Cúram 8.2

Pod Developers Guide

Note

Before using this information and the product it supports, read the information in Notices on page 45

Edition

This edition applies to Cúram 8.2.

Contents

Note

Edition

1 Developing pods 11

1.1 Overview 11

Prerequisites 11

Further Reading 11

1.2 A Technical Overview 11

What is a Pod? 11

What is a Pod page? 12

How does it work? 12

UIM Page 12

PodContainer 12

PodLoader 12

Database Tables 12

Loading the Page 13

Rendering the page 13

Saving the Page 13

Configuring Pods 13

Product Pods 14

User configuration of Pod Pages 14

Developing new Pods 15

1.3 Getting Started 15

Creating a page with a Pod container 15

Identifying a Pod page 15

Configuring the database information about the page 16

Testing the page 17

1.4 Hello World Pod 17

Declaring a new Pod 18

Declaring a new PodLoader 18

Creating a Pod using a PodLoader 19

Adding a Pod to the Pod Container 20

Viewing the Pod 20

Review 21

1.5 Creating a Pod with a list 21

Creating a new list Pod 21

Register new Pod 21

Create a new PodLoader 22

Create the list....22
Deconstructing the code....23
Constructing the list....23
Adding rows....24
Creating content in the cells....24
Adding the list to a Pod....24
1.6 Adding a Pod filter....25
What is a Pod filter?....25
Types of filter....25
Adding a Drop Down Filter....26
Creating the Pod Filter....26
Creating the options....27
Creating the selections....27
Setting the type of filter....28
Adding a label and CSS styling....28
Add the Filter to the Pod....28
Filtering your Pod....29
1.7 Creating new Pod filters....29
Create a Pod filter Renderer....30
Preparing to delegate....30
Setting a source path....30
Setting a target path....30
Creating the input field....32
Create a configuration for the Pod filter Renderer....32
Create a new PodFilter in the PodLoader....32
1.8 Localization in Pods....34
The textresource property....34
Setting the text resource....35
Localizing the My Favorite Movies Pod....35
Localizing the Pod....35
Localizing the filter....36
Localizing the movie list....36
Sharing properties files....37
1.9 Sample program listings....37
Sample: The movies DB: A Java class serving our favorite movies....38
Sample: Hello World Pod-Loader....39
Sample: My favorite movies Pod-Loader....39
Sample: My Favourite Movies Pod-Loader for Pod filter....40
Sample: PodTextFilterRenderer for new Pod filter example....41
Sample: My Favorite Movies Pod-Loader for new Pod filter....42
Sample: My Favourite Movies Pod-Loader for localization....43

Notices.....
Privacy policy..... 46
Trademarks..... 46

Chapter 1 Developing pods

Use this information to develop SPM pods. Pods are presented through a standard UIM Page. The UIM includes the PodContainer.vim that contains the predefined API for interacting with the pod. The PodContainer interface allows the client to interact with the server. A PodLoader is required for each pod.

1.1 Overview

The guide is a cookbook for Developers who want to create Pods. The guide coaches Developers through various scenarios beginning with the simplest implementation of a Pod, then adding content to Pods using tools provided and eventually introducing more advanced scenarios where the user requires knowledge of the widget development process.

The guide is aimed at Developers who want to create new Pods and new Pod Pages.

Prerequisites

Users of this guide need basic Java® , XML, HTML and CSS skills and a knowledge of the development environment. For the more advanced material the user needs to be familiar with the rendering framework which is covered in the Cúram Widget Development Guide.

Further Reading

Table 1: Further Reading

Guide	Description
Cúram Custom Widget Development Guide	A complete reference for developing custom widgets
Cúram Personal Page Configuration Guide	How to configure Personal Pages (Pod Pages)

1.2 A Technical Overview

What is a Pod?

A Pod is a user interface widget that can be placed on a client page. In this respect, it is no different to any other user interface widget that presents data such as a list or cluster. Where a Pod differs from other types of widgets is that it can be placed in a Pod-Container where a number of more features are activated, such as the ability to be repositioned in the container and the persistence of user settings such as whether the Pod is displayed and what filter settings are

applied. A filter is an optional feature of a Pod that allows the content to be customized by the user, it can be accessed if available through the pen icon on the title bar of the Pod.

