

Perceptive Content Connector

Installation Guide

Version: 1.2.x

Date: Monday, August 22, 2016

© 2016 Lexmark. All rights reserved.

Lexmark is a trademark of Lexmark International Inc., registered in the U.S. and/or other countries. All other trademarks are the property of their respective owners. No part of this publication may be reproduced, stored, or transmitted in any form without the prior written permission of Lexmark.

Table of Contents

Overview	5
Prerequisites	5
Licenses	6
Overview of the setup process	6
Install Content Connector	6
Configure the Content Connector	6
Verify the Content Connector connection	7
Recovering Integration Server Connection	7
Configure Content to use the connector	7
Define the Content Envoy service	8
Configure the Content Connect service	8
Create a Connect Runtime user	9
Configure Content workflow	9
<i>Create the workflow process</i>	9
<i>Create queues in Content</i>	9
Create the success and failure queues	10
Create the Integration ASQ or Connect ASQ	10
Creating an Integration ASQ	10
Creating a Connect ASQ	10
Create an ASQ channel	11
Modify a disabled channel	12
Method 1	12
Method 2	12
Modify an enabled channel	12
Appendix A: Triggers	13
ASQ trigger	13
Appendix B: Actions	14
RouteImageNowWorkflowItem action	14

Appendix C: Readers	16
Object Property reader	16
Form Data Definition reader	18
Form reader	19
Document Pages reader	19
Workflow Item reader	20
Object Search reader	21
Appendix D: Writers	26
Form writer	26
Document Pages writer	26
Object Property writer	27
Custom Properties in Object Property Writer	28
Notes	29
Document Notes Writer	29
Appendix E: SSL	31
Configure Integration Server	31
Configure Content Connector	31
Configure the Envoy Service	32
<i>Enable SSL in Envoy</i>	32
<i>Create a new Envoy service</i>	32
<i>Use the new Envoy service</i>	32
Appendix E: Endpoint Validation	33

Overview

Perceptive Content Connector, built on Perceptive Connect Runtime, allows you to create custom data channels between Perceptive Content and your business applications. The connector facilitates workflow processes that integrate with applications outside of Perceptive Content, also called ImageNow.

Connect Runtime channels consist of a trigger, an action, and a results phase.

- A **trigger** is an event that initiates the channel. The Content Connector provides the **Integration ASQ Trigger**, which allows you to create channels that are triggered when a document enters an Integration automation system queue (ASQ) in Content.
- An **action** is a task that the channel performs when the trigger event occurs. The Content Connector provides the **RouteImageNowWorkflowItem** action, which allows you to create channels that, when triggered, route a Content document or folder to a specified queue. Other Perceptive Connect Runtime Connectors also add actions that can be used in a channel.
- The **results** phase lets you specify what to do with the data that is available after the action completes. For example, you can output values from a channel to Content document keys or custom properties.

This guide outlines the installation and configuration procedures for Content Connector. It also includes instructions for creating Integration ASQ channels and reference appendices for help configuring channels that use the connector.

This connector also provides a `PCR Trust Validator`. A `Trust Validator` can be used to ensure that only authenticated users can access sensitive information through REST/SOAP endpoints. Another connector that exposes a data lookup endpoint through PCR can optionally depend on the `com.perceptivesoftware.imagenow.service.validator.ImageNowValidatorRestValidationFilter` name. See the Perceptive Connect Runtime developer guide for more information about `Trust Validators`.

For more information about Connect Runtime, refer to the [Perceptive Connect Runtime Installation and Setup Guide](#).

Prerequisites

To use the Content Connector, you must have access to working installations of the following products.

- Perceptive Connect Runtime 1.3
- ImageNow Client and Server, version 6.7 or 6.8 or Perceptive Content Client and Server, version 7.0 or 7.1
- Perceptive Integration Server, version 6.7, 6.8, 7.0, or 7.1

Licenses

The following licenses are required for Content Connector. * Integration Framework Transaction Pack *
Integration Server 6.7, 6.8, 7.0, or 7.1

Overview of the setup process

To install and configure Content Connector, complete the following sections in order.

1. [Install Content Connector](#)
2. [Configure the Content Connector](#)
3. [Configure Content to use the connector](#)
4. [Create an Integration ASQ channel](#)

Install Content Connector

Complete the following steps.

1. Go to the Lexmark website at www.lexmark.com and log in to the Customer Portal.
2. In the **Product Downloads** page, download the **Perceptive Content Connector for Windows** ZIP file.
3. Extract the file to a temporary directory.
4. To install the Content Connector iScript Extension, navigate to **[drive:]\{Content Connector directory}** and copy **PerceptiveConnectExtensions.js** to the **\script** directory of your Content Server. For example, **[drive:]\inserver6\script**.
5. In a browser, go to the **Connect Runtime Dashboard** at `http://{Connect Runtime host name}:{port}`.
6. In the browser dialog box, enter the Connect Runtime user name and password. **Note** The default user name and password are `admin`. However, the administrator can change the defaults during the Connect Runtime installation process.
7. Click **Install a Connector**.
8. On the **Bundle Management** page, drag the content connector zip file over the **DRAG FILES HERE** area of the page and then drop it.
9. When the installation completes, press **Accept** to accept the installation or **Roll back** to undo the installation.

Note For more information on installing Connectors please see the `Install connectors` section of the Perceptive Connect Runtime install guide.

Configure the Content Connector

To configure the Content Connector, complete the following steps.

