IBM Content Navigator
Version 2 Release 0

Developing Applications with IBM Content Navigator APIs

SC19-3716-00

IBM Content Navigator

Version 2 Release 0

Developing Applications with IBM Content Navigator APIs

SC19-3716-00

Note

Before using this information and the product it supports, read the information in “Notices” on page 79.

This edition applies to version 8, release 5 of later of IBM Content Manager OnDemand for Multiplatforms (product number 5724-J33), version 8, release 5 or later of IBM Content Manager OnDemand for z/OS (product number 5697-N93), version 7, release 1 or later of IBM Content Manager OnDemand for i (product number 5770-RD1), version 8, release 4 or later of IBM Content Manager Enterprise Edition (program number 5724-B19), version 8, release 4 or later of IBM Content Manager for z/OS (program number 5697-H60), version 5 release 0 or later IBM FileNet Content Manager (program number 5724-R81) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

ibm.com and related resources

Product support and documentation are available from ibm.com®.

Support and assistance

Product support is available on the web at:

IBM® Content Navigator
http://www.ibm.com/support/entry/portal/overview/software/Enterprise_Content_Manager/Content_Navigator

Information center

You can view the product documentation in an Eclipse-based information center on ibm.com. The information is available in the following information centers:

- IBM Content Manager Enterprise Edition Version 8.4 Information Center at http://publib.boulder.ibm.com/infocenter/cmgmt/v8r4m0/index.jsp?topic=/com.ibm.developingeuc.doc/eucdi000.htm
- IBM Content Manager OnDemand for Multiplatforms Version 8.5 Information Center at http://publib.boulder.ibm.com/infocenter/cmod/v8r5m0/index.jsp?topic=/com.ibm.developingeuc.doc/eucdi000.htm
- IBM FileNet® P8 Version 5.1 Information Center at http://publib.boulder.ibm.com/infocenter/p8docs/v5r1m0/index.jsp?topic=/com.ibm.developingeuc.doc/eucdi000.htm

PDF publications

You can view the PDF files online using the Adobe Acrobat Reader for your operating system. If you do not have the Acrobat Reader installed, you can download it from the Adobe Web site at http://www.adobe.com.

See the following PDF publication web site:

Product	Web site
IBM Content Navigator	http://www.ibm.com/support/docview.wss?uid=swg27025015

"How to send your comments" on page vi
Your feedback helps IBM to provide quality information. Please share any comments that you have about this information or other documentation that IBM Software Development ships with its products.

"Contacting IBM" on page vi
To contact IBM customer service in the United States or Canada, call 1-800-IBM-SERV (1-800-426-7378).

How to send your comments

Your feedback helps IBM to provide quality information. Please share any comments that you have about this information or other documentation that IBM Software Development ships with its products.

You can use any of the following methods to provide comments:

- Add comments by using the Comments pane at the bottom of every page in the information center. See Contributing to the information center.
- Send your comments by clicking the **Feedback** link at the bottom of any topic in the information center.
- Send your comments by using the online readers' comment form at http://www.ibm.com/software/data/rcf/.
- Send your comments by e-mail to comments@us.ibm.com. Include the name of the product, the version number of the product, and the name and publication number of the information (if applicable). If you are commenting on specific text, please include the location of the text (for example, a title, a table number, or a page number).

Consumability survey

Tell us how you feel about the value of quality content by taking the Importance of High Quality Technical Information survey at the following link: http://www.ibm.com/survey/oid/wsb.dll/s/ag2c1. If you want to help IBM make this product easier to install and use, take the Consumability Survey at the following link: http://www.ibm.com/software/data/info/consumability-survey/.

Related information:

Contributing to the information center
Share your opinions and contribute to the documentation in the information center. Simply sign in with your IBM ID to comment on the documentation, start discussion threads, share examples, rate topics, watch topics or threads, and more.

Contacting IBM

To contact IBM customer service in the United States or Canada, call 1-800-IBM-SERV (1-800-426-7378).

To learn about available service options, call one of the following numbers:

- In the United States: 1-888-426-4343
- In Canada: 1-800-465-9600

For more information about how to contact IBM, see the Contact IBM website at http://www.ibm.com/contact/us/.

Developing applications with IBM Content Navigator

IBM Content Navigator provides a powerful platform for building custom web applications to manage content. You can use various extension points and application programming interfaces (APIs) to extend the existing web client by adding custom actions, menus, layouts, or services. You can also use the APIs to build custom applications that incorporate IBM Content Navigator features without using the standard web client.

You use the IBM Content Navigator Java API to create plug-ins to implement the functionality that you want to add to the web client. You use the IBM Content Navigator JavaScript API to create custom widgets to use with the plug-ins. In addition, you can use IBM Content Navigator to access the APIs for the content servers.

IBM Content Navigator development architecture

IBM Content Navigator uses a model-view-controller (MVC) architecture.

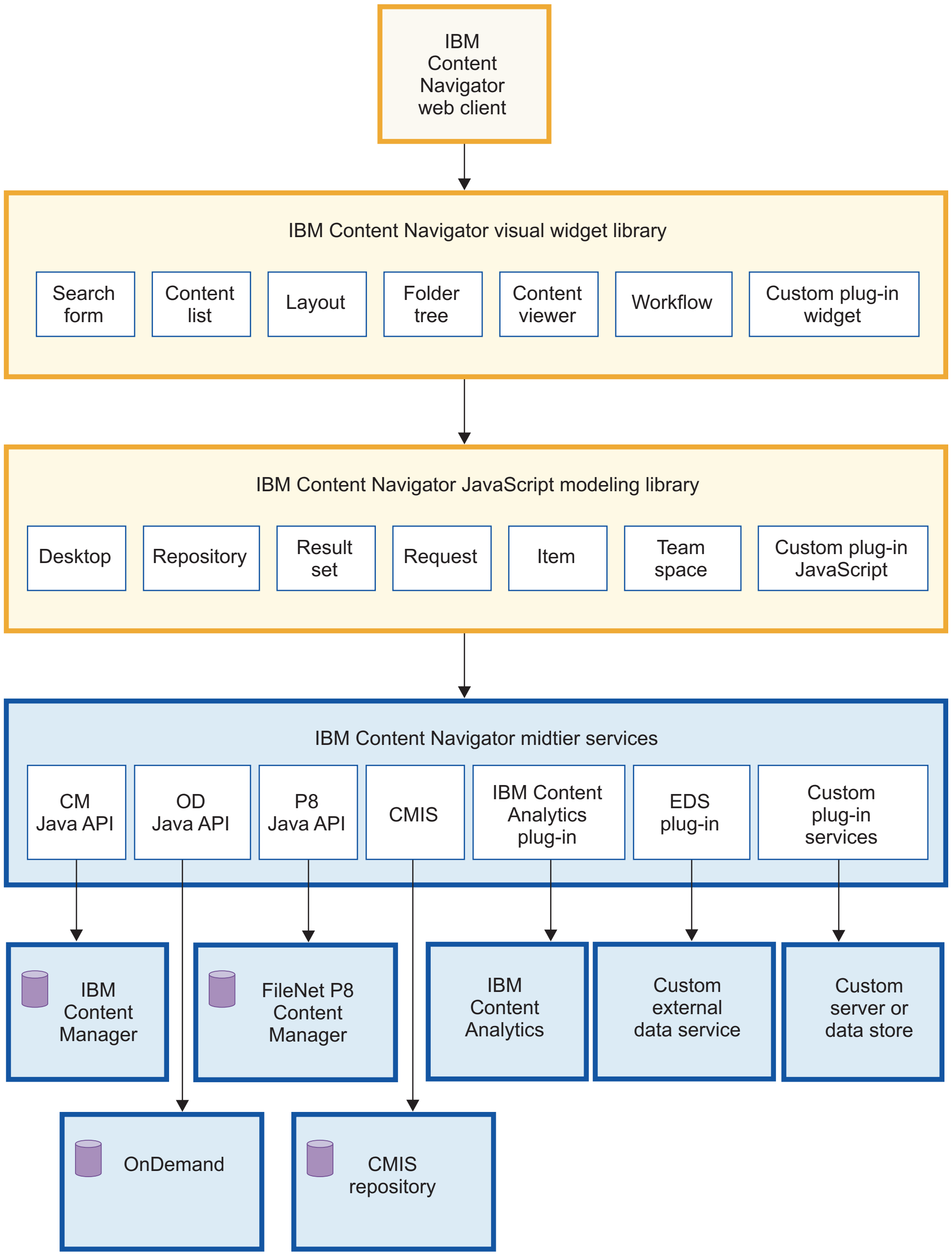
As the following diagram shows, the components of IBM Content Navigator are separated functionally according to the MVC design:

Web client
: The web client acts as a controller that provides the communication between the view and the model. Users interact with the widgets that are contained in the web client. The web client translates these interactions into the actions that are performed by instances of the model classes.

Visual widget library
: This library of JavaScript classes defines widgets that are used to build the view for IBM Content Navigator.

Modeling library
: This library of JavaScript classes defines the model for IBM Content Navigator. The modeling classes provide the business logic and data. These classes define communication with the midtier services to provide access to the content repositories and their respective application programming interfaces (APIs).

You can use the modeling classes to customize the communication and data exchange between the web client and the repositories.


IBM Content Navigator web client
IBM Content Navigator visual widget library
Search form
Content list
Layout
Folder tree
Content viewer
Workflow
Custom plug-in widget
IBM Content Navigator JavaScript modeling library
Desktop
Repository
Result set
Request
Item
Team space
Custom plug-in JavaScript
IBM Content Navigator midtier services
CM Java API
OD Java API
P8 Java API
CMIS
IBM Content Analytics plug-in
EDS plug-in
Custom plug-in services
IBM Content Manager
FileNet P8 Content Manager
IBM Content Analytics
Custom external data service
Custom server or data store
OnDemand
CMIS repository

Restriction: The CMIS service is provided only for technical preview in IBM Content Navigator V2.0.

You can customize IBM Content Navigator at each level of the architecture by using custom plug-ins. You can create plug-ins to implement custom widgets, services, and other components. By creating custom plug-ins, you can integrate information and content from external sources and augment or modify the behavior of IBM Content Navigator.

IBM Content Navigator uses plug-ins to implement various features. For example, IBM Content Analytics with Enterprise Search is integrated as a plug-in to IBM Content Navigator. The external data service plug-in augments the behavior of many IBM Content Navigator widgets to support the integration of data from sources other than the content repositories. You can use this feature to add such features as choice lists, choice list dependencies, and validation to the web client.

“IBM Content Navigator midtier services”
The IBM Content Navigator services provide the connections to the repositories. In addition, the services support features such as search and document viewing across the repositories.

Related tasks:

“Creating plug-ins to IBM Content Navigator” on page 13
You can create custom plug-ins to add features such as menus and services to IBM Content Navigator.

Related reference:

“IBM Content Navigator visual widget catalog” on page 56
The JavaScript classes in the IBM Content Navigator visual widget library are grouped by package. In some packages, the classes can be further grouped by the type of widgets that they represent.

“IBM Content Navigator modeling library structure” on page 53
The classes in the modeling library provide the business logic and data for IBM Content Navigator. These classes are used by the widgets to access and represent data in the content servers and in the IBM Content Navigator configuration.

IBM Content Navigator midtier services

The IBM Content Navigator services provide the connections to the repositories. In addition, the services support features such as search and document viewing across the repositories.

IBM Content Navigator includes services to connect to:

- IBM Content Manager Express® Edition
- IBM Content Manager for z/OS® repositories
- IBM Content Manager Enterprise Edition repositories
- IBM Content Manager OnDemand repositories
- IBM FileNet Content Manager repositories
- Content Management Interoperability Services (CMIS) repositories

IBM Content Navigator uses the application programming interfaces (APIs) of each repository type to establish connections to these repositories. You can create custom plug-ins that use these APIs to perform actions on repository items.

IBM Content Navigator also includes plug-ins to support the following services:

IBM Content Analytics with Enterprise Search plug-in
Supports searching across different repositories and data sources that are configured for use with IBM Content Navigator. You can search with simple or advanced queries. In addition, you can use facets to organize and classify items, which makes it easier to find documents quickly. The plug-in provides actions that you can use to work with the search results. For example, you can edit documents that are returned by the search or add the documents to workflows.

External data service plug-in
Supports integration of data from an external data source into IBM Content Navigator. You can use this external data to customize field properties and manage property behavior in IBM Content Navigator.

To use this plug-in, you must create a service to pass the data from the external data source to IBM Content Navigator.

IBM Content Collector viewer plug-in
Supports the use of IBM Content Collector to view archived email, IBM Lotus® Notes® documents, and IBM Connections archives.

To use this plug-in, you must:

- Provide the URL to your IBM Content Collector web service
- Update the IBM Content Navigator viewer mappings to use the IBM Content Collector viewer for the IBM Content Collector archive content types

Related tasks:

"Creating an external data service for IBM Content Navigator" on page 34
Use the EDS REST protocol to create an external data service that specifies requests to get data from an external data source to customize field properties and manage property behavior in IBM Content Navigator. You implement an external data service as a web application.

"Creating a service" on page 24
You can create a plug-in to define a service that performs operations on a content server. For example, you can create a service that provides a viewer for specific document types on a IBM FileNet P8 server.

Sample applications for IBM Content Navigator

The IBM Content Navigator software package includes several sample applications: a plug-in, an external data service, and a set of web pages. You can use these sample applications to create custom applications.

Sample plug-in application

The sample plug-in application implements a plug-in that provides a custom viewer. The files in this sample application show you how to create custom components such as menus, features, and services for use in the IBM Content Navigator client.

The files for the sample plug-in are available in the `*ECMClient_installdir*\samples\samplePages` directory. The `SamplePlugin.jar` file that you can deploy by using the IBM Content Navigator administration tool is available in the *`ECMClient_installdir`*`\navigator` directory.

Sample external data service

The sample external data service implements a service that is used to obtain values for a choice list from an external data source.

The sample web pages are available in the *`ECMClient_installdir`*`\samples\sampleEDSService` directory.

Sample web pages

This sample web application provides HTML pages that use IBM Content Navigator modeling classes and widget classes. The pages demonstrate different aspects of the IBM Content Navigator toolkit.

The sample web pages are available in the *`ECMClient_installdir`*`\samples\samplePages` directory.

Related concepts:

"Sample external data service" on page 49
In the sample external data service, the values for a choice list are used from an external data source.

Related tasks:

"Creating plug-ins to IBM Content Navigator" on page 13
You can create custom plug-ins to add features such as menus and services to IBM Content Navigator.

Constructing a URL for IBM Content Navigator

IBM Content Navigator desktops, folders, and documents can be remotely invoked by using a URL-based mechanism to send the user's browser to a specified URL with information about the unique identifier of the item in the repository. This capability makes plugging documents and folders into a custom application simple and consistent even when the source code in IBM Content Navigator is changed.

Before you begin

IBM Content Navigator must be installed and configured.

About this task

You can open IBM Content Navigator on a particular desktop, folder, or document with a URL. For example, you can embed a URL in your custom application to open IBM Content Navigator to a particular document.

You can get URLs for documents or folders when you select the item and click **Actions** > **View link**.

Each type of URL opens a different view of IBM Content Navigator:

Desktop URL
Opens to the default pane that is associated with the desktop in the full IBM Content Navigator client.

Folder URL
Opens the IBM Content Navigator link viewer, which provides a view of a specified folder.

Document URL
Opens the document in the **IBM Content Navigator Viewer** or the configured default viewer for the MIME type of the document.

A sample scenario where you can use the URL-based interface is to embed a link in the portal of a custom application to a particular document or folder in a particular repository. Review participants can review the contents of the folder or document. For instance, you can embed a link on your company-wide intranet to a folder in the repository that contains safety and emergency protocol documents. For a more advanced example, you can create a web application that adds documents to a repository, records the `docid` value, and generates the `docid` value as a confirmation number. The `docid` value can then be used in the URL syntax to create a valid link to the document.

Procedure

To construct a URL for IBM Content Navigator desktops, documents, or folders:

1. Insert the following base URL into the application:

   Desktops
   `http://`*`server`*`:`*`port`*`/navigator/?`

   Documents and Folders
   `http://`*`server`*`:`*`port`*`/navigator/bookmark.jsp?`

After you enter the base URL, use the question mark (?) before you append any parameters.