What is a Pod page?

A Pod page is a UIM page, which contains a Pod-Container widget. The Pod-Container widget manages Pods. The widget is configured to present a selection of Pods that can be viewed in the container. The addition and removal of Pods from the container is managed through a *customization-console*. The Pod-Container widget manages the movement of Pods to different locations within the container. Where applicable it processes filters associated with Pods. In each case, the last configuration of the Page is saved for the current user and retrieved the next time that they load the page.

How does it work?

The next section provides an overview of the artifacts that work together to present a Pod page.

UIM Page

The Pods are presented through a standard UIM Page. The UIM must include the *`PodContainer.vim`* which contains the predefined API for interacting with the Pods infrastructure including the display of the page and saving of user preferences.

PodContainer

The PodContainer is the interface through which the client interacts with the server. At the display phase, the server interface invokes the `loadData()` method on the `PodContainer` class. At action phase 1 of the save APIs processes the data from the Pod-Container. The *`PodContainer.vim`* provides a reusable interface to the Pod infrastructure, add the *`PodContainer.vim`* to your UIM page and you have a fully functioning interface.

PodLoader

A PodLoader must be written for each Pod. The PodLoader defines the Pod and its content. This book mainly deals with the development of PodLoaders.

Database Tables

A number of tables are used to manage Pods.

Table 2: Database tables used to load Pods

Table	Description
PODTYPE	A list of all existing Pods

Table	Description
PODLOADERBINDINGS	A list of all existing PodLoaders mapped to a Pod type
PAGECONFIG	A list of configurations of Pod Pages
USERPAGECONFIG	A list of user customizations of Pod Pages

Loading the Page

At the display phase, the server interface starts the `loadData()` method on the `PodContainer` class. The `PodContainer` uses the `PodContainerManager` to identify all the Pods to be displayed on the page that uses the information in the PAGECONFIG and USERPAGECONFIG database tables. The PodContainerManager then identifies the PodLoader for each Pod to be displayed using the information in the PodType and PodLoaderBindings codetables. The PodContainer manager starts the createPod() method on each PodLoader. The PodLoader supplies the data for a single Pod and the PodContainerManger builds up the cumulative data for all the Pods within the container.

Rendering the page

The page rendering is handled by a collection of renderers. The rendering begins with the PodContainerRenderer, which receives the document from the loading process and generates the PodContainer widget. It then delegates the rendering of Pods to a Pod renderer, which in turn, delegates to other renderers by using markers in the data it receives. Each renderer returns its own content that it either generates itself or generates with the help of other renderers. This pattern of delegation is repeated until all content is rendered.

For details about the rendering framework and how renderers interact, see "Developing Custom Widgets".

Saving the Page

At the action phase, the server interface saves any changes that the user made to the Pod selection and layout of the container back to the database again through the PodContainer API. The page is saved by any of the following actions, clicking the save button in the customization console, clicking the save button on a Pod filter, dragging and dropping a Pod (each time a Pod is dropped the save action is invoked to record the new layout of the page).

Configuring Pods

A Pod page can be configured through an Administration wizard, which allows the layout and content of the Pod page to be defined. A full explanation of the Administration wizard is available in the Curam Personal Page Configuration guide.

Pod Dimensions

The dimensions of a Pod are not directly specified by a Pod. This allows Pods to dynamically resize to fit their environment and facilitates the reuse of Pods across Pod containers.

- **Pod Height**
  The height of each Pod is determined by its content. A Pod's height extends to display its content.
- **Pod Width**
  The width of a Pod is determined by the container it is being displayed in. Each Pod container is configured with a number of equally sized columns. The Pod width will dynamically size to fill the width of the column it is placed in.

Tip: When deciding on a layout for your Pod page we recommend that you consider the type of Pods you are adding to the container and how they might be affected by resizing. Many of the predefined Pods are optimally sized for a 3 column layout. Using alternate layouts may distort the content of the Pods and visually this could detract from the page.

Product Pods

A collection of Pods are provided with the product. The Home section of each Application view is pre-configured with a set of Pods appropriate to that Application view (Pods can be shared across Pod pages). The configuration for each Application view can be updated by an administrator.