1. In the **Connect Runtime Web Console**, click **Perceptive Connect > View Configuration**.
2. In the **Name** column, under **Perceptive Content Connector**, click **Connection Manager**.
3. In the **Connection Provider Target** list, select your desired connection provider.
 - Supported providers:
 - Integration Server 6.7
 - Integration Server 6.8
 - Integration Server 7.0
 - Integration Server 7.1
4. Connect Runtime requires a Content user name and password to access Content Server. In the **User name** and **Password** boxes, specify a user name and password that is valid with your network authentication method. **Note** You add this user to Content Server later in the installation process in [Create a Connect Runtime user](#).
5. Content Connector requires a connection to Perceptive Content Integration Server. On the **View Configuration** screen, click the line for the connection you selected in **Step 3**.
6. In the Integration Server URL, enter a valid URL. For example, `http://imagenow:8080/integrationserver`.
7. Click **Save**.

Verify the Content Connector connection

To verify that Content Connector has connected to Integration Server, complete the following steps.

1. In the **Connect Runtime Web Console**, click **Main > Components**.
2. Verify that "ImageNow Endpoint Manager" is listed as "active."

Note If a ImageNow Endpoint Manager is not listed as "active", check the log file for errors

Recovering Integration Server Connection

Content Connector maintains a "heartbeat" connection to the configured Integration Server. Periodically, Content Connector verifies that its connection to Integration Server is still operating correctly. If there is a problem verifying the connection to Integration Server, Content Connector will automatically disable any components, and therefore any channels, which depend on Integration Server. When the connection to Integration Server has be re-established, those components and channels will become active again.

Configure Content to use the connector

In this section, you define the Content Envoy service, create a user that Connect Runtime uses to access the Content system, and configure a Content workflow for use with Connect Runtime.

Define the Content Envoy service

Connect Runtime requires one Envoy service per Connect Runtime instance.

To define a Content Envoy service, complete the following steps. To configure multiple Envoy services, repeat this process and name each service uniquely.

1. In **Management Console**, in the left pane, click **Envoy Services** and then click **New**.
2. In the **Envoy Services** dialog box, in the **Definition** page, set the following attributes.
 - In the **Name** box, type `Perceptive Connect`.
 - In the **URI** box, type the URI for your Integration Server in the format `http://{Connect Runtime host name}:{port}/ws/workflowTrigger?wsdl`.
 - In the **Authentication** list, select **None**.
 - Optional. To enable interceptor logging for the remote service, select the **Enable interceptor logging** check box. The logging files produced by the Interceptor are useful for troubleshooting issues with SOAP requests and responses.
3. Click **Next**.
4. In the **Operations** page, under the **WorkflowTriggerService** operation, select the check box for **InvokeTrigger** and click **Finish**.

Configure the Content Connect service

As of Perceptive Content 7.1.5, a new queue type called Connect ASQ is available, which can be configured to use a single instance of the Connect Runtime.

To configure the Content Connect service, complete the following steps.

1. Navigate to the location where the Content Server is installed and enter the `{install location}\etc` directory.
2. Open the `inserverWorkflow.ini` file.
 - Configure the `connect.uri` setting and set it to your Connect Runtime instance with this format `http://{Connect Runtime host name}:{port}/rs/workflowTrigger`.
 - Configure the `connect.timeout` to your desired expiration time.
3. Save the file.
4. Optional. The setting will be automatically loaded after a short period. To reload the configuration immediately, you can restart the Content Server service.

Create a Connect Runtime user

Connect Runtime requires a unique user account on Content Server. This manager account is only for Connect Runtime; you do not use the Connect Runtime user as a login account.

To create a Connect Runtime user, complete the following steps.

1. On your operating system or LDAP server, create a unique user account for Connect Runtime.
2. Using the Content owner or administrator account, start **ImageNow Client** and open **Management Console**.
3. In the left pane, click **Users**.
4. In the right pane, on the **User Profiles** tab, click **New** and type a user name, such as `Perceptive Connect Runtime`.
5. To promote the user to Manager, on the **Security** tab, click the Connect Runtime user and click **Promote**.
6. In the confirmation dialog box, click **Yes**.

Configure Content workflow

To configure Content workflow, complete the following procedures in order.

[Create the workflow process](#)

[Create queues in Content](#)

Create the workflow process

To create a workflow process, complete the following steps.

1. In **Management Console**, in the left pane, click **Workflow** and then click **New**.
2. In the **Add Process** dialog box, in the **Name** box, type a name, such as `Integration WF`.
3. Optional. In the **Description** box, type a description of the process.
4. Click **OK**.

Create queues in Content


Integration ASQs and Connect ASQs send data using web service notifications to facilitate enhanced business processes between Content and other applications. When a document enters the ASQ, Content sends a web service notification to the application URI using the specified Envoy service (for Integration ASQs) or the configured Connect URL (for Connect ASQs).

To enable integration between Content and Connect Runtime, you create three queues: a work queue for successful processing, a work queue for failures, and either an Integration ASQ or Connect ASQ for incoming files.

Open the workflow process you created in the previous section and complete the following procedures.

Create the success and failure queues

To create the success and failure queues, complete the following steps.


1. In **ImageNow Workflow Designer**, under **Queues**, drag two **Work**  queues to the process diagram.
2. To create the success queue, double-click a **Work** queue to open the **Queue Properties** dialog box. In the **Name** box, type `Success` and then click **OK**.
3. To create the failure queue, repeat the previous step with the other **Work** queue and name it `Failure`.

Create the Integration ASQ or Connect ASQ

The Integration ASQ and Connect ASQ are functionally the same. The Connect ASQ was added in Perceptive Content 7.1.5.


Creating an Integration ASQ

To create an Integration ASQ, complete the following steps.

1. Under **Queues**, drag an **Integration**  queue to the process diagram.
2. Double-click the Integration queue to open the **Queue Properties** dialog box. In the **Name** box, type a name for the queue, such as `Begin Processing`.
3. Under **Automated Action**, set the following attributes.
 - Under **Success Action**, in the **Queue** list, select the name of the success queue.
 - Under **Failure Action**, in the **Queue** list, select the name of the failure queue.
 - In the **Route After (Days)** box, type the number of days that items should remain in the **Failed** queue after the business application receives a successful call for those items. The maximum number of days is 365. By default, Content routes items to the failure queue after one day.
 - Under **Envoy Service**, in the **Service Operation Name** list, select **Perceptive Connect::InvokeTrigger**. **Note** Record the queue ID value to use in the [Create an ASQ channel](#) section.
4. Click **OK**.