2. Append the appropriate parameters to the URL.

   Separate each parameter-value pair with an ampersand (&).

   The following lists describe the parameters that you can append to the URL, the appropriate format, and valid values for each parameter.

   Logging in to a particular desktop

Table 1. Valid parameters and values for viewing documents and folders

Parameters	Valid values
`desktop`	Specifies which desktop to log in to.⌂ Specify the ID that is displayed for the desktop in the **ID** column on the **Desktops** tab in the administration tool. Format: **`desktop`**=*desktop_ID*

Example: `http://`*`myserver.mycompany`*`.com:9080/navigator/?`**`desktop`**`=`*`AccountsPayable`*

Viewing documents

Valid parameters and values include:

Table 2. Valid parameters and values for viewing documents

Parameters	Valid values
`desktop`	Specifies which desktop to log in to.⌂ Specify the ID that is displayed for the desktop in the **ID** column on the **Desktops** tab in the administration tool. Format: **`desktop`**=*desktop_ID*
`repositoryId`	Specifies the server to log in to.⌂ Specify the ID that is displayed for the server in the ID column on the **Repositories** tab in the administration tool. Format: **`repositoryId`**=*repository_ID*
`docid`	The system-generated identification number for the document. Uses the **`template_name`** value as the prefix to this folder ID. Format: **`docid`**=*document_ID*
`template_name`	`Document`, `Folder`, or the name of the custom document class or item type. Format: **`template_name`**=*template_name*
`version`	`current`, `released`, or a specific version number. Format: **`version`**=*version_number*
`vsId`	The IBM FileNet P8 version series object, which is the Globally Unique Identifier (GUID), that is associated with the document. Format: **`vsId`**=*GUID_number*

Example: http://*myserver*.*mycompany*.com:9080/navigator/bookmark.jsp?**desktop**=AccountsPayable&**repositoryId**=AP_P8_System&**docid**=30dd879c-ee2f-11db-8314-0800200c9a66&**template_name**=APtemplate&**version**=released

Viewing folders

Table 3. Valid parameters and values for viewing folders

Parameters	Valid values
desktop	Specifies which desktop to log in to.⌂ Specify the ID that is displayed for the desktop in the **ID** column on the **Desktops** tab in the administration tool. Format: **desktop**=*desktop_ID*
repositoryId	Specifies the server to log in to.⌂ Specify the ID that is displayed for the server in the ID column on the **Repositories** tab in the administration tool. Format: **repositoryId**=*repository_ID*
docid	The system-generated identification number for the document. Uses the **template_name** value as the prefix to this folder ID. Format: **docid**=*document_ID*
template_name	Document, Folder, or the name of the custom document class or item type. Format: **template_name**=*template_name*

Example: http://*myserver*.*mycompany*.com:9080/navigator/bookmark.jsp?**desktop**=*AccountsPayable*&**repositoryId**=*CM_System*&**docid**=30dd879c-ee2f-11db-8314-0800200c9a66&**template_name**=*CHEMtemplate*

Creating plug-ins to IBM Content Navigator

You can create custom plug-ins to add features such as menus and services to IBM Content Navigator.

Before you begin

You must understand the following programming languages to create a plug-in to IBM Content Navigator:

- Java
- JavaScript

In addition, you must understand web programming to implement the JavaScript functions for the actions that are defined by the plug-in.

You should also be familiar with the Dojo Toolkit, which is the toolkit that IBM Content Navigator uses.

About this task

The plug-in that you create to use with IBM Content Navigator must provide:

- The web browser logic that enables users to call the plug-in
- The midtier server logic that enables the plug-in to load the plug-in components.

If administrators must provide additional information for the plug-in to function, you must also provide a configuration component. For example, the IBM Content Analytics with Enterprise Search plug-in requires administrators to provide a URL to the IBM Content Analytics with Enterprise Search server. Administrators use the IBM Content Navigator administration tool to configure the plug-in.

The plug-in consists of the following components that are contained in a single JAR file:

The web browser logic

The web browser logic is a required component of the plug-in that enables users to call the plug-in from the web client.

The web-browser logic component of your plug-in communicates with the IBM Content Navigator client. The web client is built by using the IBM Content Navigator visual widgets, which are based on the Dojo Toolkit. You should use the IBM Content NavigatorC visual widgets and the Dojo Toolkit to build your plug-in.

The web browser logic is implemented in JavaScript.

The midtier server logic

The midtier server logic is a required component of the plug-in. This logic calls the APIs for the content servers to retrieve the data that is used by the plug-in or that is displayed to users.

The midtier logic component of your plug-in communicates with the IBM Content Navigator services component on the web application server.

The midtier server logic is implemented in Java.

The configuration component

The configuration component is an optional component of the plug-in that enables an instance of the plug-in to be configured with the IBM Content Navigator administration tool. See step 2 in the following procedure for information about creating a configuration component for your plug-in.

The configuration component is implemented in Java.

Procedure

To create a plug-in to IBM Content Navigator:

1. Extend the `Plugin` class to create the plug-in.
2. If the plug-in requires configuration, define a configuration pane to display in the IBM Content Navigator administration tool:
   a. Extend the `PluginConfigurationPane.js` class to define parameters that need to be configured.
   b. Create an HTML file to define a form template to define the user interface for the configuration pane.
   c. In your `Plugin` class file, implement the `getConfigurationDijitClass` method to return the `PluginConfigurationPane.js` class.
3. Extend the appropriate `ecm.extension` classes to create the components that define the functionality to be provided by the plug-in:

Component type	`ecm.extension` class
Menu action	`PluginAction`
Menu	`PluginMenuType` and `PluginMenu`
Feature	`PluginFeature`
Layout	`PluginLayout`
Request filter	`PluginRequestFilter`
Response filter	`PluginResponseFilter`
Service	`PluginService`
Widget	Not applicable. Instead, you extend the Dojo `dijit._widget` class or another Dojo `dijit` class.

4. In your `Plugin` class file, implement the appropriate `get` methods to identify the custom components that are included in this plug-in:

Component type	Method
Menu action	`getActions`
Menu	`getMenuTypes` and `getMenus`
Feature	`getFeatures`
Layout	`getLayouts`
Request filter	`getRequestFilters`
Response filter	`getResponseFilters`
Service	`getServices`
Widget	`getScript`

5. Package the plug-in files in a JAR file.

6. Deploy the JAR file into your web application server.

   Important: The plug-in JAR file must be available on a URL addressable web application server or the plug-in will not work in the web client.
7. Use the IBM Content Navigator administration desktop to add the new plug-in.

Sample plug-in definition and configuration files

The following files in the sample plug-in define the plug-in and the configuration:

Table 4. Sample files that define the plug-in and the configuration

File	Description
`SamplePlugin.java`	This file extends the `Plugin` class to list the custom components that are provided by the sample plug-in.
`SamplePlugin.js`	This file extends the `Plugin` class to list the custom components that are provided by the sample plug-in.
`ConfigurationPane.js`	This file defines the configuration pane that is used for the sample plug-in.
`ConfigurationPane.html`	This file provides the HTML form template that is used for the sample plug-in configuration pane.

"Creating the plug-in components" on page 16
You can create plug-ins to add components such as menus, services, widgets, and layouts to IBM Content Navigator.

"Packaging the plug-in components" on page 28
You extend the `Plugin` class to identify the components that are used for a custom plug-in. You then create a JAR file to package the components deploy to IBM Content Navigator.

Related concepts:

"IBM Content Navigator development architecture" on page 3
IBM Content Navigator uses a model-view-controller (MVC) architecture.

Related reference:

"Sample applications for IBM Content Navigator" on page 7
The IBM Content Navigator software package includes several sample applications: a plug-in, an external data service, and a set of web pages. You can use these sample applications to create custom applications.

Related information:

Dojo Toolkit Reference Guide
Find information about the Dojo Toolkit that is used by IBM Content Navigator.

Class ecm.widget.admin.PluginConfigurationPane
Provides a base class that can be extended to create a configuration interface for a plug-in.

Class Plugin
Provides the main class of an IBM Content Navigator plug-in. The abstract methods provide the name and version of the plug-in, as well as actions, services, and scripts provided by the plug-in.

Creating the plug-in components

You can create plug-ins to add components such as menus, services, widgets, and layouts to IBM Content Navigator.

"Creating a menu action" on page 17
You can create a plug-in to provide an action to use in IBM Content Navigator. For example, you might define a custom action that imports documents or that generates a custom report generation for a set of selected documents. This action can be added to toolbars and context menus.

"Creating a menu" on page 18
You can create a plug-in to provide a custom toolbar or context menu to use in IBM Content Navigator.

"Creating a feature" on page 20
You can create a plug-in to add a major feature to IBM Content Navigator. For example, you might create a feature to show statistical data on content usage or a summary of recent document activity. You might also create a feature or to select documents for archival. This feature can be displayed with the features that are provided by IBM Content Navigator such as favorites, browse, search, and administration.

"Creating a layout" on page 21
You can create a plug-in to provide a custom layout to use in IBM Content Navigator. The layout defines the arrangement of the widgets that are used in a desktop.

"Creating a request or response filter" on page 22
You can create a plug-in to filter a request that is made to a service or to filter a response that is received from a service. For example, you might create a request filter to modify the parameters that are sent to a service to provide custom validations and error handling. You can also create a request filter to override a service. For example, you might create a response filter to apply custom formatting to the data that is received from a service before the data is displayed.

"Creating a service" on page 24
You can create a plug-in to define a service that performs operations on a

content server. For example, you can create a service that provides a viewer for specific document types on a IBM FileNet P8 server.

"Creating a viewer" on page 26
You can create a plug-in to provide a content viewer. Users can then map the custom viewer to document types by using the IBM Content Navigator administration tool.

"Creating a widget" on page 27
You can create a widget to provide the user interface for your plug-in. For example, you might create a widget that enables users to run a custom action. You also might create a widget to enable administrators to configure the plug-in.

Creating a menu action

You can create a plug-in to provide an action to use in IBM Content Navigator. For example, you might define a custom action that imports documents or that generates a custom report generation for a set of selected documents. This action can be added to toolbars and context menus.

About this task

Procedure

To create an action:

1. Implement a JavaScript function to define the behavior of the action. For information about actions in IBM Content Navigator, see the `ecm.model.Action.js` class.
2. If you want to specify when the action is enabled or disabled, extend the `ecm.model.Action.js` class.
3. Extend the `PluginAction.java` class to define the plug-in for the action:
   a. Implement the `getActionFunction` method to return name of the JavaScript function that you created in step 1.
   b. If this action overrides the standard code for enabling actions, implement the `getActionModelClass` method to return name of the JavaScript class that you created in step 2.
4. Add the action as a component in your `Plugin.java` subclass. To add the action, implement the `getFeatures` method to return an instance of the `PluginAction.java` class that you created in step 3.

Sample plug-in files for a custom menu action

The following files in the sample plug-in define a custom menu action:

Table 5. Sample files that define a plug-in menu action

File	Description
`SamplePluginAction.java`	This file extends the `PluginAction.java` class to define a sample action. The behavior of this action is defined by the `samplePluginAction` function that is defined in the `SamplePlugin.js` file.

Table 5. Sample files that define a plug-in menu action (continued)

File	Description
`SamplePluginCheckInAction.java`	This file extends the `PluginAction.java` class to define a sample action. The behavior of this action is defined by the `samplePluginAction` function that is defined in the `SamplePlugin.js` file.
`SamplePluginFilteredAction.java`	This file extends the `PluginAction.java` class to define a sample action. This class implements an action that is available only for documents with a MIME type of text. The behavior of this action is defined by the `samplePluginFilteredAction` function that is defined in the `SamplePlugin.js` file.
`CustomAction.js`	This file implements the `isEnabled` function so that an action is enabled only for documents with a MIME type of text.

What to do next

Add the action to a custom toolbar or context menu by implementing the `getMenuItems` method in a `PluginMenu` subclass to return an instance of this `PluginAction` class. This method adds the action to the custom toolbar or menu that is represented by the `PluginMenu` subclass.

After you add the plug-in in the web client, use the IBM Content Navigator administration tool to add the action to a toolbar or context menu.

Related information:

Class ecm.model.Action
Represents a user-executable action. This action can be configured for a desktop in the IBM Content Navigator administration tool.

Class PluginAction
Provides an abstract class that is extended to define a client-side action that is provided by a plug-in. The actions that are defined by subclasses of this class appear on the IBM Content Navigator toolbar and on the context menus within the content list.

Class Plugin
Provides the main class of an IBM Content Navigator plug-in. The abstract methods provide the name and version of the plug-in, as well as actions, services, and scripts provided by the plug-in.

Creating a menu

You can create a plug-in to provide a custom toolbar or context menu to use in IBM Content Navigator.

About this task

To create a menu, you must first create the menu type. The menu type indicates where and how the menu will appear in the user interface. For example, you might define a menu type to create a context menu for a custom widget. You can use the menu type for multiple menus.

Procedure

To create a custom menu:

1. Extend the `PluginMenuType.java` class to define the menu type:

   If the menu is a toolbar, implement the `isToolbar` method to return `true`. If you do not implement this method or if the method returns `false`, the menu is a context menu.

   For a default menu, the menu type provides an identifier for the default menu. When a user creates a copy of this menu, the menu type is identified in the copy to link it back to the original menu.
2. Extend the `PluginMenu.java` class to define the menu:
   a. Implement the `getId` method to return the identifier for the menu. Prefix the identifier with "Default" if users will be able to create copies of this menu.
   b. Implement the `getMenuType` method to return the `PluginMenuType` class that you created in step 1.
   c. Implement the `getMenuItems` method to return instances of the actions that are included in the menu.
3. Add the menu as a component in your `Plugin.java` subclass:
   a. Implement the `getMenuTypes` method to return an instance of the `PluginMenuType.java` subclass that you created in step 1.
   b. Implement the `getMenus` method to return an instance of the `PluginMenu.java` subclass that you created in step 2.

Sample plug-in files for a custom menu

The following files in the sample plug-in define a context menu and two toolbar menus:

Table 6. Sample files that define a plug-in menu

File	Description
SamplePluginContextMenuMenuType.java	This file extends the `PluginMenuType.java` class to create a menu type for a context menu.
SamplePluginContextMenuMenu.java	This file extends the `PluginMenu.java` class to defines a context menu that includes the following actions: • A sample action, which is defined in the `SamplePluginAction.java` file • Properties • Help • About • Preview • Open The following actions are in the available actions box: • Custom Checkin • Only For Text Docs
SamplePluginToolbarMenuType.java	This file extends the `PluginMenuType.java` class to create a menu type for a toolbar.

Table 6. Sample files that define a plug-in menu (continued)

File	Description
SamplePluginToolbarMenu.java	This file extends the `PluginMenu.java` class to defines a toolbar that includes the following actions: • A sample action, which is defined in `SamplePluginAction.java` file • Help • About
SamplePluginToolbarMenuType2.java	This file extends the `PluginMenuType.java` class to create a menu type for a toolbar.
SamplePluginToolbarMenu2.java	This file extends the `PluginMenu.java` class to defines a toolbar that includes the following actions: • A sample action, which is defined in the `SamplePluginAction.java` file • Custom Checkin, which is defined in the `SamplePluginCheckInAction.java` file • Help

What to do next

After you add the plug-in in the web client, use the IBM Content Navigator administration tool to add the menu to a desktop.

Related information:

Class PluginMenuType
Provides an abstract class that is extended to define a custom menu type provided by a plug-in.

Class PluginMenu
Provides an abstract class that is extended to define a menu in a plug-in.

Class Plugin
Provides the main class of an IBM Content Navigator plug-in. The abstract methods provide the name and version of the plug-in, as well as actions, services, and scripts provided by the plug-in.

Creating a feature

You can create a plug-in to add a major feature to IBM Content Navigator. For example, you might create a feature to show statistical data on content usage or a summary of recent document activity. You might also create a feature or to select documents for archival. This feature can be displayed with the features that are provided by IBM Content Navigator such as favorites, browse, search, and administration.

