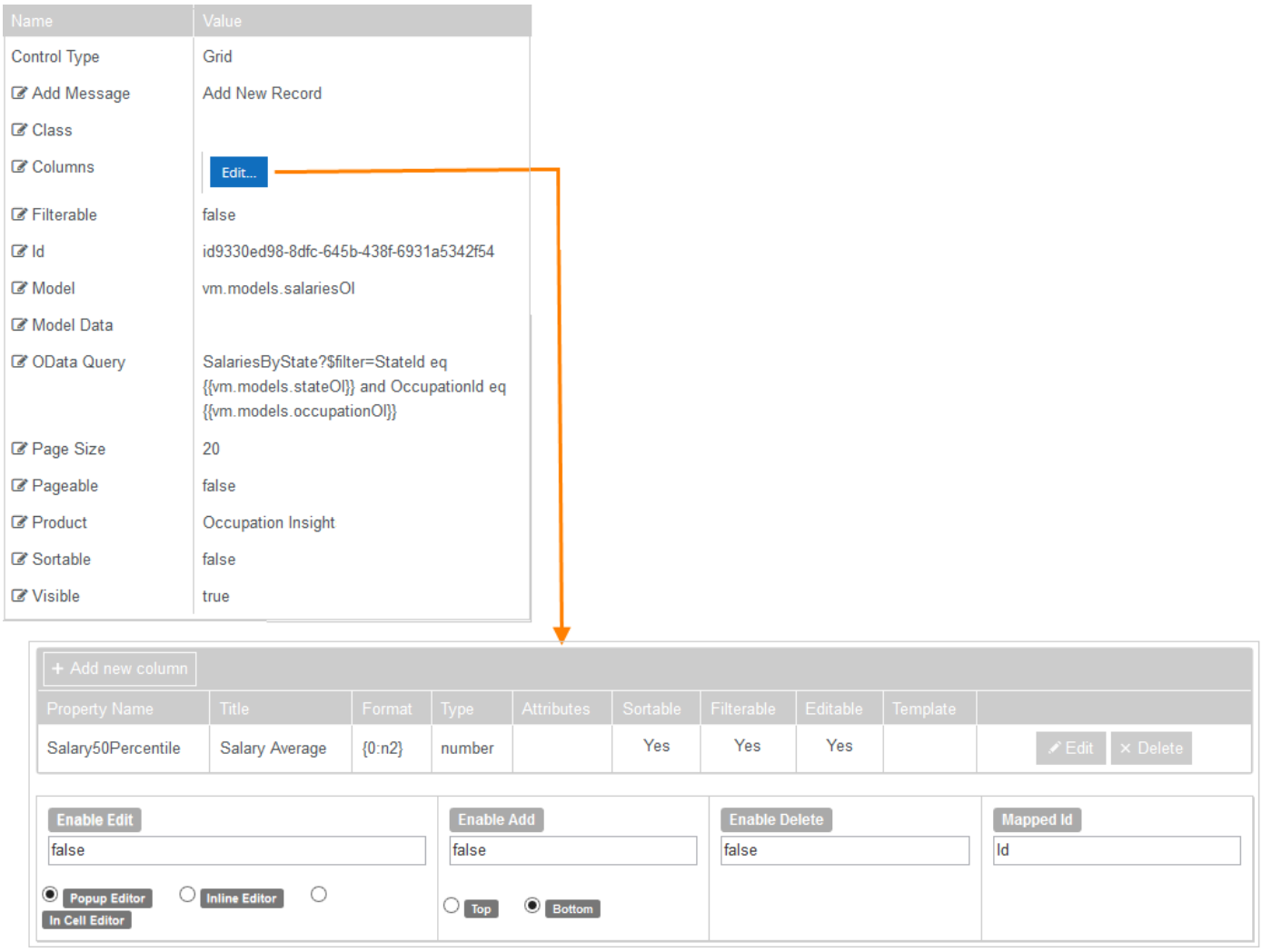
The table below highlights the control properties that are related to the Occupation Insight data source.

Drop-down List - Occupations

Control Property	Property Value	Data Source Occupation Insight API
Lookup Display Member	OccupationName	
Lookup Query	Occupations	
Lookup Sort Member	OccupationName	
Lookup Value Member	OccupationId	
Model	vm.models.occupationOI	
Product	Occupation Insight	

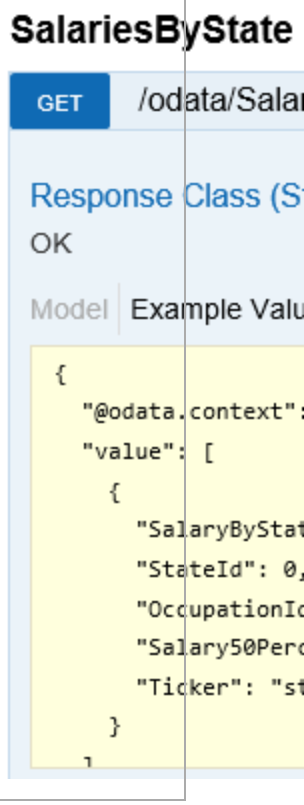
Grid - SalariesByState

The grid for SalariesByState consists of one column that displays the Salary50Percentile value retrieved using the Occupation Insight API.