Creating a Connect ASQ

To create an Connect ASQ, complete the following steps.

1. Under **Queues**, drag a **Connect**  queue to the process diagram.
2. Double-click the Integration queue to open the **Queue Properties** dialog box. In the **Name** box, type a name for the queue, such as `Begin Processing`.
3. Under **Automated Action**, set the following attributes.
 - Under **Success Action**, in the **Queue** list, select the name of the success queue.
 - Under **Failure Action**, in the **Queue** list, select the name of the failure queue.
 - In the **Route After (Days)** box, type the number of days that items should remain in the **Failed** queue after the business application receives a successful call for those items. The maximum number of days is 365. By default, Content routes items to the failure queue after one day. **Note** Record the queue **ID** value to use in the [Create an ASQ channel](#) section.
4. Click **OK**.

Create an ASQ channel

A channel that uses the Integration ASQ Trigger listens for incoming web service notifications from a specified Integration ASQ or Connect ASQ. To create a channel using this trigger, complete the following steps.

Notes

- You must have a valid ImageNow or Perceptive Content connection to create an ASQ channel.
 - You must create an Integration ASQ or Connect ASQ in Content before mapping a Connect Runtime channel to it. To do so, complete the procedures outlined in [Configure Content workflow](#). When you create the ASQ, record the queue ID, which is how you link the channel to the ASQ.
1. In a browser, go to the **Connect Runtime Dashboard** at `http://{Connect Runtime host name}:{port}` and click **Create a channel**.
 2. In the **Select a trigger** list, select the **Integration ASQ Trigger**.
 3. In the **Workflow Queue ID** box, type the ID for the Integration ASQ you want to trigger the channel and then click **Next**. **Notes** The **Workflow Queue ID** must be a valid ImageNow Queue ID. If a timeout error occurs during authentication, check the log file for errors.
 4. In the **Select an action** list, select an action. **Note** If you do not want the channel to perform an action, select the **No Action** option.
 5. Update the input data mapping XML for the action in the **Configure input mapping** box that appears and then click **Next**. For more information about configuring the input mapping, refer to the appendices. Connect Runtime performs validation checks on both the input and output data mapping XML. Any errors in the XML display in the **Validation results** message under the text box.

6. In the **Configure output mapping** box, update the output data mapping XML and then click **Save Channel**. For more information about configuring the output mapping, refer to [Appendix D: Writers](#).
7. Click **OK** in the pop-up window if you want to enable the channel. Click **Cancel** to save the channel without enabling it.

After you enable a channel, you cannot update it from the Connect Runtime Dashboard.

Modify a disabled channel

If you save a channel but do not enable it, you can edit it from the Connect Runtime Dashboard. To modify an existing disabled channel, complete the following steps.

Method 1

1. Navigate to the **Connect Runtime Dashboard** at `http://{Connect Runtime host name}:{port}` and click **View all channels**.
2. Click on the ID of the channel that you wish to modify.
3. Modify the channel by following the steps outlined in the previous section.

Method 2

1. Navigate to the **Connect Runtime Dashboard** at `http://{Connect Runtime host name}:{port}` and click **Create a channel**.
2. In the **Workflow Queue ID** box, type the ID of the queue you previously configured and click **Next**.
3. In the message that displays indicating the trigger is already in use, click **Click to view the channel**.
4. Modify the channel by following the steps outlined in the previous section.

Modify an enabled channel

You cannot modify any channel in the Connect Runtime until it has been disabled. If you need to change an enabled channel, follow the instructions in the **Connect Runtime Install Guide** to disable the channel. Once disabled, you can modify the channel by clicking the channel's row in the list, then clicking the **View mapping** button at the top of the page.

Appendix A: Triggers

A trigger is an event that causes a channel to execute. Each channel has only one trigger, so the event defined by the trigger and its inputs is the only entry point into a given channel. Triggers can also bring data into the channel that is available for input mapping for the channel action or the results output.

The Content Connector provides the following trigger and associated data.

ASQ trigger

The ASQ Trigger allows you to create channels that are triggered when a document enters an Integration ASQ or Connect ASQ in Perceptive Content. If the channel execution is successful, the document is routed to a success queue specified in Content. If the execution is unsuccessful, the document is routed to a failure queue.

The only input this trigger requires is the ASQ ID. The trigger includes several output values that you can use in the channel data context. These values can be consumed using the Trigger reader, provided by Connect Runtime. The following data fields are available.

- WorkflowItemId
- Workflow Queue Id
- QueueName
- SuccessQueueId
- SuccessQueueName
- FailureQueueId
- FailureQueueName You can reference these values in data context configuration, as shown in the following example.

```
<c:trigger>WorkflowItemId<c:trigger>
```

Appendix B: Actions

An action is a connector-defined task configured in the channel that executes when the channel is triggered. The various inputs required for the action are configured in an XML document that you edit during channel creation. The action uses these inputs to complete its task in an application outside of Connect Runtime, as defined in the connector.

The Content Connector provides the following action.

RoutelImageNowWorkflowItem action

The RoutelImageNowWorkflowItem action lets you route a workflow document or folder in Content to specified success or failure queues, depending on the outcome of the channel execution.