Procedure

To define a feature:

1. Create an icon to represent the feature in the web client. The icon must be 64 pixels wide by 64 pixels high. For best results, create the icon as a transparent PNG file.

2. Create a widget to provide the primary pane that defines the user interface for the feature. This widget provides users with access to the behavior of the feature.
3. Extend the `PluginFeature` class to describe the feature:
   a. Implement the `getIconUrl` method to return the icon that you created in step 1 on page 20.
   b. Implement the `getContentClass` method to return an instance of the widget that you created in step 2.
4. Add the service as a component in your `Plugin.java` subclass. Implement the `getFeatures` method to return an instance of the `PluginFeature` class that you created in step 3.

Sample plug-in files for a custom feature

The following files in the sample plug-in define a feature:

Table 7. Sample files that define a plug-in feature

File	Description
`SamplePluginFeature.java`	This file extends the `PluginFeature` class to create a sample feature.

What to do next

After you add the plug-in in the web client, use the IBM Content Navigator administration tool to add the feature to a desktop. You add the feature on the **Appearance** tab for the desktop.

Related information:

Class PluginFeature
Provides an abstract class that is extended to define a feature provided by the plug-in. Features are major functional areas that typically appear as icons along the left side of the user interface. Features are configurable for each desktop. Examples of features include Browse and Favorites.

Class Plugin
Provides the main class of an IBM Content Navigator plug-in. The abstract methods provide the name and version of the plug-in, as well as actions, services, and scripts provided by the plug-in.

Creating a layout

You can create a plug-in to provide a custom layout to use in IBM Content Navigator. The layout defines the arrangement of the widgets that are used in a desktop.

Procedure

To create a layout:

1. Extend the `ecm.widget.layout.MainLayout.js` class to create the primary widget for the layout:
   a. Implement the `getAvailableFeatures` method to return the features that to be available in the layout. Each feature is declared as an instance of the `ecm.model.Feature` class.

   b. Implement the `setFeatures` method to configure how the features are arranged in the layout.
2. Extend the `PluginLayout.java` class. Implement the `getLayoutClass` method to return an instance of the widget class that you created in step 1 on page 21.
3. Add the layout as a component in your `Plugin.java` subclass. To add the feature, implement the `getLayouts` method to return an instance of the `PluginLayout.java` class that you created in step 2.

Sample plug-in files for a custom layout

The following files in the sample plug-in define a layout:

Table 8. Sample files that define a plug-in layout

File	Description
`SampleLayout.js`	This file extends the `MainLayout.js` class to create the widget that is used for the sample plug-in layout.
`SamplePluginLayout.java`	This file extends the `PluginLayout.java` class to define sample plug-in layout.

What to do next

After you add the plug-in in the web client, use the IBM Content Navigator administration tool to associate the layout with a desktop.

Related information:

Class ecm.widget.layout.MainLayout
Provides the main layout for IBM Content Navigator or for similar desktop layouts. This class displays the launch bar that contains the features on the left side of the layout.

Class PluginLayout
Provides an abstract class that is extended to define a layout that is provided by a plug-in. The layout is the main widget for the desktop and defines the overall arrangement of widgets in the desktop. The particular layout that is used for a desktop is selected in the IBM Content Navigator administration tool.

Class Plugin
Provides the main class of an IBM Content Navigator plug-in. The abstract methods provide the name and version of the plug-in, as well as actions, services, and scripts provided by the plug-in.

Creating a request or response filter

You can create a plug-in to filter a request that is made to a service or to filter a response that is received from a service. For example, you might create a request filter to modify the parameters that are sent to a service to provide custom validations and error handling. You can also create a request filter to override a service. For example, you might create a response filter to apply custom formatting to the data that is received from a service before the data is displayed.

About this task

The external data service plug-in uses request and response filters to enable a service to provide custom choice lists and validation.

To use a filter, a plug-in must handle the `JSONArtifact` object that is sent in the request or the `JSONObject` object that is returned in the response.

Important: The JSON format that is used for request and response filters is subject to change without notice. If you create request and response filters for plug-ins, you might need to make updates to your code if a new version of the IBM Content Navigator Java application programming interface is released.

Procedure

To create a request or response filter:

1. If you are creating a request filter, extend the `PluginRequestFilter` class:
   a. Implement the `getFilteredServices` method to return the names of the services for which this filter is used.
   b. Implement the `filter` method to define the filter that is to be applied to the request.

      Tip: Examine the JSON object that is sent to the service to determine the requirements for the filter method.
2. If you are creating a response filter, extend the `PluginResponseFilter` class:
   a. Implement the `getFilteredServices` method to return the names of the services for which this filter is used.
   b. Implement the `filter` method to define the filter that is to be applied to the request.

      Tip: Examine the JSON object that is returned by the service to determine the requirements for the filter method.
3. Add the filter as a component in your `Plugin.java` subclass:

Type of filter	Add to the `Plugin.java` subclass
Request	Implement the `getRequestFilters` method to return an instance of the `PluginRequestFilter.java` class that you created in step 1.
Response	Implement the `getResponseFilters` method to return an instance of the `PluginResponseFilter.java` class that you created in step 2.

Sample plug-in files for a custom response filter

The following files in the sample plug-in define a response filter.

Table 9. Sample files that define a plug-in response filter

File	Description
`SamplePluginResponseFilter.java`	This file extends the `PluginResponseFilter` class to define a filter that applies custom property formatting to search results.

Related information:

Class PluginRequestFilter
Provides an abstract class that is extended to create a filter for requests to a particular service. The filter is provided with the request parameters before being examined by the service. The filter can change the parameters or reject the request.

Class PluginResponseFilter
Provides an abstract class that is extended to create a filter for responses from a particular service. The response from the service is provided to the filter in JSON format before it is returned to the web browser. The filter can then modify that response, and the modified response is returned to the web browser.

Creating a service

You can create a plug-in to define a service that performs operations on a content server. For example, you can create a service that provides a viewer for specific document types on a IBM FileNet P8 server.

About this task

You use the application programming interfaces (APIs) that are native to the content server to define the operations that are performed by the service.

Tip: Follow best practices for servlets when you implement an IBM Content Navigator plug-in service:

- Because the application server is multithreaded, the plug-in service needs to handle multiple threads safely.
- The service will be called on multiple threads for multiple users; therefore, do not use instance variables. Using instance variables can have side effects such as unintentional sharing of information among users.

Procedure

To create a service:

1. Extend the `PluginService` class to create the service:
   a. Implement the `execute` method to provide the service with access to the APIs for the content server.
   b. If this service replaces one of the IBM Content Navigator services, implement the `getOverriddenService` method to identify that service.
   c. If the service requires configuration, use the `PluginServiceCallbacks.loadConfiguration` method to obtain the values that were configured for the service.
2. If the service provides a custom viewer, extend the `PluginViewerDef` class to define the URL template to launch the viewer.
3. Add the service as a component in your `Plugin.java` subclass:
   a. Implement the `getServices` method to return an instance of the `PluginService.java` class that you created in step 1.
   b. If the service provides a custom viewer, implement the `getViewers` method to return an instance of the `PluginViewerDef` class that you created in step 2.

Sample plug-in files for a custom service

The following files in the sample plug-in define a service:

Table 10. Sample files that define a plug-in service

File	Description
`SamplePluginService.java`	This file extends the `PluginService` class to define a service that obtains system-related details about a document. To obtain this information, the service invokes the appropriate API depending on whether the document is in an IBM Content Manager, IBM Content Manager OnDemand, or IBM FileNet P8 repository.
`SamplePluginViewerDef.java`	This file extends the `PluginViewerDef` class to define the URL template to launch the viewer that is defined in the `SamplePluginViewerService.java` file.
`SamplePluginViewerService.java`	This file extends the `PluginService` class to define a service that provides a custom viewer that does simple formatting of an archived RSS feed.

What to do next

Call the service as needed from your custom actions, viewers, or features. You then use the IBM Content Navigator to associate these components with a desktop, which makes the service available to users.

Related concepts:

"IBM Content Navigator midtier services" on page 5
The IBM Content Navigator services provide the connections to the repositories. In addition, the services support features such as search and document viewing across the repositories.

Related information:

Class PluginService
Provides an abstract class that is extended to create a class implementing each service provided by the plug-in. Services are actions, similar to servlets or Struts actions, that perform operations on the IBM Content Navigator server. A service can access content server application programming interfaces (APIs) and J2EE APIs.

Class PluginViewerDef
Provides an abstract class that is extended to define a viewer provided by the plug-in. The viewer can be used in viewer mappings to identify the types of documents for which the viewer should be invoked. Viewers are launched within their own iframes or web browser windows, based on a URL that is built by IBM Content Navigator by using a template that is defined in this class.

Class Plugin
Provides the main class of an IBM Content Navigator plug-in. The abstract methods provide the name and version of the plug-in, as well as actions, services, and scripts provided by the plug-in.

Creating a viewer

You can create a plug-in to provide a content viewer. Users can then map the custom viewer to document types by using the IBM Content Navigator administration tool.

Procedure

To create a viewer:

1. Extend the `PluginService` class to create a service that provides the viewer.
2. Extend the `PluginViewerDef` class to provide a URL template that will open the viewer.

Sample plug-in files for a custom viewer

The following files in the sample plug-in define a viewer:

Table 11. Sample files that define a plug-in viewer

File	Description
`SamplePluginViewerDef.java`	A definition of a new custom viewer. This class defines the sample viewer service as a viewer. The launch URL pattern describes how to invoke the viewer and provides the parameters that the viewer service uses to retrieve and format the document for viewing.
`SamplePluginViewerService.java`	This class extends the `PluginService.java` class to implement a service. The service provides a custom viewer that performs simple formatting of an archived RSS feed.

Related information:

Class PluginService
Provides an abstract class that is extended to create a class implementing each service provided by the plug-in. Services are actions, similar to servlets or Struts actions, that perform operations on the IBM Content Navigator server. A service can access content server application programming interfaces (APIs) and J2EE APIs.

Class PluginViewerDef
Provides an abstract class that is extended to define a viewer provided by the plug-in. The viewer can be used in viewer mappings to identify the types of documents for which the viewer should be invoked. Viewers are launched within their own iframes or web browser windows, based on a URL that is built by IBM Content Navigator by using a template that is defined in this class.

Creating a widget

You can create a widget to provide the user interface for your plug-in. For example, you might create a widget that enables users to run a custom action. You also might create a widget to enable administrators to configure the plug-in.

About this task

You can reuse the widgets in the IBM Content Navigator visual widget library in the user interface for your custom actions, features, layouts, and services. For example, rather than creating your own tree widget to browse a folder hierarchy, you can use the `ecm.widget.FolderTree` widget.

If no IBM Content Navigator widget meets your requirements, you can create a custom widget. The classes in the IBM Content Navigator JavaScript modeling library that make it easier to build new widgets from the basic Dojo `dijit` widgets. This library includes models of Dojo data stores that provide common methods for representing data. These data store model classes enable multiple widgets to easily use and present data. The library also includes models of Dojo trees that you can use to obtain data for navigation trees in custom widgets.

Procedure

To create a custom widget:

1. If your widget is extending the Dojo `dijit._Templated` class, create an HTML template to define the visual presentation of the widget, including buttons, fields, and labels. This template is used to generate the initial DOM structure for the widget.
2. Create a JavaScript file to define the class for the custom widget by extending the Dojo `dijit._widget` class or another Dojo `dijit` class.

   Use the Dojo `dijit._Templated` class and reference the template that you created in step 1.
3. Create a JavaScript file to instantiate the widget in the plug-in.
4. Add the widget script as a component in your `Plugin.java` subclass. To add the script, implement the `getScript` method to return the JavaScript file that you created in step 3.

Sample plug-in files for custom widgets

The following files in the sample plug-in define custom widgets:

Table 12. Sample files that define custom widgets

File	Description
`MessagesDialog.js`	This file extends the Dojo `dijit.Dialog` class to define widget that provides a message dialog box.
`MessagesDialog.html`	This file provides an HTML template that is used for the widget that is defined in the `MessagesDialog.js` file.
`PopupDialog.js`	This file extends the Dojo `dijit.Dialog` class to define a widget that provides a pop-up dialog box.
`PopupDialog.html`	This file provides an HTML template that is used for the widget that is defined in the `PopupDialog.js` file.
`SamplePlugin.js`	This file instantiates the `MessagesDialog.js` class to display progress messages for the sample plug-in. The file instantiates the `PopupDialog.js` class to open the help file for the sample plug-in in a pop-up window.

What to do next

Use the widget to provide a user interface for the custom actions or features that are defined by your plug-in.

Related information:

Class Plugin
Provides the main class of an IBM Content Navigator plug-in. The abstract methods provide the name and version of the plug-in, as well as actions, services, and scripts provided by the plug-in.

Writing Your Own Widget
See this web page for detailed information about creating and modifying widgets by using the Dojo Toolkit.

Packaging the plug-in components

You extend the `Plugin` class to identify the components that are used for a custom plug-in. You then create a JAR file to package the components deploy to IBM Content Navigator.

Procedure

To package the plug-in:

1. Create a JAR file that contains the files that are used for the plug-in.

   The JAR file must use the following structure:

```
packageName
   // Include all the Java classes that extend the Plugin class and
   // any other Java classes that are required by the plug-in
   Java class 1
   Java class 2
```

```
      ...
      WebContent
      // Include all the image, JavaScript, and HTML files
      // that are required by the plug-in
   WEB-INF
      manifest.mf
```

In the *packageName* folder, include the Java class files that are defined for the plug-in. Create the subfolders based on the package names of the Java classes. Follow the standard JAR file structure by using nested folders for nested packages.

Important: All the Java classes must be included in a named package. You cannot use the default package.

2. Modify the `manifest.mf` file for your plug-in to include the following property:

   ```
   Plugin-Class: pluginClassName
   ```

   pluginClassName is the name of your `Plugin.java` subclass.

   You can also include any of the standard properties for a JAR file in the `manifest.mf` file.

Related information:

Class Plugin
Provides the main class of an IBM Content Navigator plug-in. The abstract methods provide the name and version of the plug-in, as well as actions, services, and scripts provided by the plug-in.

External data services

You can use the external data service (EDS) representational state transfer (REST) protocol to create an external data service to get data from an external source, such as a JSON file or a table in a database, to customize field properties and manage property behavior in IBM Content Navigator. Review the data flow diagram to understand how an external data service submits and returns requests between the IBM Content Navigator client and the external data source.

When you create an external data service, your existing data is integrated with IBM Content Navigator field values and other field properties, but you continue to store and maintain the data only in the original, authoritative data source. You access the data without moving or copying that data to a separate repository, so the source remains in the original data store. The external data source must remain available to IBM Content Navigator, so that the external data can be accessed whenever the business user invokes the service through the web client.

An advantage to using an external data service is that you do not need to modify the IBM Content Navigator source code to customize the user interface properties and values. Therefore, upgrades and other major changes to IBM Content Navigator do not affect the property data that is obtained by an external data service, because the data and external data service are located separately from IBM Content Navigator source code.

Where external data services can be implemented

You can use an external data service to customize the following field properties and property behaviors when users add documents, add folders, check-in documents, use entry templates, process workflow step properties and workflow filter criteria fields (IBM FileNet P8), and create or use searches:

Look up values in a database to create choice lists
Create choice lists by using existing data that is managed in a different content repository or data source than the one that is connected to IBM Content Navigator.

For example, you can use values in a JSON file that is located and managed in an external server or repository.

Prefill properties
Specify prefilled properties and default values.

For example, you can prefill fields with custom default values that are based on a particular class ID, authenticated user, or the parent folder.

Specify property dependencies
Define dependencies between properties.

For example, you might specify a dependency between a geographic region choice list property and an office branch choice list property so that when a user chooses a geographic region, the subsequent choice list that is dependent on the selected geographic region contains only the office branches that pertain to that geographic region.

Set minimum and maximum values
Specify an integer, float, or date to define the maximum or minimum value for a property.

Restriction: You cannot reset the minimum value or maximum value to be less restrictive than the minimum value or maximum value that is specified in the repository that you are using.

For example, if the minimum value in the repository is 100, the service can set the value to 150, but not to 50.