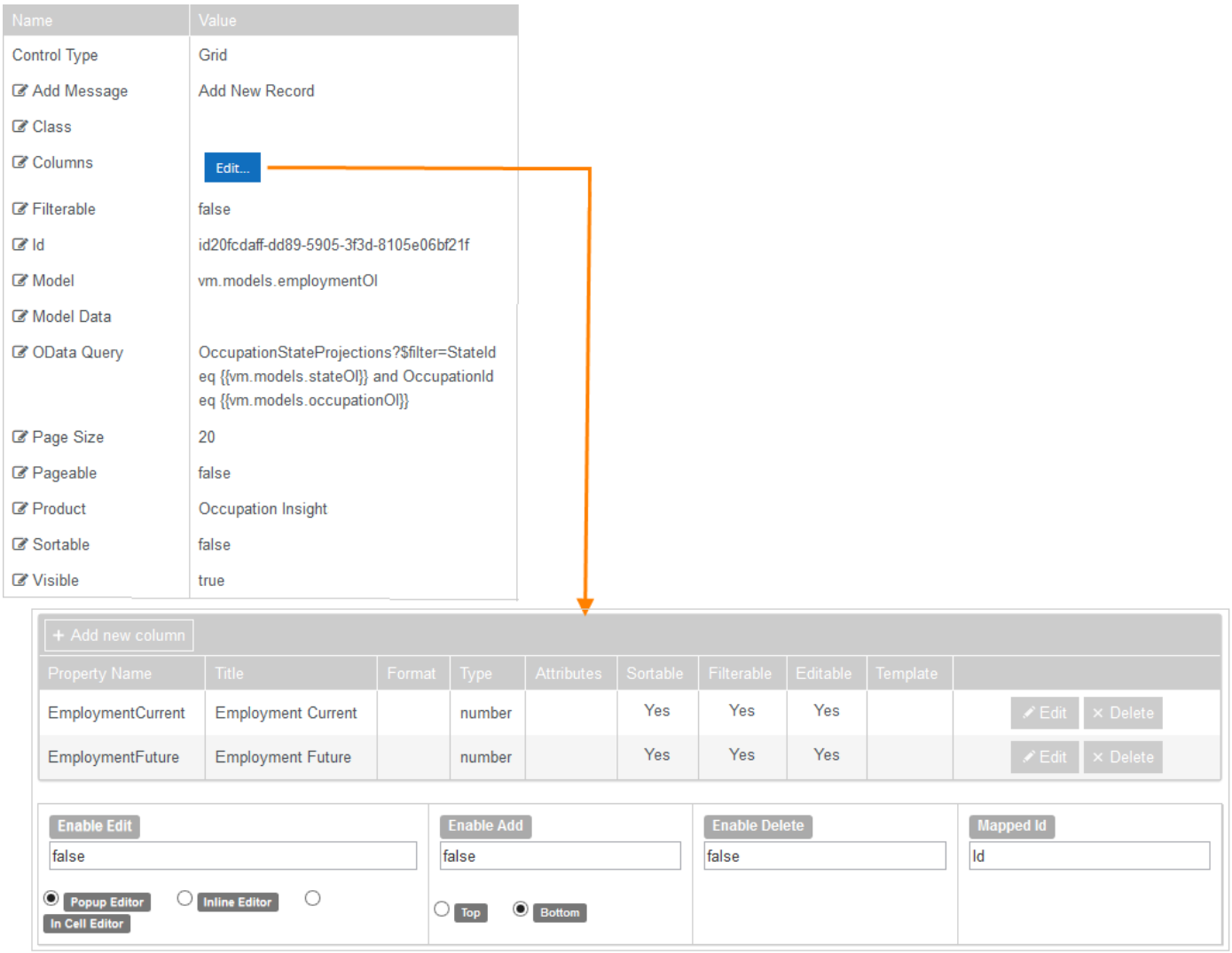
The table below highlights the control properties that are related to the Occupation Insight data source.

Grid - SalariesByState

Control Property	Property Value	Occupation Insight API
Model	vm.models.salariesOI	
OData Query	SalariesByState?$filter=StateId eq {{vm.models.stateOI}} and OccupationId eq {{vm.-models.occupationOI}} This query is linked to the selection of the StateId and OccupationId in the Drop-down Lists above. The Model property binds the Drop-down Lists to the following values: • vm.models.stateOI • vm.models.occupationOI	
Product	Occupation Insight	

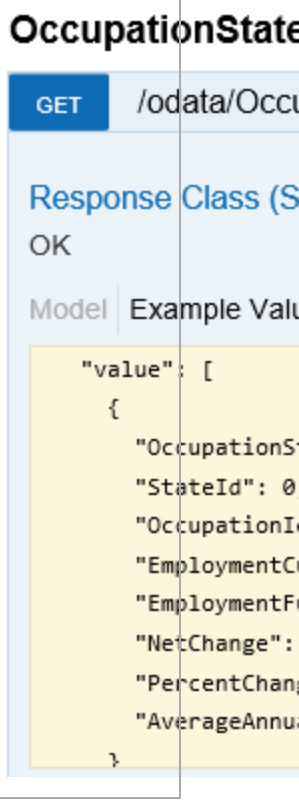
Grid - OccupationStateProjections

The grid for OccupationStateProjections consists of two columns that display the EmploymentCurrent and EmploymentFuture values retrieved using the Occupation Insight API.



The table below highlights the control properties that are related to the Occupation Insight data source.

Grid - OccupationStateProjections

Control Property	Property Value	Occupation Insight API
Model	vm.models.employmentOI	
OData Query	OccupationStateProjections?$filter=StateId eq {{vm.models.stateOI}} and OccupationId eq {{vm.models.occupationOI}} This query is linked to the selection of the StateId and OccupationId in the Drop-down Lists above. The Model property binds the Drop-down Lists to the following values: • vm.models.stateOI • vm.models.occupationOI	
Product	Occupation Insight	