When you select the RoutelImageNowWorkflowItem action, Connect Runtime automatically populates the following XML configuration template in the **Configure input mapping** box.

```
<c:parameter>
  <c:name>WorkflowItemId</c:name>
  <c:none/>
</c:parameter>
<c:parameter>
  <c:name>SuccessQueueName</c:name>
  <c:none/>
</c:parameter>
<c:parameter>
  <c:name>FailureQueueName</c:name>
  <c:none/>
</c:parameter>
```

The action requires the three parameter blocks shown above. The `WorkflowItemId` field is the ID of the workflow item that the action routes. This value can be retrieved from the Trigger or Workflow Item readers. The `SuccessQueueName` and `FailureQueueName` fields contain the names of the success and failure queues that the item is routed to depending on the outcome of the channel execution.

This action may be used with a connector-provided trigger to route an object in Content. The trigger may output a parameter named "DocumentId" that can be referenced by the action's parameter.

Example

```
<c:parameter>
  <c:name>DocId</c:name>
  <c:trigger>DocumentId</c:trigger>
</c:parameter>
<c:parameter>
  <c:name>WorkflowItemId</c:name>
  <in:workflowItem>
```

```
        <in:reference>DocId</in:reference>
        <in:objectType>DOCUMENT</in:objectType>
        <in:objectField>WORKFLOW_ID</in:objectField>
    </in:workflowItem>
</c:parameter>
<c:parameter>
    <c:name>SuccessQueueName</c:name>
    <c:literal>Success</c:literal>
</c:parameter>
<c:parameter>
    <c:name>FailureQueueName</c:name>
    <c:literal>Failure</c:literal>
</c:parameter>
```

In the example, the channel uses `DocumentId`, provided by the trigger, to retrieve the `WORKFLOW_ID`. The configuration also includes parameters that reference the success and failure queues that the document is routed to; the `literal` elements `Success` and `Failure` are the names of the respective queues.

Appendix C: Readers

Readers are components, provided by connectors, that let you configure channels to retrieve data from other applications for use in the channel's data context during its execution. You use readers to configure the action input mapping, which allows the channel to have the required data to execute the action. You can invoke readers using specific XML tags, defined per reader.

Note The channel data context refers to the data from various sources that is available in the channel for action input and results output configuration. The data available in the context depends on the trigger the channel uses as well as the connectors that are installed in Connect Runtime.

The Content Connector provides the following readers.

Object Property reader

The Object Property reader lets you read Content document keys, document properties, and custom properties in a channel. Using the reader, you can configure the channel input to map object properties to the data context for use in the action execution and results output configuration.

To map object properties to a channel, include the following XML template in the channel input mapping, complete with the appropriate values.

```
<in:objectPropertyReader>
  <in:name></in:name>
  <in:objectIdRef></in:objectIdRef>
  <in:objectType></in:objectType>
  <in:propertyType></in:propertyType>
  <in:compositeChildPropertyName></in:compositeChildPropertyName>
</in:objectPropertyReader>
```

The `name` field contains the name of the property to read. The `objectIdRef` field contains a reference to a value in the data context that contains the ID of the object to read. The `objectType` field contains the type of the object being read. The reader can retrieve the following object types.

- DOCUMENT
- FOLDER

The `propertyType` is the type of property that is being read. The reader can retrieve the following property types.

- KEY
- CUSTOMPROPERTY
- DOCPROPERTY

The `compositeChildPropertyName` is the name of the child property of a composite custom property. In order to read a specific child property of a composite property, this field would be set to the child property's name and the composite property's name would be specified in the `name` field with a `propertyType` of `CUSTOMPROPERTY`. However, this field is optional. When `compositeChildPropertyName` is not filled out but a composite property has been specified, all child properties of the composite property are returned in the form of key-value pairs with the key being the custom property name and value being the custom property value.

There are a number of different custom property types in ImageNow. Below is a list of types currently supported by the Object Property reader, and a description of how their values are stored in the Context.

- **STRING** Stored in the context as a `String`.
- **NUMBER** Stored in the context as a `Number` object.
- **FLAG** Stored in the context as a `boolean`.
- **DATE** Stored in the context as a `Date` object.
- **LIST** The currently selected List value is stored in the context as a `String`.
 - **Note** There is currently no way to retrieve the other candidate values for the List type.
- **USER GROUP/USER LIST** The currently selected user is stored in the context as a `String`.
 - **Note** As with the List type, there is currently no way to retrieve the other users in the list or group.
- **COMPOSITE** Stored in the context as a `Map<String, Object>` when `compositeChildPropertyName` is not filled out and there is only one composite property of the same ID in the document properties list.
- **COMPOSITE** Stored in the context as `List<Map<String, Object>>` when `compositeChildPropertyName` is not filled out and there are multiple composite properties with the same ID in the document properties list.
- **COMPOSITE** Stored in the context as a `String, Number, Flag, Date` when `compositeChildPropertyName` is filled out.
- **ARRAY** Stored in the context as a `Map<String, List<Object>>` the `String` is the name of the custom property tied to the array.

Example

```
<c:parameter>
  <c:name>DrawerVal</c:name>
  <in:objectPropertyReader>
    <in:name>DRAWER</in:name>
    <in:objectIdRef>DocId</in:objectIdRef>
    <in:objectType>document</in:objectType>
    <in:propertyType>key</in:propertyType>
    <in:compositeChildPropertyName></in:compositeChildPropertyName>
```

```
    </in:objectPropertyReader>
  </c:parameter>
```

The Reader reads the value of the “DRAWER” key of the docId and stores it in the DrawerVal context item. The **DocId** referenced is provided by some other parameters. Refer to [Workflow Item Reader](#) for more information.

```
<c:parameter>
  <c:name>FirstNameVal</c:name>
  <in:objectPropertyReader>
    <in:name>FullNameComposite</in:name>
    <in:objectIdRef>DocId</in:objectIdRef>
    <in:objectType>document</in:objectType>
    <in:propertyType>customproperty</in:propertyType>

    <in:compositeChildPropertyName>FirstName</in:compositeChildPropertyName>
  </in:objectPropertyReader>
</c:parameter>
```

The Reader reads the value of the “FirstName” child property from custom property “FullNameComposite” of the docId and stores it in the FirstNameVal context item.