Set read-only status
Set a property to be read-only.

For example, you might create a property that requires a particular value. To prevent users from entering a different value that could cause an error, you can specify the correct default value and make that property read-only.

Set required status
Set a property to be a required field. When you use this attribute on a property, an asterisk appears in the user interface to indicate that the field is required. Users can not proceed from the page or dialog box unless the field contains a value.

Set hidden status
Hide a property from the user interface.

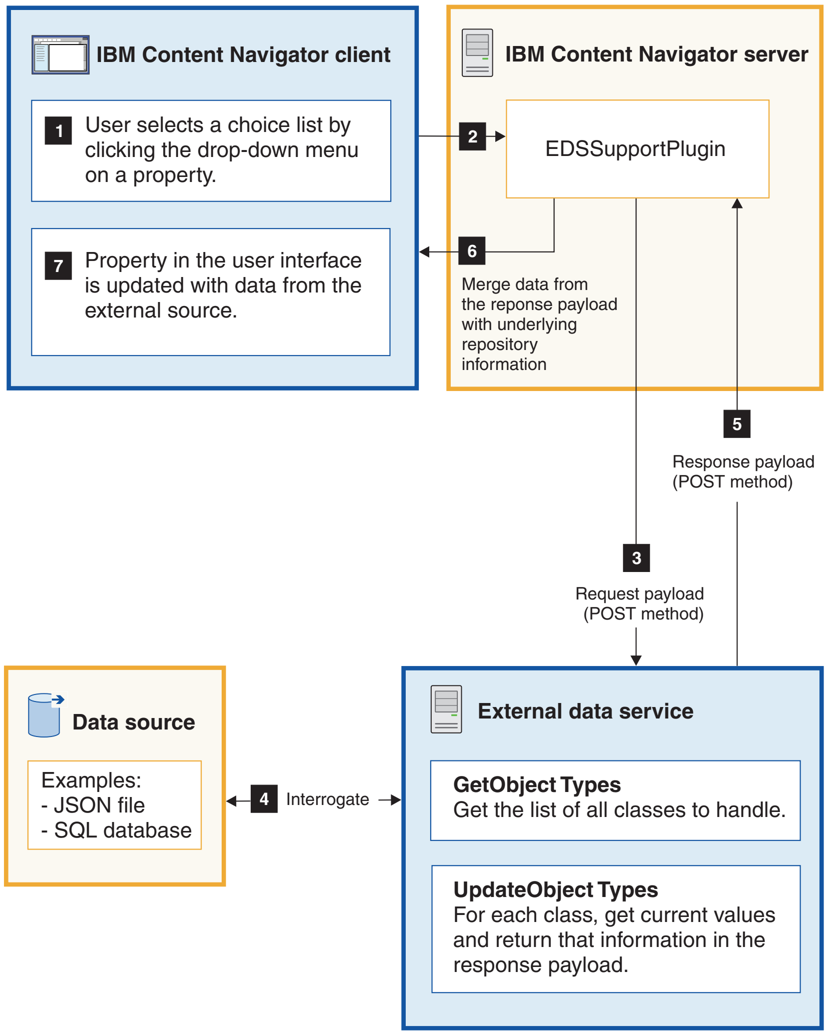
For example, you might create a choice list that dynamically determines subsequent text input fields to present in a form. To hide a property that does not apply in a particular situation, you can use the hidden attribute.

Implement property validation and error checking
Show a custom message or provide assistance when users enter values into a property field.

Restriction: You cannot use custom validation for object properties, reference attributes, read-only properties, hidden properties, or search criteria.

External data service architecture and data flow

When an external data service is implemented for a certain action or property, the service is invoked when a business user interacts with that item in the web client. The following diagram shows how an external data service submits and returns requests.


IBM Content Navigator client
1 User selects a choice list by clicking the drop-down menu on a property.
7 Property in the user interface is updated with data from the external source.
IBM Content Navigator server
2
EDSSupportPlugin
6
Merge data from the reponse payload with underlying repository information
5
Response payload (POST method)
3
Request payload (POST method)
Data source
Examples:
- JSON file
- SQL database
4 Interrogate
External data service
GetObject Types
Get the list of all classes to handle.
UpdateObject Types
For each class, get current values and return that information in the response payload.

The external data service works by using two services, which the application developer needs to create:

- A `GetObjectTypes` service to get the list of all classes, item types, or workflow information that the external data service needs to handle.
- An `UpdateObjectType` service for each class to get the current attributes or values and return that information in the response payload.

The external data service is invoked when a user opens a certain dialog box or tries to set a value on a property in the user interface. The middle tier services sends a request payload to the external data service, which interrogates the external data source for the requested information about the classes and attributes. The information from the response payload is then merged with the underlying information in the repository that you are using, for example, IBM Content Manager or FileNet P8.

The precedence order in which data appears in the user interface is that IBM Content Navigator first uses the class attributes, then checks for values that are specified in the active selected entry template, and finally, IBM Content Navigator checks for values specified by the external data service.

“Creating an external data service for IBM Content Navigator” on page 34
Use the EDS REST protocol to create an external data service that specifies

requests to get data from an external data source to customize field properties and manage property behavior in IBM Content Navigator. You implement an external data service as a web application.

"External data service REST protocol specifications" on page 36
Understand the specifications for the object type request, the particular object type request, and the responses that you need to create for each request.

"Sample external data service" on page 49
In the sample external data service, the values for a choice list are used from an external data source.

Creating an external data service for IBM Content Navigator

Use the EDS REST protocol to create an external data service that specifies requests to get data from an external data source to customize field properties and manage property behavior in IBM Content Navigator. You implement an external data service as a web application.

Before you begin

Understand which properties and areas in IBM Content Navigator you can customize through an external data service.

Review the specifications for defining an object types request and a particular object type request.

You need the following skills to implement an external data service:

- Java
- Java 2 Platform, Enterprise Edition
- Implement a web application with two services, for example, two Java servlets
- Database development
- Read and write JSON

About this task

As you design your service, consider the following tips to improve the usability of your choice lists and other field properties:

Reduce the size of choice lists by adding properties that can narrow subsequent lists
For example, you can use questions to narrow the scope of the list, or use dependencies between properties to break long choice lists into shorter choice lists.

Create dependent choice lists
Dependent choice list values are replaced with a different list depending on the value that is selected in the preceding parent list. For example, you can set up **Region** and **Branch Office** choice lists, where the **Branch Office** list depends on the **Region** list selection. If the user selects a different region, the **Branch Office** list is cleared and populated by a different list of branch offices.

Implement custom validation so that users can immediately resolve any errors in data entry.
For example, if you implement text fields that have character restrictions, instead of using a custom validation error message to display only a

description of the error, you can also provide users with a list of the unsupported characters so that users can avoid typing those characters in the future.

Procedure

To create an external data service:

1. Create a service that gets the list of classes to be handled by the external data service. For workflow information, you create a service that gets the list of registered service types instead of classes.

   For example, you might create a `GetObjectTypes` service:

   - Request: `GET /types?repositoryId=`*`repository_Id`*.
   - Response:

     ```
     [
      {
       "symbolicName" : "object_type_or_service_type"
      }
      // More object types
     ]
     ```

     This service is invoked with the `GET` request, which provides the repository ID. The response in this example is an array of name values in a JSON file. This list must contain all the classes that are to be supported by the external data service.

2. Create a service that gets current attributes and values for each class that is included in the service that you created in the previous step. This service gets current attributes and values for a class and then returns information about those properties to the middle tier services. The data that is posted is JSON, and the response is also JSON.

   For example, you might create an `UpdateObjectType` servlet that handles a POST method with a URL of the form `/type/`*`object_type_name`*:

   - Request:

     ```
     POST /type/class_name_or_item_type_name

     {
      "repositoryId":"target_repository",
      "objectId" : "if_an_existing_instance,_the_GUID_or_PID",
      "requestMode" : "indicates_context_that_info_is_being_requested",
      "externalDataIdentifier" : "opaque_identifier_meaningful_to_service",
      "properties":
      [
       {
        "symbolicName" : "symbolic_name",
        "value" : "the_current_value",
       }
       // More properties ...
      ]
     }
     ```

   - Response:

     ```
     {
       "externalDataIdentifier": "opaque_identifier_meaningful_to_the_service",
       "properties":
       [
          {
            "symbolicName": "symbolic_name",
            "choiceList":
            {
              "displayName": "display_name",
     ```

```
                "choices":
                [{
                    "displayName": "name",
                    "value": value
                  },
                  // More choices in the choice list
                ]
           }
             }
          // More properties
          ]
       }
```

What to do next

After you write the external data service, you must deploy and start your service in the web application server, and register the `EDSSupportPlugin` plug-in in the IBM Content Navigator administration tool.

Related concepts:

"IBM Content Navigator midtier services" on page 5
The IBM Content Navigator services provide the connections to the repositories. In addition, the services support features such as search and document viewing across the repositories.

Related tasks:

Registering and configuring plug-ins

Related reference:

"Example: GET method request and response" on page 39
This sample code submits a `GET` request to an external data service. The service then returns the supported object types, which includes the classes, workflow information, or item types for whose properties are to be updated with the values that are obtained from the external data source.

"Example: POST method request and response" on page 48
This sample code submits a request to an external data service when an IBM Content Navigator user selects a value for the region property, **`Region`**. The service then updates the choice list for the branch office property, **`BranchOffice`**, which depends on the **`Region`** property.

External data service REST protocol specifications

Understand the specifications for the object type request, the particular object type request, and the responses that you need to create for each request.

"Object types resource"
The object type resource represents the classes, item types, workflow step processors, or workflow in-basket filters to get for the external data service.

"Particular object type resource" on page 39
The particular object type resource represents the properties for which property values are obtained from an external data source. When the user creates a new item or modifies an item of the particular object type, the EDS REST protocol uses the particular object type resource to obtain data for the corresponding class, item type, workflow step processors, or workflow in-basket filters from the external data source.

Object types resource

The object type resource represents the classes, item types, workflow step processors, or workflow in-basket filters to get for the external data service.

“GET method”
The GET request gets the list of object types for the external data service.

“External data service integration into FileNet P8 workflows” on page 38
The external data service (EDS) can be implemented in data fields for process in-basket filter criteria, in step processors, and in launching workflows. You must construct a multi-part String that finds the service and identifies the in-basket filter, step, or workflow.

“Example: GET method request and response” on page 39
This sample code submits a GET request to an external data service. The service then returns the supported object types, which includes the classes, workflow information, or item types for whose properties are to be updated with the values that are obtained from the external data source.

GET method

The GET request gets the list of object types for the external data service.

URI syntax

For the response, you create the object types resource, for example, an ObjectTypes.json file, which contains the class names, item type names, or workflow information for whose properties are to be updated with the values that are obtained from the external data source.

For IBM FileNet P8 workflow information, multi-part String values are used instead of a class or item type name.

```
/types?repositoryId=Repository_ID
```

The repository ID is the symbolic name of the data source that contains the class names, item type names, or workflow information that is used in the service.

Request content

```
GET /types?repositoryId=Repository_ID
```

Response content

The response to the request must include a JSON payload that contains the following parameters:

```
{
 "types":
  [
   {
    "symbolicName": "TypeOne",
   },
   //More object types
 ...
  ]
}
```

Response codes

Table 13. Response codes for the GET method

Code	Description
200 OK	The method completed successfully. The response that is returned by the GET method includes the JSON payload that contains the list of object types.
400 Bad Request	One of the required parameters was missing or a parameter value was invalid.

Table 13. Response codes for the `GET` method (continued)

Code	Description
404 Not Found	Class names or item type names were not found in the **`repositoryId`** data source. This error indicates that the external data service does not manage any property values for the class because the object types request is not implemented by this external data service. The EDS REST protocol does not return an error to IBM Content Navigator. IBM Content Navigator ignores the external data service and does not call the service for the particular object type for any class.
500 Internal Server Error	A server error occurred. For information about the error, see the userMessage element in the JSON response.

Related reference:

"External data service integration into FileNet P8 workflows"
The external data service (EDS) can be implemented in data fields for process in-basket filter criteria, in step processors, and in launching workflows. You must construct a multi-part String that finds the service and identifies the in-basket filter, step, or workflow.

External data service integration into FileNet P8 workflows

The external data service (EDS) can be implemented in data fields for process in-basket filter criteria, in step processors, and in launching workflows. You must construct a multi-part String that finds the service and identifies the in-basket filter, step, or workflow.

You can implement an EDS in workflow data fields for the following areas:
- Process in-basket filter criteria
- Launch workflow and step processor properties

The EDS must register multiple-part values to identify the service request type. For workflow information, the multiple-part values are used instead of the class or item type name.

Process in-basket filter criteria

Construct the multiple-part string to identify the in-basket filter that you are using in your EDS implementation. Use the following format to identify the in-basket filter:

```
application_space_name.process_role_name.in-basket_name
```

Example:

```
{"symbolicName": "DefaultApplication.Personal_Items.My_Personal_Work"}
```

Launch workflow and step processor

Construct the multiple-part string to identify the launch workflow step or other steps in the step processor that you are using in your EDS implementation. Use the following format to identify the step:

```
workflow_name.sheet_name.step_name
```

Example:

```
{"symbolicName": "Data_Field_Workflow.Workflow.LaunchStep"}
```

The workflow registry supports an optional fourth part to the tokenized, multiple-part string:

workflow_name.sheet_name.step_name.EDS_Service_value

The *EDS_Service_value* value comes from the value of an exposed custom data field that is called *EDS_Service* on that particular step. The workflow designer must manually create the *EDS_Service* data field and specify for that data field to be exposed for steps that require the *EDS_Service* value for EDS processing. When a workflow exposes this string on a step, the string is not displayed in the processors properties user interface; instead, the string is a hidden property.

For example, this string identifies a step called `Approval_Step` with the *EDS_Service* value of `MyService`:

```
{"symbolicName": "Data_Field_Workflow.Workflow.Approval_Step.MyService"}
```

Example: GET method request and response

This sample code submits a `GET` request to an external data service. The service then returns the supported object types, which includes the classes, workflow information, or item types for whose properties are to be updated with the values that are obtained from the external data source.

Request example

```
GET /types?repositoryId=Repository_ID
```

Response example

```
[
 {"SymbolicName": "Book"},
 {"SymbolicName": "Invoice"},
 {"SymbolicName": "Folder"}
 {"SymbolicName": "Document"}
]
```

Particular object type resource

The particular object type resource represents the properties for which property values are obtained from an external data source. When the user creates a new item or modifies an item of the particular object type, the EDS REST protocol uses the particular object type resource to obtain data for the corresponding class, item type, workflow step processors, or workflow in-basket filters from the external data source.

The EDS REST protocol calls the POST method for resources when an item is created or a property is modified. The service then returns the required information in the response to the `POST` request.

"POST method" on page 40
The `POST` method obtains data from an external data source for item types of a specific class. You do not call this method directly. Instead, the EDS REST protocol calls this method automatically when workflow information or a content item is being added or modified.

"Request modes" on page 42
When an item is created or modified in the web client, IBM Content Navigator calls the `POST` method for the particular object type resource to submit a request to the external data service. This request payload contains a request mode that indicates the action that is being performed.

"Response payload to a POST method request payload" on page 43
The external data service responds to a `POST` method that was submitted by the `EDSSupportPlugin` plug-in. The response payload contains values for the properties that are managed by the service.

"Error responses for an external data service" on page 47
If the POST method call fails, the response code that the external data service returns indicates the type of error that occurred. Write the error responses for your external data service.

"Example: POST method request and response" on page 48
This sample code submits a request to an external data service when an IBM Content Navigator user selects a value for the region property, **Region**. The service then updates the choice list for the branch office property, **BranchOffice**, which depends on the **Region** property.

POST method

The POST method obtains data from an external data source for item types of a specific class. You do not call this method directly. Instead, the EDS REST protocol calls this method automatically when workflow information or a content item is being added or modified.

When IBM Content Navigator calls the POST method, the request payload contains the current value for each property. The current value can be one of the following values:

- The default value
- The value persisted for the property in the repository
- The working value that the user entered for the property

The response payload that the external data service returns includes changes to the properties that it manages. The service can modify attributes of properties in addition to modifying property values.