See the Curam Personal Page Configuration guide for details.

User configuration of Pod Pages

Each Pod page is pre-configured with a set of available Pods and a set of selected Pods which are visible in the container.

An application user can further customize the workspace by...

- adding Pods from the available list by using the customization-console
- removing Pods by using the customization console
- removing Pods by using the close button on the title bar of the Pod
- moving Pods by dragging to a new location in the container
- filtering Pods by using the filter feature (where available).

Each time a user takes one of the actions that are listed above a record of the current configuration of the page is saved. When the page reloads this saved configuration is redisplayed.

Developing new Pods

In addition to reusing the Pods that are provided in the product, an Organization may want to create new Pods. The Pod framework has the ability to create new Pods with custom content. This guide presents examples of how this can be done.

1.3 Getting Started

Before creating a Pod you need to create a Page to host it. The page that hosts our Pods needs a Pod container which manages the Pods allowing them to be added/removed/moved and updated.

Creating a page with a Pod container

Starting with a page that is mapped to a section and tab in the application, add a Pod Container to the page by including the PodContainer.vim file as in the following example:

```
<PAGE PAGE_ID="MyPodContainer"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="file://Curam/UIMSchema.xsd"
>

  <CONNECT>
    <SOURCE NAME="CONSTANT" PROPERTY="MyPodContainer"/>
    <TARGET NAME="DISPLAY" PROPERTY="pageID$pageID"/>
  </CONNECT>

  <INCLUDE FILE_NAME="PodContainer.vim"/>

</PAGE>
```

Related concepts

For information on how to map pages in the application.

Identifying a Pod page

Add a *`Constant.properties`* file to the same folder as the UIM file. Add a property to the file that maps to the name of the constant used in the UIM to the *page-id* of the UIM page. When the server interface is called this value is used to uniquely identify the Pod page.

Constant.properties

```
MyPodContainer=MyPodContainer
```

Configuring the database information about the page

The Pod page requires 2 database records to operate. The PAGECONFIG table stores information about which Pods are available on the page. The USERPAGECONFIG table stores the users customizations.

Add the following DMX files to the component and run the database build target to insert the records:

```
<?xml version="1.0" encoding="UTF-8"?>
<table name="PAGECONFIG">
  <column name="pageConfigID" type="id"/>
  <column name="userRoleName" type="text"/>
  <column name="pageID" type="text"/>
  <column name="config" type="text"/>
  <column name="versionNo" type="number"/>

  <row>
    <attribute name="pageConfigID">
      <value>9999</value>
    </attribute>
    <attribute name="userRoleName">
      <value></value>
    </attribute>
    <attribute name="pageID">
      <value>MyPodContainer</value>
    </attribute>
    <attribute name="config">
      <value>
       &lt;page-config&gt;
       &lt;contexts&gt;
       &lt;sequence domain="CURAM_CONTEXT"/&gt;
       &lt;/contexts&gt;
       &lt;availablePods&gt;
       &lt;sequence domain="POD_TYPE_SELECT"&gt;
       &lt;/sequence&gt;
       &lt;/availablePods&gt;
       &lt;layout&gt;
       &lt;sequence domain="COL_SIZE"&gt;
       &lt;value&gt;33&lt;/value&gt;
       &lt;value&gt;33&lt;/value&gt;
       &lt;value&gt;33&lt;/value&gt;
       &lt;/sequence&gt;
       &lt;/layout&gt;&lt;/page-config&gt;
      </value>
    </attribute>
    <attribute name="versionNo">
      <value>1</value>
    </attribute>
  </row>
</table>
```

USERPAGECONFIG.DMX

```
<?xml version="1.0" encoding="UTF-8"?>
<table name="USERPAGECONFIG">
  <column name="userPageConfigID" type="id"/>
  <column name="userRoleName" type="text"/>
  <column name="userName" type="text"/>
  <column name="pageID" type="text"/>
  <column name="config" type="text"/>
  <column name="defaultInd" type="bool"/>
  <column name="versionNo" type="number"/>
  <row>
    <attribute name="userPageConfigID">
      <value>9999</value>
    </attribute>
    <attribute name="userRoleName">
      <value></value>
    </attribute>
    <attribute name="userName">
      <value/>
    </attribute>
    <attribute name="pageID">
      <value>MyPodContainer</value>
    </attribute>
    <attribute name="config">
      <value>
       &lt;user-page-config&gt;&lt;/user-page-config&gt;
      </value>
    </attribute>
    <attribute name="defaultInd">
      <value>1</value>
    </attribute>
    <attribute name="versionNo">
      <value>1</value>
    </attribute>
  </row>
</table>
```

Testing the page

Build the application, launch it, login and go to the new Pod Page.