Note * The objectPropertyReader reads through any Content document keys, document properties, and custom properties listed in the mapping even if an error in reading any of these types occurs. If such an error occurs, it is logged and a ReaderException is thrown.

Form Data Definition reader

The Form Data Definition reader retrieves a data definition XML document, associated with a form in Content, as a w3c.Document Java type.

To map a form data definition document to a channel, complete with the appropriate values. Include the following XML template in the channel input mapping.

```
<in:eFormDataDefinitionSource></in:eFormDataDefinitionSource>
```

The eFormDataDefinitionSource field contains the name of the form.

Example

```
<c:parameter>
  <c:name>DataDef</c:name>
  <in:eFormDataDefinitionSource>AP Invoice</in:eFormDataDefinitionSource>
</c:parameter>
```

In the example, the reader looks for the data definition associated with the AP Invoice form. If a data definition document exists for the form, it is stored in the DataDef field.

Form reader

The Form reader retrieves all of the data stored in a form for a folder or document in Content and saves it in the channel data context as a `w3c.Document` Java type.

To map form data to a channel, complete with the appropriate values. Include the following XML template in the channel input mapping.

```
<in:eFormSource>
  <in:reference></in:reference>
  <in:name></in:name>
  <in:type></in:type>
</in:eFormSource>
```

The `reference` field contains a reference to the document ID from which you want to read form data. The `name` field specifies the name of the form. The `type` field specifies the Content item type the form is associated with. The only acceptable type of form is `DOCUMENT`.

Example

```
<c:parameter>
  <c:name>FormVal</c:name>
  <in:eFormSource>
    <in:reference>DocId</in:reference>
    <in:name>AP Invoice</in:name>
    <in:type>DOCUMENT</in:type>
  </in:eFormSource>
</c:parameter>
<c:parameter>
  <c:name>DocId</c:name>
  <c:trigger>DocumentId</c:trigger>
</c:parameter>
```

In the example, `DocId` references the trigger `DocumentId` value, which the reader uses to retrieve the document's `AP Invoice` form. The resulting `w3c.Document` type is saved as `FormVal`.

Document Pages reader

The `DocumentPages` reader lets you access the binary page data inside documents in `ImageNow` and insert them into a list of `MappingInputStream` objects.

To map a binary page to a channel, include the following XML template in the channel input mapping, complete with the appropriate values.

```
<in:documentPagesReader>
  <in:docIdRef></in:docIdRef>
  <in:pageNums></in:pageNums>
</in:documentPagesReader>
```

The `docIdRef` field contains a reference to the document ID from which you want to read binary page data. The `pageNums` field specifies the page numbers whose binary data you want to access.

Note Page numbers are indexed starting at "1." You can select pages indexed from the end of the document using negative numbers. For example, "-1" is the final page in the document.

A list of page numbers is delineated by commas, and page ranges are marked with colons. For example, "5:15" will cause the reader to read the binary page data for pages 5 up to and including 15.

Example

```
<c:parameter>
  <c:name>PageList</c:name>
  <in:documentPagesReader>
    <in:docIdRef>DocId</in:docIdRef>
    <in:pageNums>1,2,5:15,22,25:27</in:pageNums>
  </in:documentPagesReader>
</c:parameter>
```

In the example, the reader retrieves the specified `pageNums` associated with the `DocIdRef`.

Usage Example

```
<c:parameter>
  <c:name>PageList</c:name>
  <in:documentPagesReader>
    <in:docIdRef>DocId</in:docIdRef>
    <in:pageNums>1,2,5:15,22,25:27</in:pageNums>
  </in:documentPagesReader>
</c:parameter>
<parameter>
  <name>myPage</name>
  <listItem>
    <listRef>PageList</listRef>
    <index>0</index>
  </listItem>
</parameter>
```

In this example, the [List Item Reader](#) retrieves an item at index "0" from `PageList` and stores it in the context as `myPage`.

Workflow Item reader

The Workflow Item reader lets you input the value of a field attached to a workflow item. Using the reader, you can configure the channel input to map the workflow item field to the data context for use in the action execution and results output configuration.

To map a workflow item field to a channel, include the following XML template in the channel input mapping, complete with the appropriate values.

```

<in:workflowItem>
  <in:reference></in:reference>
  <in:objectType></in:objectType>
  <in:objectField></in:objectField>
</in:workflowItem>

```

The `reference` field contains the name of a reference to the document ID from which you want to read a workflow item value. The `objectType` field specifies the type of the item ID; accepted types include `DOCUMENT` and `WORKFLOW`. The `objectField` element specifies the field to read from the workflow item. The reader can retrieve the following fields.

- `WORKFLOW_ID`
- `OBJECT_ID`
- `OBJECT_TYPE`
- `QUEUE_NAME`
- `QUEUE_ID`

Example

```

<c:parameter>
  <c:name>DocId</c:name>
  <in:workflowItem>
    <in:reference>WFId</in:reference>
    <in:objectType>WORKFLOW</in:objectType>
    <in:objectField>OBJECT_ID</in:objectField>
  </in:workflowItem>
</c:parameter>
<c:parameter>
  <c:name>WFId</c:name>
  <c:trigger>WorkflowItemId</c:trigger>
</c:parameter>

```

In the example, `WFId` references the trigger `WorkflowItemId` value, which the reader uses to retrieve the workflow's `OBJECT_ID` value and store it in the `DocId` field.

Object Search reader

The Object Search reader lets you search for objects in ImageNow using one or more search terms to find the object. The search reader will run a configured view and pull out the configured column from the view. The results of the search is a list of strings representing the column value for each row in the view search result. For example, this reader would allow you to find document ID's based on a list of search terms.