IBM Content Navigator then merges these changes with the base data in the repository and returns the data to IBM Content Navigator.

URI syntax

/type/{*class name*}

Path elements

Table 14. Path elements for the POST method

Name	Type	Description
{object type name}	String	The symbolic name of the class name or item type name that defines the property that is being updated.

Request content

```
{
   "repositoryId":"target_object_store_name",
   "objectId"   : "GUID_or_PID_of_item_that_is_being_edited",
   "requestMode" : "request_context",
   "externalDataIdentifier" : "identifier_for_the_service",

   "properties":
   [
   {
      "symbolicName" : "property_name",
      "value"        : current_value,
   }

   // More properties ...

   ],
    "clientContext":
    {
       "Key1":"Value1",
       "Key2":"Value2"
    }

 }
```

Table 15. Request parameters for the POST method

Name	Type	Required?	Description
repositoryId	String	Yes	The symbolic name of the target external data store that contains the property data.
objectId	String	No	The globally unique identifier (GUID) or persistent identifier (PID) that identifies the item that is being edited.
requestMode	String	Yes	One of the following request modes that indicates the reason that the POST method is being called: • initialNewObject • initialExistingObject • inProgressChanges • finalNewObject • finalExistingObject See Request modes for information about the request modes.
externalData Identifier	String	Yes	A string that indicates the state of the data that was returned by the external data service. The request must include this identifier if the **requestMode** parameter is set to one of these values: • inProgressChanges • finalNewObject • finalExistingObject
properties	Array	Yes	An array that contains values for the properties that are defined for the class or item type. For each property, the request contains the symbolic name and the property value.
clientContext	Array	No	An array that contains a series of key value pairs that specify contextual information for a specific class or item type. This parameter is used to send information to an external data service when an IBM Content Navigator user begins to add a document, add a folder, use an entry template, or create a search.

Response codes

Table 16. Response codes for the POST method

Code	Description
200 OK	The method completed successfully. The response that is returned by the POST method includes the updated information for the property.
400 Bad Request	One of the required parameters was missing or a parameter value was invalid.
404 Not Found	The class that was specified in the request was not found. This error does not indicate that the class is invalid. Instead, it indicates that the external data service does not manage any property values for the class. The EDS REST protocol does not return an error to IBM Content Navigator.
500 Internal Server Error	A server error occurred. For information about the error, see the userMessage element in the JSON response.

Related reference:

"Request modes"
When an item is created or modified in the web client, IBM Content Navigator calls the `POST` method for the particular object type resource to submit a request to the external data service. This request payload contains a request mode that indicates the action that is being performed.

Request modes

When an item is created or modified in the web client, IBM Content Navigator calls the `POST` method for the particular object type resource to submit a request to the external data service. This request payload contains a request mode that indicates the action that is being performed.

You must configure the external data service to respond with the data that is required for that action. For example, if the request is to add a document, when the Add Document wizard is opened, the service needs to respond with the initial property values that are defined for the `Add Document` class.

The **`requestMode`** parameter indicates the action that is being performed in IBM Content Navigator. This action determines the response that is returned by the external data service.

For input payloads, the following values are provided on the context ID:

Table 17. Values provided on the context ID for input payloads

Value	Description
userid	The user ID for the user who is logged in to IBM Content Navigator.
locale	The locale of the browser that initiated this call to the external data service.
desktop	The desktop ID for the desktop that is in use.

The **`requestMode`** parameter can have the following values:

initialNewObject
Indicates that a new object is being created. For each property, the input payload contains the symbolic name and the default value that is defined in the repository.

initialExistingObject
Indicates that an existing object is being edited. For each property, the input payload that is passed to the service contains the symbolic name and the value that is currently stored in the repository.

The input payload also contains the **`objectId`** parameter that specifies the globally unique identifier (GUID) or persistent identifier (PID) of the content item that is being edited. The service can use the GUID to refer to the content item. However, remember that the values stored in the repository for the class can change. Therefore, the values that are provided in the input payload might not match the values that are currently stored in the repository for the class.

The input payload does not contain the **`externalDataIdentifier`** parameter. Instead, this parameter is set by the external data service and returned in the response payload. Subsequent requests made during the update of the class include the **externalDataIdentifier** parameter to indicate the current state of the data to the service.

inProgressChanges
Indicates that the external data service is being called in response to changes in one or more properties that have dependent properties. The input payload can contain the following information:

- The current working value for each property in the class or item type
- The **externalDataIdentifier** parameter, which indicates to the service the previous state of any properties that it updated
- For an existing class, the **objectId** parameter, which specifies the GUID or the PID of the content item that is being edited

The external data service responds to this request if the attributes or working value of any property that it manages changed. The service also responds to return a custom validation error.

finalNewObject
Before the object is persisted in the repository, this value indicates that the external data service is being called for the final time in the sequence of exchanges to create the object. After this call, the new object is created and the property values are persisted in the repository.

For each property, the input payload that is passed to the service contains the working values for all properties that are defined by the class.

finalExistingObject
Indicates that the external data service is being called for the final time in the sequence of exchanges to update an existing property. After this call, the updated property values are persisted in the repository.

For each property, the input payload that is passed to the service contains the working values for all properties that are defined by the class or item type.

Response payload to a POST method request payload

The external data service responds to a `POST` method that was submitted by the `EDSSupportPlugin` plug-in. The response payload contains values for the properties that are managed by the service.

Response content

The response to the request must include a JSON payload that contains the following parameters:

```
{
  "externalDataIdentifier": "opaque_identifier_meaningful_to_the_service",

  "properties":
  [
    {
      "symbolicName": "symbolic_name",
      "value": potential_new_value,
      "customValidationError": "Description of an invalid reason",
      "customInvalidItems": [0,3,4,8], // Invalid multi-value items
      "displayMode": "readonly_or_readwrite",
      "required": true_or_false,
      "hidden": true_or_false,
      "maxValue": overridden_maximum_value,
      "minValue": overridden_minimum_value,
      "maxLength": underlying_maximum_length,
          "choiceList":
          {
            "displayName": "display_name",
            "choices":
            [
              {
                "displayName": "name",
                "value": value
              },

              {
                "displayName": "name",
                "value": value
              },

              // More choices ...
            ]
          }

          "hasDependentProperties": true_or_false,

        }

        // More properties ...

      ]

    }
```

Table 18. Response parameters for the `POST` method

Name	Type	Required?	Description
externalData Identifier	String	No	The identifier provides contextual information to indicate the state of the data that the service is returning. You implement this parameter with values that are meaningful for your data source. Typically, the parameter references the specific configurations that define attributes other than the property value such as the minimum value, maximum value, or the choice list. These configurations can be selected dynamically based on other property values. To select configurations dynamically, the service can use the **externalDataIdentifier** parameter to determine that the configuration changed since the previous call. If you do not implement dynamic behavior or dependencies for any attributes, you might not need to capture the data state. In this situation, you might implement the parameter to return a fixed string value. If the external data service returns data that is dynamic, you must capture the data state. For example, the service might manage a property whose value or other attributes are determined by the value of another property. In this situation, you must implement the parameter to return a value that references the specific configuration that was used to determine the value or other attributes of the dependent property. The parameter must capture enough information to identify changes in property data when the **externalDataIdentifier** parameter is returned to the service in an **inProgressChanges** request. For example, assume that a list of conditions is used to select a set of configurations based on the working property values. The data that is captured in the **externalDataIdentifier** parameter might include the indexes of the matching conditions.
properties	Array	Yes	An array that contains values for the properties that are managed by the external data service. For each property, you can specify the symbolic name and the attributes, such as value, choice list, and maximum length.

Property attributes

The **Properties** parameter contains the following attributes for each property that is managed by the external data service. The external data service can determine many of these values dynamically so that the service can return a different value in each response.

Table 19. Attributes of properties in the response payload

Name	Type	Required?	Description
symbolicName	String	Yes	The symbolic name of the property. The name must match the symbolic name that was specified in the request payload.
value	Determined by setting in the class	No	The value of the property. The value that is set by the external data service must correspond to the data type that is specified for the property in the class. The external data service can dynamically determine the property value based on the values of another other property. If the service does not specify a value, the current working value for the property is unchanged.
custom Validation Error	String	No	A message that describes why a property value is invalid. You can configure the external data service to validate the current value of a property. If the value is invalid, the service can leave the value unchanged and return an error message in the **customValidationError** parameter. For example, the service might determine that an account number is invalid. However, you do not want the service to replace the account number. Instead, you can configure the service to return an error message in the **customValidationError** parameter. If this parameter is included in the response, the property value is deemed invalid. However, the absence of this attribute indicates only that the parameter passed the validation that is performed by the external data service. The value might still be invalid based on attributes that are not validated by the service. Values are validated by using regular expressions. For example, the following regular expression can be used to validate a simple email address: `"\\b[\\w\\.-]+@[\\w\\.-]+\\.\\w{2,4}\\b"` **Restriction:** You can use custom validation for properties in workflow properties, teamspaces, and entry template properties. You cannot use custom validation for object properties, reference attributes, read-only properties, hidden properties, or search criteria.
custom Invalid Items	Array of indexes	No	An array of indexes for a list of values for a multi-valued property. When the external data service validates a multi-valued property, it can return this parameter to indicate the specific values that are invalid. If a multi-valued property is invalid and this parameter is not set, the property value as a whole is considered invalid. This attribute is applicable only if the **customValidationError** parameter indicates that the property is invalid.

Table 19. Attributes of properties in the response payload (continued)

Name	Type	Required?	Description
displayMode	String	No	A string that specifies whether IBM Content Navigator is to display the property value as read-only. The external data service can set this parameter to one of the following values: **readonly** The user can view the property value but cannot modify it. **readwrite** The user can modify the property value. This setting is the default value. If the property value is set to readonly in the class, the external data service cannot make the value writable. In this situation, a value of readwrite is ignored.
format	String	No	An expression that describes the correct format for values to enter into the property. Uses **formatDescription** to provide a description for the **format** parameter.
format Description	String	No	The description that is displayed in a tooltip if the user enters a format that does not match the expression specified in the **format** parameter.
required	Boolean	No	A Boolean value that is set to true to indicate that a value is required for the property. The external data service can dynamically determine this setting based on the values of other properties. However, the service cannot override the **required** parameter if the **required** parameter is set to true in the class.
hidden	Boolean	No	A Boolean value that is set to true to indicate that the property is to be hidden in IBM Content Navigator. The external data service can dynamically determine this setting based on the values of other properties. If this parameter is not specified, the value specified in the class is used.
maxValue	Integer, float, or date-time	No	A number that indicates the maximum value of the property. The external data service can dynamically determine this setting based on the values of other properties. If a maximum value is specified for the property in the repository, the service cannot make the setting less restrictive. That is, the service can set the maximum only to a smaller value. It cannot increase the maximum value. For example, if the maximum value in the repository is 100, the service can set the value to 50, but not to 150.
minValue	Integer, float, or date-time	No	A number that indicates the minimum value of the property. The external data service can dynamically determine this setting based on the values of other properties. If a minimum value is specified for the property in the repository, the service cannot make the setting less restrictive. That is, the service can set the minimum only to a larger value. It cannot decrease the minimum value. For example, if the minimum value in the repository is 100, the service can set the value to 150, but not to 50.

Table 19. Attributes of properties in the response payload (continued)

Name	Type	Required?	Description
maxLength	Integer	No	A number that indicates the maximum length of characters in the property value. The external data service can dynamically determine this setting based on the values of other properties. If a maximum length is specified for the property in Content Engine, the service cannot make the setting less restrictive. That is, the service can set the maximum length only to a smaller value. It cannot increase the maximum length. For example, if the maximum length in Content Engine is 100, the service can set the value to 50, but not to 150.

choiceList	Object	No	An array that defines a list of choices for the property value.
			
			The external data service can specify a choice list only if one is not defined for the property in the repository. The service can determine the choices in the list dynamically based on the values of other properties.
			
			The **choiceList** value can contain a flat list of choices:
			`"choiceList":`
			`{`
			`  "displayName": "`*`display_name_for_the_choice_list`*`",`
			`  "choices":`
			`  [`
			`    {`
			`      "displayName": "`*`display_name_for_a_specific_choice`*`",`
			`      "value": `*`value`*
			`    },`
			
			`    {`
			`      "displayName": "`*`display_name_for_a_specific_choice`*`",`
			`      "value": `*`value`*
			`    },`
			
			`    // More choices ...`
			`  ]`
			`}`
			
			Important: Do not implement choice lists in the Content Engine definitions for the same properties that are associated with the choice lists that you provide by using the external data service.

hasDependent Properties	Boolean	No	A Boolean value that is set to true if other properties depend on the value of this property. When this parameter is set to true, the `POST` method is called to update the dependent properties based on the new value whenever this property is updated. By default, this parameter is set to false.

Error responses for an external data service

If the `POST` method call fails, the response code that the external data service returns indicates the type of error that occurred. Write the error responses for your external data service.

For example, the response code `404 Not Found` indicates that the method did not find a resource, such as the class or item type. The response code `400 Bad Request` indicates that a required parameter was not provided or that an incorrect value was specified for a parameter.

The JSON response that is returned by the method contains additional information about the error condition. The following example shows the format that the response uses to provide that information:

```
#Response
HTTP/1.1 404 Not Found
Content-Type: application/json;charset-UTF-8
{
  "userMessage":
  {
    "text":"The specified object type is not a valid object type.",
  }
  "underlyingDetails":
  {
    "causes":
    [
      "More detailed message 1",
      "More detailed message 2",
    ]
  }
}
```

Example: POST method request and response

This sample code submits a request to an external data service when an IBM Content Navigator user selects a value for the region property, **`Region`**. The service then updates the choice list for the branch office property, **`BranchOffice`**, which depends on the **`Region`** property.

Request example

```
POST /type/class_name_or_item_type_name

{
 "repositoryId":"ObjectStore1",
 "objectId" : "{EC4D244B-5980-4...6978-8796-A5B4C3D2E1F0}",
 "requestMode" : "inProgressChanges",
 "properties":
  [
   {
    "symbolicName" : "Region",
    "value" : "Western",
   }
  ],
 "clientContext":
  {
   "userid":"user1",
   "locale":"en_US",
   "desktop": "default"
  }
}
```

Response example

```
{
  "externalDataIdentifier": "opaque_identifier_meaningful_to_the_service",
  "properties": [
     {
       "symbolicName": "BranchOffice",
       "dependentOn": "Region",
       "dependentValue": "Western",
       "choiceList": {
          "displayName": "Western Branch Offices",
                "choices": [{
                   "displayName": "Los Angeles",
                   "value": "Los Angeles"
                },
                {
                   "displayName": "San Francisco",
                   "value": "San Francisco"
                },
                {
                   "displayName": "Salt Lake City",
                   "value": "Salt Lake City"
                },
                {
                   "displayName": "Seattle",
                   "value": "Seattle"
                }]
             }
          }
       ]
    }
```

Sample external data service

In the sample external data service, the values for a choice list are used from an external data source.

A sample external data service and JSON files, which are the sample data source, are available in the `*ECMClient_installdir*\samples\sampleEDSService\WEB-INF\classes` directory.

"Files in the sample external data service"
You can use the sample service as an example and as a starting point for your own external data service. The sample service consists of a `GetObjectTypes` servlet, an `UpdateObjectTypes` servlet, the object types resource JSON file, JSON files as the data source, and the `web.xml` deployment file.

"Deploying the sample external data service" on page 50
You modify the sample external data service to be appropriate for your content classes and then deploy the service in the web application server.

Related reference:

"Sample applications for IBM Content Navigator" on page 7
The IBM Content Navigator software package includes several sample applications: a plug-in, an external data service, and a set of web pages. You can use these sample applications to create custom applications.

Files in the sample external data service

You can use the sample service as an example and as a starting point for your own external data service. The sample service consists of a `GetObjectTypes` servlet, an `UpdateObjectTypes` servlet, the object types resource JSON file, JSON files as the data source, and the `web.xml` deployment file.

The sample service consists of the following files:

GetObjectTypesServlet.java servlet
This servlet gets the list of all classes to handle in the external data service.