When the new Pod page loads it is empty except for few buttons in the top right corner. The container is empty because you did not add any Pods to the page. Clicking the Customize button opens the customization-console. When the console opens it is empty except for the action buttons. Again, because you do not assign any Pods to the container there are no Pods to select.

- The *Save* button stores the current users customizations.
- The *Reset* button deletes the current users customizations and revert to the default for this Page.
- The *Cancel* button resets the selection in the customization-console and closes it.

In the next section you create a simple Pod and add it to the container.

1.4 Hello World Pod

In this section, you are going to create a basic Pod with a title and some text. You also use the Admin Wizard to add the new Pod to the Pod page.

There are 4 basic steps to get the Pod on a page...

1. Declaring a Pod
2. Declaring a PodLoader
3. Implementing a PodLoader
4. Adding the Pod to the Pod Container

Declaring a new Pod

The first step is to declare a new Pod. The `PodType` is used for this purpose. Create a file CT_PodType.ctx in the component. Add a code and value for the new Pod like the following example. The convention is to use the prefix *PT* for the value. The description field is used by the Administration wizard to refer to the Pod.

Example CT_PodType.ctx, declaring a 'Pod-Type':

```
<?xml version="1.0" encoding="UTF-8"?>
<codetables package="curam.codetable">
  <codetable java_identifier="PODTYPE" name="PodType">
    <code
      default="false"
      java_identifier="HELLOWORLD"
      status="ENABLED"
      value="PT9001"
    >
      <locale language="en" sort_order="0">
        <description>Hello World!</description>
        <annotation/>
      </locale>
    </code>
  </codetable>
</codetables>
```

Related reference

Sample: The movies DB: A Java class serving our favorite movies on page 38

Declaring a new PodLoader

Next, you need to declare the PodLoader. The PodLoader is the java class that generates the fragment of XML that will populate the Pod. The `CT_PodLoaderBindings.ctx` codetable entry binds a Pod-Type to a PodLoader. When the infrastructure processes the Pod, it looks up the PodLoader class in this codetable.

- The *value* field must match the value field on the PodType codetable. This is what binds the 2 codetable entries.
- The *description* field contains the fully qualified name of the PodLoader class.

CT_PodLoaderBindings.ctx, declaring a 'PodLoader':

```
<?xml version="1.0" encoding="UTF-8"?>
<codetables package="curam.codetable">
  <codetable
    java_identifier="PODLOADERBINDINGS"
    name="PodLoaderBindings"
  >
    <code
      default="false"
      java_identifier="HELLOWORLD"
      status="ENABLED"
      value="PT9001"
    >
      <locale language="en" sort_order="0">
        <description>pods.podloaders.HelloWorld</description>
        <annotation/>
      </locale>
    </code>
  </codetable>
</codetables>
```

Now that you added the codetable entries to the PodType and PodLoaderBindings files you need to run the **ctgen** target to create the codetables and the **database** target to insert the codetable values into the database.

Related reference

Sample: Hello World Pod-Loader on page 39

Creating a Pod using a PodLoader

The next step is to create the PodLoader class. The PodLoader extends the class *curam.cefwidgets.pods.pod.impl.PodLoader* and implements the `createPod` method. Create a new class on the Server by copying this example into a class named *HelloWorld* in the package pods.loaders.