To map an object search reader to a channel, include the following XML template in the channel input mapping, complete with the appropriate values.

```

<in:objectSearch>
  <in:viewType></in:viewType>
  <in:viewName></in:viewName>
  <in:viewColumnID></in:viewColumnID>
  <in:maxWaitTime></in:maxWaitTime>
  <in:searchTerm>
    <in:joinOperation></in:joinOperation>
    <in:operatorType></in:operatorType>
    <in:parameterType></in:parameterType>
    <in:propertyName></in:propertyName>
    <in:searchValueRef></in:searchValueRef>
  </in:searchTerm>
</in:objectSearch>

```

- `viewType` specifies the type of ImageNow view to run.
- `viewName` specifies the name of the view to be run.
- `viewColumnID` specifies the view column of the desired data. For example, the column ID for document ID is 8.
- `maxWaitTime` specifies the maximum time (in seconds) to wait before returning an empty result set. The reader will poll the view at a fixed interval until the wait time elapses. A time of 0 will run the view once. A maximum of 300 seconds (5 minutes) is allowed for a wait time. The default is 0, no wait time.
- `searchTerms` record is repeatable. This record of information is used to build up your search criteria for finding the object.
- `joinOperation` specifies how to join this term with the next `searchTerm`. Field is ignored if no subsequent `searchTerm`.
- `operatorType` specifies the operation type.
- `parameterType` specifies the parameter type.
- `propertyName` specifies which the Key or Custom Property used by the operation.
- `searchValueRef` contains a reference to the search value that will be used in the search query.

The `objectType` can be one of the following values:

- DOCUMENT
- FOLDER

The `joinOperation` can be one of the following values:

- AND
- OR

The `operatorType` can be one of the following values:

- EQUALS
- GREATER_THAN
- LESS_THAN
- GREATER_THAN_EQUAL_TO
- LESS_THAN_EQUAL_TO
- STARTS_WITH
- ENDS_WITH
- CONTAINS
- NOT_EQUALS
- IS_NULL
- IS_NOT_NULL
- DOES_NOT_CONTAIN
- DOES_NOT_START_WITH
- DOES_NOT_END_WITH

The `parameterType` can be one of the following values:

- DRAWER_NAME
- FIELD1
- FIELD2
- FIELD3
- FIELD4
- FIELD5
- DOC_TYPE
- DOC_ID
- TOTAL_PAGES
- CREATION_USER_NAME
- CREATION_TIME
- MOD_USER_NAME
- MOD_TIME
- LAST_VIEWED_USER_NAME
- LAST_VIEWED_TIME
- KEYWORDS
- QUEUE_NAME

- QUEUE_ITEM_ID
- QUEUE_ITEM_USER_NAME
- IS_UNDER_VERSION_CONTROL
- VERSION_NUMBER
- IS_CHECKED_OUT
- CHECK_OUT_USER_NAME
- CHECK_OUT_TIME
- IS_VERSION_PRIVATE
- PRIVATE_VERSION_USER_NAME
- IS_IN_PROJECT
- CUSTOM_PROPERTY
- NOTES
- DIGITAL_SIGNATURE_STATUS
- IS_IN_WORKFLOW
- WORKFLOW_USER_NAME
- NAME
- FOLDER_ID
- FOLDER_TYPE

The `propertyName` will be the name of a custom property when the `parameterType` is set to `CUSTOM_PROPERTY`.

A complete list of view columns ID's can be found in the view operation section of the [Integration Server documentation](#).

The result of the search reader will be an list of strings. The search result can be chained with subsequent channel paramters using the [List Item Reader](#).

Example

```
<c:parameter>
  <c:name>ref1</c:name>
  <c:literal>ACME Business</c:literal>
</c:parameter>
<c:parameter>
  <c:name>ref2</c:name>
  <c:literal>abc123</c:literal>
</c:parameter>
<c:parameter>
  <c:name>documentId</c:name>
```



```
<in:objectSearch>
  <in:viewType>DOCUMENT</in:viewType>
  <in:viewName>ValidViewName</in:viewName>
  <in:viewColumnID>8</in:viewColumnID>
  <in:maxWaitTime>0</in:maxWaitTime>
  <in:searchTerm>
    <in:joinOperation>AND</in:joinOperation>
    <in:operatorType>EQUALS</in:operatorType>
    <in:parameterType>FIELD1</in:parameterType>
    <in:propertyName></in:propertyName>
    <in:searchValueRef>ref1</in:searchValueRef>
  </in:searchTerm>
  <in:searchTerm>
    <in:joinOperation>AND</in:joinOperation>
    <in:operatorType>EQUALS</in:operatorType>
    <in:parameterType>custom_property</in:parameterType>
    <in:propertyName>CustomerId</in:propertyName>
    <in:searchValueRef>ref2</in:searchValueRef>
  </in:searchTerm>
</in:objectSearch>
</c:parameter>
```

In the example, the reader will search for a document from an Perceptive Content view named 'validViewName'. It will put the results from column 8 (document ID) into a list of strings. Only document matching the following criteria will be returned - [FIELD1]='ACME Business' AND [CustomerId]='abc123'.

Appendix D: Writers

Writers are components, provided by connectors, that let you configure channels to output data to applications outside of Connect Runtime. You use writers to configure the results output mapping, which lets you specify what to do with any data resulting from the action execution. You can invoke writers using specific XML tags, defined per writer.

The Content Connector provides the following writers.

Form writer

The Form writer lets you save an XML document (`w3c.Document` Java type) to a form attached to a document in Content. The writer replaces any existing form data in the document with the new data.

To output form data from a channel to a Content document, complete with the appropriate values. Include the following XML template in the channel output mapping.

```
<in:eFormTarget>
  <in:docId></in:docId>
  <in:formName></in:formName>
</in:eFormTarget>
```

The `docId` field contains the name of a reference to the document ID to which you want to output form data. The `formName` field specifies the field in the data context that contains the name of the form to replace.