UpdateObjectTypeServlet.java servlet
For each class, this servlet gets the current attributes or values and returns that information in the response payload.

ObjectTypes.json file
This JSON file determines which classes you are using for the external data service.

JSON files as the data source
These JSON files are the external data sources for the sample EDS, and can be used to understand the required format of the JSON. The `ECMClient_installdir\samples\sampleEDSService\WEB-INF\classes` directory contains the following sample JSON data source files:

- `Article`
- `Book`
- `Document`
- `Email`
- `Folder`
- `Invoice`
- `NOINDEX`
- `DefaultApplication*`
- `Data_Field_*`

Important: These files are provided primarily to help you understand the expected JSON format and to help you start creating the JSON for your own external data service. To use the sample files for EDS deployment, extensive configuration is required.
The JSON needs to match the classes, item types, and the corresponding custom properties on your system. To use the provided JSON files in the sample, the JSON files need to be modified so that the correct custom properties are aligned with the actual classes and custom properties in your IBM FileNet P8 or IBM Content Manager repository.
Similarly, to use the workflow JSON files (IBM FileNet P8), the JSON must be modified to match your IBM Content Navigator workflows, including the appropriate steps, step names, workflow data fields, and other custom properties.
To apply the sample EDS on a workflow filter, the `Application Space`, `Role`, `Inbasket`, and filter criteria must be set up accordingly.
A sample localized file is included in the sample to help illustrate that the EDS can send locale information in the request. The EDS developer can specify for the EDS to look at the locale information in the request and return the response JSON in the proper correct translation.

web.xml file
The `web.xml` file provides configuration and deployment information for the web application. This file must reside in the `WEB-INF` directory under the context of the hierarchy of directories that exist for the external data service.

Deploying the sample external data service

You modify the sample external data service to be appropriate for your content classes and then deploy the service in the web application server.

About this task

An Eclipse project is provided with the sample EDS service. You must modify the libraries that are referenced in the project to fit your configuration. For example, the `\navigator\build\lib\j2ee\j2ee.jar` might not be the correct path to your j2ee.jar file. Find the equivalent JAR file for your environment and update the project configuration to point to that JAR file.

Procedure

To deploy the sample external data service:

1. Copy the sample external data service from the installation directory into a dynamic web project in Rational® Application Developer.
2. Modify the JSON files in the sample so that the correct custom properties are aligned with the actual classes and custom properties in your IBM FileNet P8 or IBM Content Manager repository.
3. Compile the project and build a WAR file. You can use the build.xml file that is included in the sample as a starting point.
4. Deploy the WAR file into your web application server.
5. Register and configure the external data service support plug-in, which is called edsPlugin, to use the sample external data service:
   a. Load the edsPlugin.jar file. In the IBM Content Navigator administration tool, in the configuration options for the plug-in, define the URL or the file path to the edsPlugin.jar file:
      - Enter the URL in the following sample format: http://*host_name*:*port_number*/*ECMClient_installdir*/plugins/edsPlugin.jar.
      - Enter the path to the file that is installed on the server, for example:
        – Microsoft Windows: C:\Program Files\IBM\ECMClient\plugins\edsPlugin.jar
        – Linux or AIX®: /opt/IBM/ECMClient/plugins/edsPlugin.jar

      The ECMClient folder is the IBM Content Navigator installation directory.
   b. After the edsPlugin.jar file is loaded, you must specify the URL to where the EDS is deployed. For example, you can specify: http://host_name:port_number/sampleEDSService.
   c. Click **Save** and **Close**.
   d. Refresh your web browser.

Results

When you select values from choice list properties, the list contains the values from your external data source, which is specified by the external data service.

What to do next

In the IBM Content Navigator client, verify that the fields and properties are showing the correct data and are behaving as you expect.

IBM Content Navigator API reference

IBM Content Navigator provides JavaScript and Java application programming interfaces (APIs) so that you can create custom applications. The JavaScript API includes the classes for the visual widget library and the modeling library. The Java API includes the classes that you can use to create plug-ins to IBM Content Navigator.

For screen reader users on Mozilla Firefox web browsers, traverse through the landmark navigation area in the JavaScript API reference documentation by using the Tab key.

Important: The IBM Content Navigator APIs are subject to change without notice. If you use the APIs to develop applications or plug-ins, you might need to make updates to your code if a new version of the APIs are released.

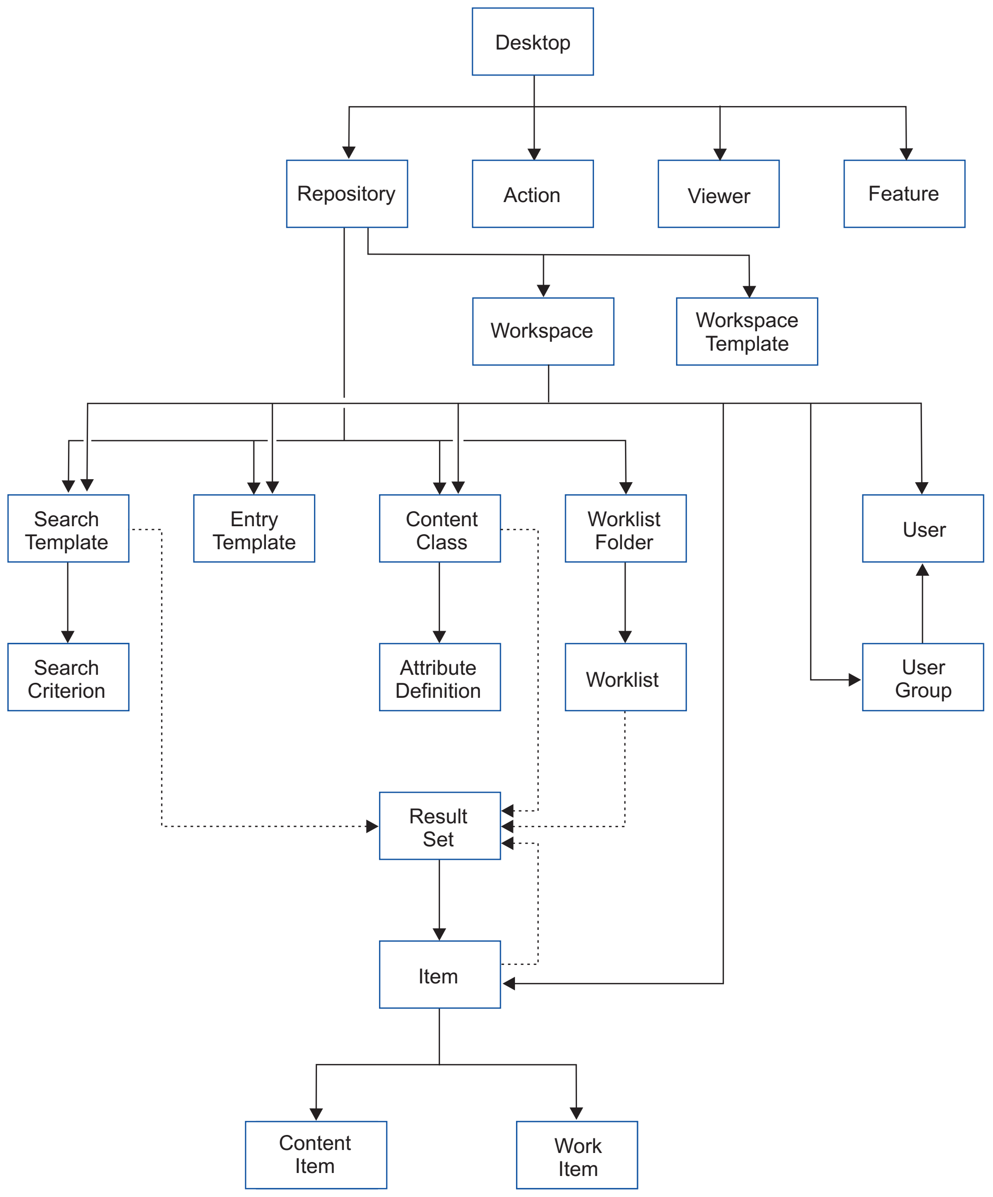
“IBM Content Navigator modeling library structure”
The classes in the modeling library provide the business logic and data for IBM Content Navigator. These classes are used by the widgets to access and represent data in the content servers and in the IBM Content Navigator configuration.

“IBM Content Navigator visual widget catalog” on page 56
The JavaScript classes in the IBM Content Navigator visual widget library are grouped by package. In some packages, the classes can be further grouped by the type of widgets that they represent.

IBM Content Navigator modeling library structure

The classes in the modeling library provide the business logic and data for IBM Content Navigator. These classes are used by the widgets to access and represent data in the content servers and in the IBM Content Navigator configuration.

The following diagram shows the hierarchy of the primary classes in the modeling library:



As shown in this diagram, an instance of the `Desktop` class encompasses the other objects in the model. The `Desktop` object specifies the repositories, features, actions, and viewers to which a set of users will have access.

An instance of the `Repository` class represents a specific repository. The repository can be an IBM Content Manager or IBM Content Manager OnDemand server, or an IBM FileNet Content Engine object store.

The `Repository` object gives users the ability to access and perform actions on repository objects. These objects are also represented by classes in the modeling library:

`SearchTemplate` class
: This class represents a search that is stored in the repository. In an IBM Content Manager or IBM FileNet P8 repository, a `SearchTemplate` object represents a saved search. In an IBM Content Manager OnDemand repository, a `SearchTemplate` object represents a folder.

A `SearchTemplate` object gives users the ability enter or modify the criteria that is used to perform a search. The criteria is represented in the model library by the `SearchCriterion` class.

`EntryTemplate` class
: This class represents an entry template that is stored in an IBM FileNet P8 repository. An `EntryTemplate` object gives users the ability to create a document, folder, or custom object. An `EntryTemplate` object also provides the default values for the destination folder, properties, and security.

`ContentClass` class
: This class represents a document or folder class in a IBM FileNet P8 repository or an item type in an IBM Content Manager repository. A `ContentClass` object gives users the ability to access the item and to edit the properties and attributes of the item.

  Each property or attribute is represented by an instance of the `AttributeDefinition` class that contains information about the property or attribute, such as type and allowed values.

`WorklistFolder` class
: This abstract class represents a collection of work lists, which are sometimes called in-baskets or inboxes. The subclasses of the `WorklistFolder` class represent collections of the different types of work lists and in-baskets. For example, the `ProcessApplicationSpace` class represents a collection of process roles that determine who can access an IBM FileNet P8 process application. The `ProcessRole` class represents a role that is defined on an IBM FileNet P8 server. An instance of this class determines who has access to the in-baskets that are defined in an application space.

`Worklist` class
: This class represents a single work list. A `Worklist` object provides users with the ability to process the work items that are assigned to them.

  The model includes the `ProcessInBasket` subclass for IBM FileNet P8 repositories.

`Workspace` class
: This class represents a teamspace. A `Workspace` object provides users with the ability to organize and share the content that a team needs to complete its tasks.

  The individual users and groups who belong to a teamspace are represented in the model by the `User` and `UserGroup` classes.

`WorkspaceTemplate` class
: This class represents a teamspace template. A `WorkspaceTemplate` object provides users with the ability a set of predefined options for creating a teamspace.

`ResultSet` class
: This class represents a set of items that are returned by a search. A `ResultSet` object provides users with the ability to locate and select documents, folders, or work items.

  An individual item in a `ResultSet` object is represented in the model by the `Item` class or one of its subclasses:

  - The `ContentItem` class represents a document, folder, or other content item in the repository.
  - The `WorkItem` class represents a workflow item.

Other classes in the modeling library support the classes that are shown in the diagram. For example, the model includes the following classes:

- The `SearchTemplateFolder` class and `WorkspaceFolder` class represent collections of search templates and teamspaces. These abstract classes provide collections for items such as such the recent folders or all folders that are displayed in navigation trees in the user interface.

  For an IBM Content Manager OnDemand repository, a `SearchTemplateFolder` object represents a cabinet.
- The `Request` class represents a request that is made to an IBM Content Navigator service.
- The `PropertyFormatter` class represents the formatting that is applied to properties when they are displayed. This class can be extended or overridden to provide custom formatting of certain types of properties.
- The `_ModelStore` class and the classes that are suffixed with *TreeModel* represent Dojo data stores and trees. These classes can be used with Dojo `dijit` classes to populate widgets with data.

Related concepts:

"IBM Content Navigator development architecture" on page 3
IBM Content Navigator uses a model-view-controller (MVC) architecture.

IBM Content Navigator visual widget catalog

The JavaScript classes in the IBM Content Navigator visual widget library are grouped by package. In some packages, the classes can be further grouped by the type of widgets that they represent.

"Widgets package" on page 57
The classes in the `ecm.widget` package define widgets that you use to create, edit, and search repository objects such as documents, folders, and work items. This package also includes classes that define common user interface components such as buttons, menus, and tabs.

"Administration widgets package" on page 67
The classes in the `ecm.widget.admin` package define the widgets that make up the IBM Content Navigator administration tool. You can use these widgets to customize the administration user interface.

"Administration model package" on page 70
The classes in the `ecm.widget.admin.model` package provide the model for the user interface of the IBM Content Navigator administration tool. You can use the widgets in this package to extend the capabilities of the IBM Content Navigator administration tool. Alternatively, you can use these widgets to provide a separate administration user interface that is based on the IBM Content Navigator user interface.

"Dialog box widgets package" on page 71
The ecm.widget.dialog package contains classes that define the dialog boxes that are used in the IBM Content Navigator web client.

"Layout widgets package" on page 74
The `ecm.widget.layout` package contains classes that define components of the web client layout.

"Process widgets package" on page 76
The `ecm.widget.process` package contains classes that are used to build custom launch processors and step processors for an IBM FileNet P8 workflow.

"Teamspace builder widgets package" on page 76
The `ecm.widget.workspaceBuilder` package contains the widgets that create and

edit teamspaces and teamspace templates. In IBM Content Navigator, these widgets are used for the teamspace builder.

"Viewer widgets package" on page 78
The `ecm.widget.viewer` package contains classes that are used to view document content.

Related concepts:

"IBM Content Navigator development architecture" on page 3
IBM Content Navigator uses a model-view-controller (MVC) architecture.

Widgets package

The classes in the `ecm.widget` package define widgets that you use to create, edit, and search repository objects such as documents, folders, and work items. This package also includes classes that define common user interface components such as buttons, menus, and tabs.

Document and folder widgets

These widgets are used to view and edit documents and folders.

Table 20. Widgets that are used to view and edit documents and folders

Widget class	Description
AddContentItemGeneralPane	Provides a pane that is used in the AddContentItemDialog widget to add a document or folder to a repository. This pane is also used in the CheckInDialog widget to check a document in to a repository.
AddContentItemPropertiesPane	Provides a pane that is used in the AddContentItemDialog widget to edit properties when a user adds a document or folder to the repository. This pane is also used in the CheckInDialog widget to edit properties when a user checks a document in to the repository.
AddContentItemSecurityPane	Provides a pane that is used in the AddContentItemDialog widget to view and edit the security settings of a document or folder that is being added to a repository. This pane is also used in the CheckInDialog widget to view and edit the security settings of a document or folder that is being checked in to a repository.
AttributeDefinitionsForm	Provides a widget that is used to define attributes for an IBM Content Manager class or item type.
ChildComponentPropertiesPane	Provides a widget that is used to view and edit the attributes of a child component of an IBM Content Manager item type. A child item type can have unique attributes. For example, IBM Content Collector uses child components to store email instance information and references to attributes of the distinct email item type.

Table 20. Widgets that are used to view and edit documents and folders (continued)

Widget class	**Description**
ChildComponentSelector	Provides a widget that is used to display the child components of an IBM Content Manager item type.
CommonPropertiesPane	Provides a widget that is used to view and edit properties. This widget can be used in dialog boxes for adding documents, creating folders, and editing properties.
ContentClassSelector	Provides a widget that contains a tree or drop-down list that is used to select the class for a document or folder.
ContentList	Provides a widget that displays the results of a search, the contents of a folder, or the work items in a work list. This widget can be used to navigate folders and launch actions for the items that are displayed.
ContentListDropDownDialog	Provides a widget that contains a drop-down list that is used in a content list.
ContentListToolbar	Provides a widget that is used as a toolbar for the ContentList widget.
FolderSelector	Provides a widget that is used to select folders in a repository, workspace, or parent folder.
FoldersFiledInPane	Provides a widget that shows the folders in which a document is filed.
FolderTree	Provides a widget that is used to browse the folders in a repository.
ItemEditPane	Provides a widget that is used to edit a folder or a document in a repository.
ItemPreviewPane	Provides a widget that displays a preview of the content for an item.
ItemPropertiesDisplayPane	Provides a widget the displays a read-only view of the properties for an item.
ItemPropertiesPane	Provides a widget that is used to edit the properties of an item.
ItemPropertiesStackPane	Provides a pane that contains a stack of ItemPropertiesPane widgets. Typically, the ItemPropertiesStackPane contains a single ItemPropertiesPane widgets. However, for reference attributes, a separate ItemPropertiesPane widget is added to the stack for each reference attribute the user opens. The user sees the top widget in the stack.
ItemSecurityPane	Provides a widget that contains the SecurityPane widget and is used by the Edit Properties dialog. This class retrieves the item permissions and other item information that is required by the SecurityPane widget to display the security information.

Table 20. Widgets that are used to view and edit documents and folders (continued)

Widget class	Description
ModifyPermissionPane	Provides a pane that is used to edit the permissions for accessing a document or folder.
SecurityPane	Provides a widget that contains a list of users and roles that have access to an item.
VersionsPane	Provides a widget that is used to view the versions of an item.

Search widgets

These widgets provide the mechanisms for searching a repository.

Table 21. Widgets that are used to search repositories

Widget class	Description
AttributeDefinitionWidget	Provides a widget that is used by the search builder to represent an attribute in a search template.
BasicSearchBuilder	Provides a widget that is used for the search builder interface. The search builder is used for creating and editing searches and for performing ad hoc searches.
BasicSearchDefinition	Provides a widget that is used by the BasicSearchBuilder widget to define and save the search criteria.
ClassCriteria	Provides a widget that is used to display the item class on which a saved search is defined to search. For a saved search that was defined in IBM FileNet Workplace XT to support searching multiple item classes, this widget includes a link to open a drop-down list of the item classes from which users can select.
SearchBuilder	Provides a wrapper widget that contains the BasicSearchBuilder widget. This widget handles relationships with other widgets such as maintaining opened searches and closing the widget when a search is canceled. This widget also locks or unlocks a saved search when the search is opened or closed.
SearchForm	Provides a widget that is used to render a search form for a saved search. Users with the appropriate access rights can use this widget to edit a saved search.
SearchInCriteria	Provides a widget that displays the folder or repository to be searched by a saved search. For a saved search that was defined in IBM FileNet Workplace XT to support searching multiple folders or repositories, this widget includes a link to open a drop-down list of the folders or repositories from which users can select.

Table 21. Widgets that are used to search repositories (continued)

Widget class	Description
SearchMoreOptions	Provides a widget that displays additional options for a search. The additional options include the object type, version, and join relationship of the property, and text conditions.
SearchPropertyOptions	Provides a widget that contains property search criteria in a drop-down dialog box. The dialog box can be defined to return results that match all conditions or any condition.
SearchResultsDisplayOptions	Provides a widget that contains the columns that are used to display search results. The search results can be sorted by a property in descending or ascending order. If the text search is enabled, the search results can be sorted by rank in descending order.
SearchSelector	Provides a widget that contains a tree or drop-down list that is used to select a search template.
SearchTab	Provides a widget that contains a SearchForm widget and a ContentList widget with the widget events wired together.
SearchTabContainer	Provides a widget that contains and controls the search tabs that are open.
TextSearchOptions	Provides a widget that is used to enter text search options in an IBM FileNet P8 repository. The search options include the text search operator that is to be used. Users can show or hide the text search section to enable or disable the text search feature.

Teamspace and user widgets

These widgets support the definition of teamspaces and the selection of users.

Table 22. Widgets that are used to define teamspaces and select users

Widget class	Description
TeamList	Provides a widget that displays a list of the users and groups that are assigned to a teamspace.
UserGroupSelector	Provides a widget that is used to select a user or group. This widget can be configured to display both users and groups or only users. The widget can also be configured to support multiple selections or only single selections.
UserGroupSelectorPane	Provides a pane that is used by the `_PropertiesMixin` class to select users or groups for assigning property value. The widget can be configured to display voting approval properties for Entry Template workflows.

User interface widgets

These widgets provide common user interface components.

Table 23. Widgets that are used to define common user interface components

Widget class	Description
_HoverHelpMixin	Provides a base class for creating and displaying hover help in the user interface.
_MoveUpDownMixin	Provides a base class for a widget that is used to move items up or down in a list.
_PropertiesMixin	Provides a base class that contains methods for working with property metadata. This class provides a function is used to create form fields for the properties and to hold the values in a private array variable.
_RepositoryDepWidgetBase	Provides a base class for all widgets that change based on the repository that is being used. This class provides common properties and connections for the widgets that inherit from it. For example, this class is used for a widget that must load a different toolbar based on whether the user is using an IBM FileNet Content Manager repository or an IBM Content Manager repository.
_SecurityMixin	Provides a base class that contains methods for working with IBM FileNet P8 security data.
_TilesListItemFactory	Provides a class that is used to generate entries for the ecm.widget.TilesList widget.
ActionMenu	Provides a widget that is used as a context menu.
Banner	Provides a banner that is displayed in the user interface for an IBM Content Navigator application.
BookmarkPane	Provides a widget that is provides a URL-addressable page that can display a bookmarked folder. The URL for the page can be created from the **View Link** action or the **Email as Link** action for a folder.
Breadcrumb	Provides a widget that displays the position of a user in the IBM Content Navigator application.
Button	Provides a widget that is used as a button in the user interface. This class extends the Dojo `dijit.form.Button` class to provide hover help for the button.

Table 23. Widgets that are used to define common user interface components (continued)

Widget class	Description
CheckBox	Provides a check box that can be used to select several values for the same property. This class extends the Dojo `dijit.form.CheckBox` class to provide hover help for the check box.
ComboBox	Provides a widget that displays a combination box that consists of a field and a drop-down list. The user can select a value from the list or type a value in the field. The value in the field does not need to match a value in the list. This class extends the Dojo `dijit.form.ComboBox` class to provide hover help for the combination box.
CompositeButton	This widget provides a button that can be configured to display a button icon on the left of the container, a button title in the middle, or an action button icon on the right. This button is used by the SecurityPane widget to display item security entries.
DatePicker	Provides a widget that is used to select a date. This class extends the `idx.form.DatePicker` class to provide additional formatting patterns.
DesktopPane	Provides a widget that is used to display a desktop.
DropDownDialog	Provides a widget that is used to display a drop-down list.
DropDownInput	Provides a widget that combines a drop-down list and a field. A user can enter a value in the field or select a value from the drop-down list.
DropDownLink	Provides a widget that contains a link with a drop-down menu and summary.
Ellipsis	Provides a widget that displays an ellipsis (...) to indicate that data is truncated so the user is not seeing all the values. The widget can be configured to truncate data on the left or on the right. By default, data is truncated on the right.
FavoritesTree	Provides a widget that contains a tree that is used to display the user's favorites.

Table 23. Widgets that are used to define common user interface components (continued)

Widget class	Description
FilteringSelect	Provides a combination box that consists of a drop-down list and a field. The user can select a value from the list or type a value in the field. The list is filtered to display values that begin with the value entered in the field. A valid entry must match completely a value from the list. This class extends the Dojo `dijit.form.FilteringSelect` class to support hover help.
FilterTextBox	Provides a widget that is used to filter the input in a text box.
GlobalToolbar	Provides a widget that is used as a global toolbar for an IBM Content Navigator application.
HoverHelp	Provides a widget that is used to display the question mark icon that links to help in the user interface.
HoverHelpText	Provides a widget that is used to display hover help and text in a grid cell in the user interface.
InbasketContainer	Provides a widget that is used to display the in-baskets that are associated with an IBM FileNet Process Engine role. Each in-basket is displayed in a ContentList widget that is contained in the InbasketTabContainer widget.
InbasketFilterContainer	Provides a widget that is used to display the query filters that are specified on the in-basket content. The filters can be used to refine the query for the items that are displayed in the in-basket.
InbasketFiltersPane	Provides a widget that is used to display the Filter and Reset buttons for working with the in-basket filters. In addition, this widget displays the number of filters that are currently in effect.
LoginPane	Provides a widget that is used to log in to a content management server. This widget prompts the user for credentials to authenticate to the server.
MessageBar	Provides a widget that is used as a message bar in an IBM Content Navigator application.
MultiColumnList	Provides a simple multiple-column list that is used to select items.
MultiValueChoicePane	Provides a widget that is used to select multiple values from a choice list. This widget is used for IBM FileNet Content Manager repositories.

Table 23. Widgets that are used to define common user interface components (continued)

Widget class	Description
MultiValueInputPane	Provides a widget that is used to enter multiple values. This widget is used for IBM FileNet Content Manager repositories.
NumberTextBox	Provides a widget that contains a text box that is used to enter a numeric value. This class extends the Dojo `dijit.form.NumberTextBox` class to support hover help.
RadioButton	Provides a widget that contains a set of choices from which only one can be selected. This class extends the Dojo `dijit.form.RadioButton` class to support hover help.
RangeBoundTextBox	Provides a widget that contains a text box for which a range of valid values is specified. This class extends the Dojo `dijit.form.RangeBoundTextBox` class to support range checking for large exponents.
ReferenceAttributeButton	Provides a widget that is used as a button to display a the object identified by a reference attribute in an IBM Content Manager repository.
RepositorySelector	Provides a widget that is used to select a repository.
Select	Provides a widget that is used as a button that displays a drop-down list when it is clicked. This class extends the Dojo `dijit.form.DropDownButton` to provide hover help for the button.
SelectObject	Provides a widget that is used to select a repository object by entering a query.
ShowHyperlinkPane	Provides a widget that is used to display the hyperlink for a document or folder.
SingleSelectTree	Provides a widget that contains a tree in which the user can select one node in the tree at a time.
SingleValueTreePane	Provides a widget that contains a tree in which the user can select one value at a time. This widget is used for IBM FileNet Content Manager object stores.

Table 23. Widgets that are used to define common user interface components (continued)

Widget class	Description
SloshBucket	Provides a widget that is used to pick a subset of unique items from a larger set of items. The widget contains two single column lists. One list contains the available items and the other list contains the selected items. The **Add** button and a **Remove** button that are used to move items from one list to the other appear between the columns. Optionally, the widget includes an **Up** button and a **Down** button that are used to order the selected items.
StateSelect	Provides a widget that contains a drop-down selection box. This class extends the Dojo `dijit.form.Select` class to support options that can be set programmatically for the selection box, but that are hidden from users.
TextBox	Provides a widget that contains a text box. This class extends the Dojo `dijit.form.TextBox` class to provide hover help for the text box.
TilesList	Provides a widget that displays content in a tiles list view, which is similar to the tiles view in Microsoft Windows Explorer. In IBM Content Navigator, this widget lists of teamspaces and teamspace templates; however, it can be used to display any type of content.
TilesListItem	Provides a widget that contains a single row item that is displayed in the TilesList widget.
TimePicker	Provides a widget that is used to select a time. This class extends the `idx.form.TimePicker` class to provide additional formatting patterns.
TitlePane	Provides a widget that contains a content box with a title. This widget can be opened or collapsed. This class extends the `idx.layout.TitlePane` class to provide hover help for the widget.
TreeDndSource	Provides a widget that contains a tree that is used to navigate the folders in a repository. This widget extends the Dojo `dijit.tree.dndSource` class to prevent users from using the drag-and-drop feature to move the **More** node. (The **More** node is shown in a tree when more results exist. The additional results are displayed with the user clicks this node.)

Table 23. Widgets that are used to define common user interface components (continued)

Widget class	Description
TreeSelector	Provides a widget that contains a tree that is used to navigate the folders in a repository. This widget allows users to select multiple folders, but it does not allow them to select nodes with no item value.
TreeSloshBucket	Provides a widget that contains a slosh bucket that uses a tree to display available and selected items.
ValidationTextBox	Provides a widget that contains a text box in which the user input is validated against the specified criterion. This class extends the Dojo `dijit.form.ValidationTextBox` class to add validation for byte maximum length.

Teamspace widgets

These widgets are used to view and edit teamspaces.

Table 24. Widgets that are used to view and edit teamspaces

Widget class	Description
WorkspacePropertiesPane	Provides a widget that is used to view or edit the properties of a teamspace template.
WorkspaceSelector	Provides widget that contains either a tree or a drop-down list that is used to select a teamspace.

Workflow widgets

These widgets are used to view and edit workflows, work lists, and work items.

Table 25. Widgets that are used to view and edit workflows, work lists, and work items

Widget class	Description
ReassignToUserDialog	Provides a dialog box that is used to reassign an IBM FileNet P8 workflow item to another user.
StartWorkflow	Provides a widget that is used to start folders and documents on a workflow.
SuspendWorkItems	Provides a widget that is used to suspend folders and documents on a workflow.
TrackerHistoryPane	Provides a widget that is used to view the history of an IBM FileNet P8 workflow.
TrackerMilestonesPane	Provides a pane that is used to view the milestones for a tracker.
WorkItemAttachmentsPane	Provides a widget that is used to view or edit work item attachments.
WorkItemPropertiesPane	Provides a widget that is used to view or edit the properties of a work item.

Table 25. Widgets that are used to view and edit workflows, work lists, and work items (continued)

Widget class	Description
WorklistSelector	Provides a widget that contains a tree that is used to navigate process workflow items. This widget displays the available work lists, process roles, and in-baskets for selection.

Administration widgets package

The classes in the `ecm.widget.admin` package define the widgets that make up the IBM Content Navigator administration tool. You can use these widgets to customize the administration user interface.

General widgets

These widgets define the general layout of the administration tool.

Table 26. Widgets that define the general layout of the administration tool

Widget class	Description
ActionMenu	Provides a context menu that is used in the IBM Content Navigator administration tool.
AdminGrid	Provides a scrollable, tabular grid that is used to display administration objects.
AdminLoginDialog	Provides a dialog box that is used to log in to a selected repository from the IBM Content Navigator administration tool.
AdminTabs	Provides a container for the IBM Content Navigator administration tool that has multiple panes with a tab that corresponds to each pane.
AdminTree	Provides a navigation tree for administration objects such as desktops, labels, menus, plug-ins, repositories, settings, and viewer maps.
IconMappingDialog	Provides a dialog box that is used to map icons to MIME types.
IconStatusDialog	Provides a dialog box that is used to edit the icons that are mapped to the document states.
NexusAdminPane	Provides the administration view for the application layout. This view is used for administration and configuration tasks.
ViewerMappingDialog	Provides a dialog box to create or edit a viewer map.

Desktop configuration widgets

These widgets are used to configure the desktops that are available in the web client.

Table 27. Widgets that are used to configure desktops

Widget class	Description
DesktopAppearance	Provides a pane that is used to customize the appearance of the desktop and to specify which features users can access from this desktop.
DesktopMenus	Provides a pane that is used to specify the menus that are to be accessible from a desktop.
DesktopRepositories	Provides a pane that is used to specify the repositories that are to be accessible from a desktop.
DesktopWorkflows	Provides a pane that is used to specify the IBM FileNet P8 application spaces that are displayed in a specific desktop.
TabDesktop	Provides a widget that is used to create or edit a desktop. The widget displays the desktop properties in a tabbed pane.
TabDesktops	Provides a widget that lists all available desktops. The widget displays the list in a tabbed pane.

Label configuration widgets

This widget is used to customize labels that are used in the web client.

Table 28. Widgets that are used to customize labels

Widget class	Description
TabInterfaceText	Provides a widget that is used to customize the text of labels that are displayed in specific areas of the web client.

Menu configuration widgets

These widgets are used to configure the context menus and toolbars that are available in the web client.