Example

```
<c:parameter>
  <c:name>FormXmlResult</c:name>
  <in:eFormTarget>
    <in:docId>DocId</in:docId>
    <in:formName>APForm</in:formName>
  </in:eFormTarget>
</c:parameter>
```

In the example, `DocId` is the Content document ID. `APForm` is the name of the form to be replaced. The `FormXmlResult`, which is an XML document, is output to the referenced document and replaces the data of the specified form.

Document Pages writer

The ImageNow writer appends a list of `MappingInputStream` objects to a document in ImageNow. Each `MappingInputStream` object includes a file name for an ImageNow page file and extends from an `InputStream` that points to the file itself.

To output content from a channel to an ImageNow document, complete with the appropriate values. Include the following XML template in the channel output mapping.

```
<in:binaryPagesWriter>
  <in:docId></in:docId>
</in:binaryPagesWriter>
```

The `docId` field contains a reference to the document ID from which you want to read binary page data.

Example

```
<c:parameter>
  <c:name>PageList</c:name>
  <in:documentPagesWriter>
    <in:docId>DocId</in:docId>
  </in:documentPagesWriter>
</c:parameter>
```

In this example, `DocId` is the `ImageNow` document ID.

Object Property writer

The Object Property writer lets you output a value to an object property in Content.

To write a value from a channel to an object property, include the following XML template in the channel output mapping, complete with the appropriate values.

```
<in:objectPropertyWriter>
  <in:name></in:name>
  <in:objectId></in:objectId>
  <in:objectType></in:objectType>
  <in:propertyType></in:propertyType>
  <in:compositeChildPropertyName></in:compositeChildPropertyName>
  <in:ignoreMissingProperty></in:ignoreMissingProperty>
</in:objectPropertyWriter>
```

The `objectId` field contains the name of a reference to the object ID to which you want to output. The `name` field specifies the name of the property to modify. The `objectType` field specifies the type of the object in `ImageNow`. The following `objectTypes` are acceptable.

- DOCUMENT
- FOLDER

For documents, the following `propertyTypes` are acceptable.

- KEY
- CUSTOMPROPERTY

For folders, only `CUSTOMPROPERTY` is supported.

The `compositeChildPropertyName` field contains the name of the child property of a composite property that you want to write. This field is optional. This tag only applies when the `name` field references the name of a composite property.

The `ignoreMissingProperty` field contains a `true/false` value. When this value is set to `true` the writer will log a warning message that the `custom property` could not be found, and continue processing any unprocessed `custom properties`. When this value is set to `false` then the writer will throw an error that the property could not be found, and any unprocessed `custom properties` will not be processed. When this property is not present in the configuration it will be set to `false` by default.

Custom Properties in Object Property Writer

The following custom property types are acceptable. ImageNow identifies the custom property type. The type does not need to be passed to the writer, but the writer internally translates values in the context to `String` for the **STRING**, **NUMBER**, **DATE** and **FLAG** types and require the `String` to be in a specific format.

- **STRING** Stored in the custom property just as it exists in the context.
- **NUMBER** The number is parsed. If the string is not a number, the writer will fail.
- **DATE** The date must be in the format `MM/dd/yyyy`.
- **FLAG** The following values will be parsed into the corresponding flag value. Any other value will cause the writer to fail.
 - **0** False
 - **1** True
 - **false** False
 - **true** True
- **COMPOSITE**
 - The writer expects a single value of type `String`, `Date`, `Flag`, `Number` when `compositeChildPropertyName` is filled out. In this case the user is expecting to overwrite one child property of a composite property.
 - The writer expects in its context a `Map<String, Object>` or `List<Map<String, Object>>` when `compositeChildPropertyName` is not filled out. In this case the user is expecting to overwrite a full composite property, or list of composite properties
- **ARRAY** The writer expects in its context a `Map<String, List<Object>>` where the string in the name of the customproperty tied to the `ARRAY`. The `List<Object>` is the list of values for the customproperty. All values will overwrite any existing values on the target.

The **LIST**, **USER GROUP**, **USER LIST** custom property types are not supported by the writer at this time.

Example

```
<c:parameter>
  <c:name>FinanceGroup</c:name>
  <in:objectPropertyWriter>
    <in:name>FIELD1</in:name>
    <in:objectId>DocId</in:objectId>
```

```

    <in:objectType>document</in:objectType>
    <in:propertyType>KEY</in:propertyType>
    <in:compositeChildPropertyName></in:compositeChildPropertyName>
  </in:objectPropertyWriter>
</c:parameter>

```

In the example, `DocId` has been provided by another parameter. The example sets the `FIELD1` index key name on the ImageNow DOCUMENT referenced by the `DocId` parameter to the value of the `FinanceGroup` output parameter.

```

<c:parameter>
  <c:name>FirstNameValue</c:name>
  <in:objectPropertyWriter>
    <in:name>FullNameComposite</in:name>
    <in:objectId>DocId</in:objectId>
    <in:objectType>document</in:objectType>
    <in:propertyType>customproperty</in:propertyType>

    <in:compositeChildPropertyName>FirstName</in:compositeChildPropertyName>
  </in:objectPropertyWriter>
</c:parameter>

```

In the example, `DocId` has been provided by another parameter. The example sets the `FirstName` property of the composite property `FullNameComposite` custom property on the ImageNow DOCUMENT referenced by the `DocId` parameter to the value of the `FirstNameValue` output parameter.

Notes

- The `objectPropertyWriter` will continue to process the mapping and write to valid keys in Perceptive Content even if an invalid key is referenced during the document key processing. Regardless of how `ignoreMissingProperty` is filled out.

Document Notes Writer

The document Notes writers lets you add to a document's Notes field in Content. You are required to specify a write mode to indicate if existing Notes should be replaced, appended or prepended.