Table 29. Widgets that are used to configure menus

Widget class	Description
TabMenu	Provides a tab that is used to edit a toolbar or a context menu.
TabMenus	Provides a tab that lists the available toolbars and context menus. This tab is used to select a toolbar or a menu to edit or copy.

Plug-in configuration widgets

These widgets support configuration of plug-ins.

Table 30. Widgets that are used to configure plug-ins

Widget class	Description
PluginConfigurationPane	Provides a base class that can be extended to create a configuration interface for a plug-in.
TabPlugin	Provides a tab that is used to add a plug-in or edit the information for a plug-in.
TabPlugins	Provides a tab that shows the available plug-ins. This tab is used to select a plug-in to edit or delete.

Repository configuration widgets

These widgets are used to configure the repositories that IBM Content Navigator connects to.

Table 31. Widgets that are used to configure repositories

Widget class	Description
ODCustomPropertiesDialog	Provides a dialog box that is used to edit custom properties for the connection to an IBM Content Manager OnDemand repository.
RepositoryConfigurationParameters	Provides a widget that is used to configure the properties for a repository.
RepositoryFolders	Provides a pane that is used to specify the properties for repository folders. The pane also determines the order in which the folders are displayed when users browse a repository.
RepositorySearch	Provides a pane that is used to configure the search properties, search operators, and search results for a repository.
RepositorySystemProperties	Provides a pane that is used to specify the system properties that are displayed for the documents and folders in a repository.
TabRepositories	Provides a tab that shows the available repositories.
TabRepository	Provides a tab that is used to view and configure a repository. Users use this tab to edit repository parameters, system properties, and folders.

Settings widgets

These widgets support general configuration tasks.

Table 32. Widgets that are used to configure settings

Widget class	Description
SettingAdminUsers	Provides a widget that is used to specify the users that have administrative privileges.

Table 32. Widgets that are used to configure settings (continued)

Widget class	Description
SettingIconMapping	Provides a widget that is used view and edit the mappings of icons to the MIME types and states that they represent.
SettingLogging	Provides a widget that is used to customize the logging that is performed by the web client.
SettingMimeTypeIconMapping	Provides a widget that is used to map a list of MIME types to a predefined icon that is provided with the web client or to a custom icon.
SettingOnDemand	Provides a widget that is used to configure the settings that are used by the OnDemand Web Enablement Kit.
SettingStatusIconMapping	Provides a widget that is used to customize the icons that are displayed next to the documents, folders, and work items on your repositories.
TabSettings	Provides a widget that contains the widgets that are used to set values that apply to all of the desktops in a configuration. The widgets are displayed in a tabbed pane.

Viewer mapping widgets

These widgets support the mapping of content viewers to specific file types.

Table 33. Widgets that are used to configure viewer mappings

Widget class	Description
TabViewer	Provides a tab that is used to add or edit a viewer that is used to open files.
TabViewers	Provides a tab that lists the viewers that are available to open files.

Administration model package

The classes in the `ecm.widget.admin.model` package provide the model for the user interface of the IBM Content Navigator administration tool. You can use the widgets in this package to extend the capabilities of the IBM Content Navigator administration tool. Alternatively, you can use these widgets to provide a separate administration user interface that is based on the IBM Content Navigator user interface.

These widgets define the model for the administration user interface.

Table 34. Widgets that define the model for the administration user interface

Widget class	Description
AdminTreeModel	Provides a widget that contains a hierarchical navigation tree for administration objects.

Table 34. Widgets that define the model for the administration user interface (continued)

Widget class	Description
NexusAdminCategoryObject	Provides methods that are used to handle category objects in the IBM Content Navigator administration tool. These category objects represent the **Desktops**, **Repositories**, **Plugins**, **Viewer Maps**, and **Menus** nodes in the main navigation tree in the administration tool. The methods behave differently based on the category. For example, the `onClick` method opens a tab and displays the children for the selected category.
NexusAdminInstanceObject	Provides methods that are used to handle Desktop, Repository, Plugin, Viewer Map, and Menu objects in the IBM Content Navigator administration tool. The methods behave differently based on the type of object. For example, the `getMenuActions` method lists the menu actions that are available for an object. This method returns the Edit and Copy actions for the default Viewer Map object. The method returns the Edit and Delete actions for a Repository object.
NexusAdminInterfaceTextObject	Represents the general interface text in the IBM Content Navigator administration tool.
NexusAdminSettingsObject	Represents the general settings such as logging and icon mapping in the IBM Content Navigator administration tool.

Dialog box widgets package

The ecm.widget.dialog package contains classes that define the dialog boxes that are used in the IBM Content Navigator web client.

Message dialog boxes

These widgets define dialog boxes that are used to display messages.

Table 35. Widgets that are used to display messages

Widget class	Description
ErrorDialog	Provides a dialog box that is used to display an error message that was generated by a service request.
MessageDialog	Provides a dialog box that displays a simple message to the user.
StatusDialog	Provides a dialog box that is used to display a status message during the processing of a request to the server.

Search dialog boxes

These widgets define dialog boxes that support the searching.

Table 36. Widgets that are used to search for items

Widget class	Description
ODManageSaveSearchDialog	Provides a dialog box that is used to choose a saved search in an IBM Content Manager OnDemand repository.
ODSaveSearchDialog	Provides a dialog box that is used to edit saved searches in an IBM Content Manager OnDemand repository.
SaveSearchDialog	Provides a dialog box that is used to create a saved search for an IBM Content Manager OnDemand repository.
SearchBuilderDialog	Provides a dialog box that is used to add search templates.
SearchDialog	Provides a dialog box that is used to open and run saved searches.
SearchSelectorDialog	Provides a dialog box that is used to select search templates.

Selection dialog boxes

These widgets define dialog boxes that are used to select objects and users.

Table 37. Widgets that are used to select items

Widget class	Description
ContentClassSelectorDialog	Provides a dialog box that is used to select content classes.
SelectObjectDialog	Provides a dialog box that is used to search for and select repository items.
SelectUserGroupDialog	Provides a dialog box that is used to select users and groups.

Work item and workflow dialog boxes

These widgets define dialog boxes that are used with work items and workflows.

Table 38. Widgets that are used with work items and workflows

Widget class	Description
ReassignToUserDialog	Provides a dialog box that is used to reassign an IBM FileNet P8 workflow item to another user.
StartWorkflow	Provides a dialog box that is used to start folders and documents on a workflow.
StepProcessorWindow	Provides a mechanism that is used to display the IBM FileNet Business Process Manager widget in a separate browser window.
SuspendWorkItems	Provides a dialog box that is used to suspend work items in a workflow.

Table 38. Widgets that are used with work items and workflows (continued)

Widget class	Description
WorkflowSubscriptionsDialog	Provides a dialog box that is used to suspend work items in a workflow.
WorkspacePropertiesDialog	Provides a dialog box that is used to edit workspace properties.

Miscellaneous dialog boxes

These widgets define various dialog boxes.

Table 39. Widgets that define dialog boxes

Widget class	Description
AboutDialog	Provides a dialog box that displays information about the application such as version information.
AddAnnotationDialog	Provides a dialog box that is used to add annotations to a document in an IBM Content Manager OnDemand repository.
AddContentItemDialog	Provides a dialog box that is used to add documents or folders to a repository. Users can use this dialog box to specify all required parameters for adding documents or folders.
AddPermissionDialog	Provides a dialog box that is used to add permissions to a document or folder.
AddToFavoritesDialog	Provides a dialog box that is used to add a repository item as a favorite.
AnnotationDialog	Provides a dialog box that is used to add and view annotations for an item in an IBM Content Manager OnDemand repository.
ApplyRemoveHoldDialog	Provides a dialog box that is used to apply or remove holds on IBM Content Manager OnDemand documents.
BaseDialog	Provides the base dialog box from which all other `ecm.widget.dialog` dialog box classes are derived.
ChangePasswordDialog	Provides a dialog box that is used to change the password on an IBM Content Managerserver or an IBM Content Manager OnDemand server.
CheckInDialog	Provides a dialog box that is used to check documents in to a repository.
ConfirmationDialog	Provides a dialog box that displays a question and prompts the user to approve or cancel the action.
ContentViewerWindow	Provides a separate browser window that displays the ContentViewer widget.
CreateHoldDialog	Provides a dialog box that is used to create a hold on an IBM Content Manager OnDemand item.

Table 39. Widgets that define dialog boxes (continued)

Widget class	Description
EditPropertiesDialog	Provides a dialog box that is used to edit the properties of documents, folders, and content items.
LoginDialog	Provides a dialog box that is used to log in to a content server.
MoveFileDialog	Provides a dialog box that is used to move a document or folder from one folder to another folder.
PrintDialog	Provides a dialog box that is used to print one or more documents.
ShowHyperlinkDialog	Provides a dialog box that is used to display the hyperlink to a folder or the hyperlink to a specific version of a document.
UnfileDialog	Provides a dialog box that is used to remove a document or a folder from a folder.
YesNoCancelDialog	Provides a dialog box that displays a question and the buttons **Yes**, **No**, and **Cancel**.

Layout widgets package

The `ecm.widget.layout` package contains classes that define components of the web client layout.

These widgets define components of the web client layout.

Table 40. Widgets that define components of the web client layout

Widget class	Description
_TabContainerBase	Provides the outermost tab container for a tabbed layout.
AdminPane	Provides a pane that is used to display the administration interface in the layout.
BaseLayout	Provides the base class from which all other layout widgets are derived.
BookmarkActionsHandler	Extends the `CommonActionsHandler` class to provide additional handlers for actions that are related to bookmarks.
BookmarkLayout	Provides a desktop layout that is used to display bookmarks.
BrowseFlyoutPane	Provides a fly-out pane that is used to browse folders and documents.
BrowsePane	Provides a pane that is used to browse folders and documents.
CommonActionsHandler	Provides the default implementation for many of the common actions. This class invokes methods from the ecm.widget.dialog package and ecm.model package.
FavoritesPane	Provides a pane that is used to store and access the documents, folders, or predefined searches that a user uses most frequently.

Table 40. Widgets that define components of the web client layout (continued)

Widget class	Description
FlyoutMenuContainerChild	Provides a container for the fly-out panes that are used for features.
HorizontalScrollPane	Provides a pane that can be scrolled horizontally.
LaunchBarContainer	Provides a vertical bar that displays the features available in a layout.
LaunchBarDialogPane	Provides a widget that is extended for any pane that is placed in a LaunchBarContainer fly-out. This pane provides flags and callbacks that are used by the parent container to control loading content in the fly-out and opening panes in the content area of the LaunchBarContainer widget.
LaunchBarPane	Provides a widget that is extended for any pane that is placed in the content area of the LaunchBarContainer widget. This class provides flags and callbacks that are used by the parent container to control navigation and content loading.
MainLayout	Provides the main layout for IBM Content Navigator or for similar desktop layouts. This class displays the launch bar that contains the features on the left side of the layout.
ManageTeamspacesPane	Provides a widget that displays a list of all teamspaces and teamspace templates that a user can access. From the list, a user with appropriate authority can open, edit, and delete teamspaces and teamspace templates. A user with the appropriate authority can also open the builder to create new templates and teamspaces from this widget.
NavigatorMainLayout	Extends the `MainLayout` class to provide additional interactions that are specific to IBM Content Navigator.
RecentActivityPane	Provides a pane that displays information about the recent activity on items.
SearchFlyoutPane	Provides a fly-out pane that contains the search interface for a layout.
SearchPane	Provides a pane that contains the search interface for a layout.
TeamspaceBuilderPane	Provides a pane to display the teamspace builder in the layout.
TeamspaceFlyoutPane	Provides a fly-out pane that is used for the teamspaces feature in the launch bar.
TeamspacePane	Provides a pane that is used to display a teamspace in the layout.
WorkFlyoutPane	Provides a fly-out pane that is used to select repositories and to navigate IBM Content Manager worklists and IBM FileNet P8 application spaces.

Table 40. Widgets that define components of the web client layout (continued)

Widget class	Description
WorkPane	Provides a pane that contains workflow navigation components. This pane restricts repository selection to IBM FileNet P8 repositories with workflow privileges and IBM Content Manager repositories.

Process widgets package

The `ecm.widget.process` package contains classes that are used to build custom launch processors and step processors for an IBM FileNet P8 workflow.

These widgets define launch processors and step processors.

Table 41. Widgets that define launch processors and step processors

Widget class	Description
_ProcessorMixin	Provides a mixin that defines methods for working with step processor metadata. This class creates the IBM FileNet P8 workflow response buttons. In addition, the class provides utility functions for working with filed data.
LaunchProcessorLayout	Provides a layout that is used to launch workflow processes.
MilestonesLayout	Provides a layout that is used to display milestones for an IBM FileNet P8 workflow.
ProcessorLayout	Provides the basic layout for IBM FileNet P8 workflow processors.
StepProcessorLayout	Provides the basic layout for step processors.
TrackerLayout	Provides the basic layout for the IBM FileNet P8 workflow tracker page. Users can use the tracker page to monitor the progress of a running workflow.

Teamspace builder widgets package

The `ecm.widget.workspaceBuilder` package contains the widgets that create and edit teamspaces and teamspace templates. In IBM Content Navigator, these widgets are used for the teamspace builder.

These widgets are used to create a teamspace or teamspace template.

Table 42. Widgets that are used to create teamspaces and teamspace templates

Widget class	Description
ClassesPane	Provides a widget that is used to specify the classes that teamspace users can use to add documents to the repository. In IBM Content Navigator, this widget defines the Select Classes pane.
EditRoleDialog	Provides a dialog box that is used to edit the name and description for a role.

Table 42. Widgets that are used to create teamspaces and teamspace templates (continued)

Widget class	Description
FoldersAndDocsPane	Provides a widget that is used to select the default documents and folders to include in the teamspace. In IBM Content Navigator, this widget defines the Include Folders and Documents pane.
ManageTeamspaceDialog	Provides a widget that lists the teamspaces and teamspace templates that are defined for a repository. In IBM Content Navigator, this widget defines the Teamspaces View pane.
Pane	Provides the base class from which all teamspace builder panes are derived.
PropertiesPane	Provides a widget that is used to specify properties for teamspaces and teamspace templates. For a teamspace, this widget is used to specify the name, description, and template. For a teamspace template, this widget is used to specify the name, description, and security. In IBM Content Navigator, this widget defines the Define Teamspace pane and the Define Teamspace Template pane.
RenameFolderDialog	Provides a dialog box that is used to rename a folder in a teamspace.
RoleSloshBucket	Provides a slosh bucket that is used to assign roles to users in a teamspace. In IBM Content Navigator, this widget is used in the **Select Users** pane.
RolesPane	Provides a widget that is used to add users to the teamspace and assign roles to the users. In IBM Content Navigator, this widget provides the Select Users pane.
SearchTemplatesPane	Provides a widget that is used to select the searches that will be available to teamspace users. In IBM Content Navigator, this widget provides the Select Searches pane.
SelectionToolbar	Provides a toolbar that supports selection of multiple items in teamspace builder.
TeamspaceBuilder	Provides the widget that contains the user interface for teamspace builder.
TeamspaceColumnPropsPane	Provides a widget that displays the columns of information that are returned by a search.
TeamspacePropertiesDialog	Provides a widget that is used to edit the properties of a teamspace.

Table 42. Widgets that are used to create teamspaces and teamspace templates (continued)

Widget class	Description
UsersPane	Provides a widget that is used to add users to the teamspace and to assign roles to these users. In IBM Content Navigator, this widget provides the **Select Users** pane.
VerticalSloshBucket	Provides a vertical slosh bucket that is used in teamspace builder to add classes or entry templates, searches, and existing roles to teamspace templates.

Viewer widgets package

The `ecm.widget.viewer` package contains classes that are used to view document content.

These widgets define dialog boxes that are used to view content.

Table 43. Widgets that are used to view content

Widget class	Description
AjaxViewer	Provides a widget that supports page-by-page viewing of documents.
ContentViewer	Provides a widget that contains tabs in which individual viewers for documents are displayed.
ContentViewerPane	Provides a widget that supports a single document viewer. This widget is contained in the ContentViewer widget.

Notices

Trademarks

Index

Product Number: 1234-SS1

SC19-3716-00