To set the Notes of a Content document, complete with the appropriate values. Include the following XML template in the channel output mapping.

```

<in:documentNotesWriter>
  <in:docId></in:docId>
  <in:mode></in:mode>
</in:documentNotesWriter>

```

The `docId` field contains the name of a reference to the document ID to which you want to add to the Notes field. The `mode` field specifies how the document Notes will be written. The following mode values are acceptable. * SET * APPEND * PREPEND

For APPEND AND PREPEND, the new Notes string will be joined with any existing Notes using a newline.

Example

```
<c:parameter>
  <c:name>DocumentNotes</c:name>
  <in:documentNotesWriter>
    <in:docId>DocId</in:docId>
    <in:mode>APPEND</in:mode>
  </in:documentNotesWriter>
</c:parameter>
```

Appendix E: SSL

Content Connector has full support for SSL/TLS. The following steps assume that SSL has already been configured in Perceptive Connect Runtime. To configure SSL in PCR, see the [PCR Install Guide](#).

Configure Integration Server

To enable SSL in Integration Server, you must enable SSL on the Tomcat server hosting Integration Server. To enable SSL in Tomcat, complete the following steps.

1. Open the `[tomcat install directory]/conf/server.xml` file.
2. Add the following to the file:

```
<Connector port="<https port number>"
protocol="org.apache.coyote.http11.Http11Protocol" maxThreads="150"
  SSLEnabled="true" scheme="https" secure="true" clientAuth="false"
  sslProtocol="TLS"
  keystoreFile="[tomcat install path]\tomcat-keystore.jks"
  keystorePass="<keystore password>"
/>
```

- **Note** Make the following replacements in the configuration above.
 - **port** The port number used to accept incoming SSL requests.
 - **keyStoreFile** The path to the keystore to be used by Tomcat.
 - **keystorePass** The password for the keystore.

After editing the configuration file, restart tomcat and test the SSL by browsing to `https://<tomcat-server-ip>:<https port number>`. See the [Tomcat SSL configuration](#) page for more information.

Configure Content Connector

1. Browse to the **Perceptive Connect Runtime Dashboard**.
2. Click **Open the Web Console**
3. In the **Web Console**, click **Perceptive Connect > View Configuration**.
4. Click on the integration server provider name (for example, Integration Server 6.7 or Integration Server 6.8)
5. Input the URI for the https endpoint to integration server (for example, `https://127.0.0.1:8443/integrationserver`)
6. Click **Save**

To verify the connection is successful, click **Main > Components** and make sure Content Connector components are **Active**. If the components are not active, check the PCR logs for possibly explanations.

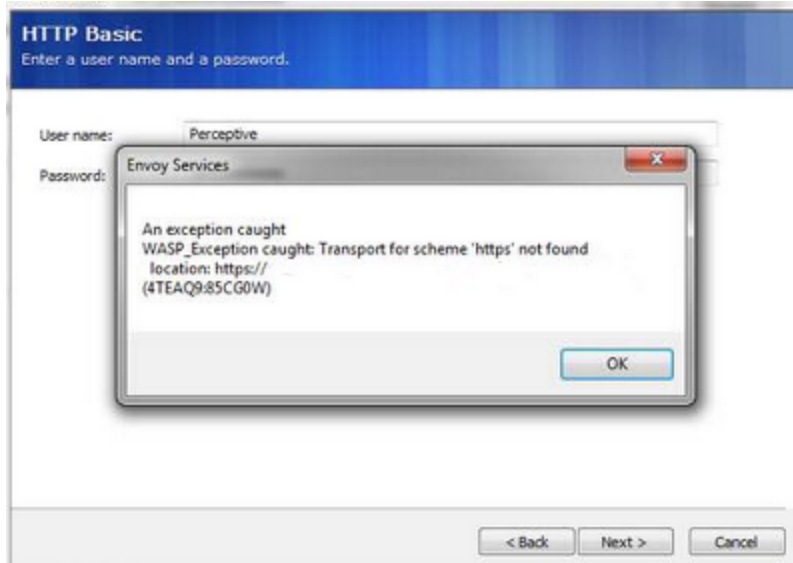
Configure the Envoy Service

Enable SSL in Envoy

To enable the HTTPS protocol for Envoy, complete the following steps.

1. Open the [imagenow server directory]\envoy\conf\security-core.xml file.
2. Change `<!--<wasp:import ref="openssl-core.xml"/>-->` to `<wasp:import ref="openssl-core.xml"/>`.

Failing to enable SSL will result in the following error when configuring an Envoy service using an HTTPS URL:



Create a new Envoy service

Create a new Envoy service by following the directions on the [Content Connector install guide](#). Be sure to use the ssl-protected endpoint when creating the service (for example, `https://127.0.0.1:443/ws/workflowTrigger?wsdl`). If Perceptive Connect Runtime requires client authentication, you may configure the envoy service to include a certificate with requests by selecting **Clientside SSL** in the **Authentication** drop-down and providing a certificate and key.

Use the new Envoy service

You can edit a current workflow to use the new Envoy service or create a new workflow. Follow the directions on the [Content Connector install guide](#) to configure the Integration ASQ. Be sure to use the Envoy service created in [Create a new Envoy service](#).

Appendix E: Endpoint Validation

Content Connector provides the ImageNow Validator service which can be used to provide validation with ImageNow. The validator service looks for a `sessionHash` cookie in the request, and validates the session hash with the configured connection provider. If the validation is successful, the request is allowed to pass through, and if the validation fails, an error is generated. The primary use case for the ImageNow Validator is Experience developers that wish to make secured requests to Perceptive Connect Runtime REST endpoints. Since the `sessionHash` cookie is included in all requests generated by Experience, it is included in requests to PCR. However, usage of the service is not restricted to Experience clients.

For general instructions on using Validator services in PCR, see the Perceptive Connect Runtime Developer Guide.