A very simple PodLoader:

```
001  package pods.podloaders;
002
003  import java.util.Map;
004  import org.w3c.dom.Document;
005  import org.w3c.dom.Node;
006  import curam.cefwidgets.docbuilder.impl.PodBuilder;
007  import curam.cefwidgets.pods.pod.impl.PodLoader;
008  import curam.codetable.PODTYPE;
009
010  public class HelloWorld extends PodLoader {
011
012    @Override
013    public Node createPod(Document document, Map<String,Object> contexts) {
014      try{
015        PodBuilder helloWorld =
016          PodBuilder.newPod(document, PODTYPE.HELLOWORLD);
017        helloWorld.setTitle("Hello World");
018        return helloWorld.getWidgetRootNode();
019      }catch(Exception e){
020        throw new RuntimeException(e);
021      }
022    }
023  }
```

Input:

The `createPod` method receives 2 parameters from the infrastructure that calls it.

Table 3: `createPod` method parameters

Parameter	Description
document	The Document parameter is an instance of a `org.w3c.Document` class. It is passed to the method by the infrastructure that calls it. The Document instance is used to create and append the 'pod' Node that describes the Pod.
context	The context parameter is used to pass page level parameters to the Pods. Currently this is not supported.

Output:

An instance of the `org.w3c.Node` object is returned by the createPod method.

Table 4: Return object from createPod

Return object	Description
org.w3cNode	The content of the Node that is returned must match a predefined schema. The PodBuilder class provides an API to create a 'pod' Node in the correct format.

In the example above, the simple Pod is created by creating a new instance of a PodBuilder class on line 16. The Document instance from the PodLoader and the codetable value from the PodType codetable are passed to the constructor. On line 17 we use the PodBuilder to set the title of the Pod. The PodBuilder builds a Node tree representing the Pod which is returned on line 18 as a Node object.

Adding a Pod to the Pod Container

The last piece of the jigsaw is adding the Pod to the Pod-Container. You use the wizard provided in the Administrator application. You must login to the Administrator application, so you need the username and password assigned to this application.

When you have logged in you must open the admin wizard by...

1. Selecting the Administration Workspace section
2. Selecting the User Interface tab
3. Selecting Personalized Pod Pages

When the Personalized Pod Pages tab loads you can see the MyPodContainer page that you created in the list of Personal Pages. Selecting edit opens the wizard for maintaining the Personal Page. The first step lists all the Pods available for selection. In this list you find the Pod 'HelloWorld!'. Finally, select the Pod and click next on the remaining steps saving the record. You have now added the Pod to the Pod Container. Log out of the Administrator application and log into the application that contains the Pod page.

Viewing the Pod

Now lets see the Pod in action. Login to the application and go to the Pod page. When the page loads it is empty except for the buttons in the top right corner. Click the customize button to open

the *customization-console*. You can see the Pod listed in the console. Select the checkbox beside the Pod and choose save. The page reloads with the Pod defaulted to the top right corner.

You notice that the Pod contains some text *NO CONTENT* which is a place holder added by the infrastructure when the Pod contains no content. In the next section you create another Pod with some content and take a closer look at the PodBuilder class.

Review

In this section, you completed the following:

- You started by adding the new Pod to the PodType and PodLoaderBindings codetables.
- You then created a PodLoader where you used the PodBuilder class to create a Pod and add the title.
- You used the wizard in the Administrator application to add the new Pod to the PodContainer.
- You used the customization-console to select and view the new Pod.

In the next section you create a new Pod with some more interesting content.

1.5 Creating a Pod with a list

In this section you expand on what you did in the previous section by adding some content to a Pod and you use the tools provided for creating the basic content types.

Use a new Pod which you add to the Pod-Container in the same way you added the Hello World! Pod in the previous section. You use a movies theme for the examples, so now you can create a Pod with a short list of your favourite movies.

Creating a new list Pod

Register new Pod

In the same way you did in the previous section you are going to register a new Pod and bind it to a PodLoader by adding the codetable entries in the PodType and PodLoaderBindings tables by using the examples shown here.

Example 1: Adding a new PodType to CT_PodType.ctx

```
<code
    default="false"
    java_identifier="MYFAVMOVIES"
    status="ENABLED"
    value="PT9002"
  >
    <locale language="en" sort_order="0">
      <description>My Favourite Movies</description>
      <annotation/>
    </locale>
  </code>
```

Example 2: Adding PodLoader binding

```
<code
    default="false"
    java_identifier="MYFAVMOVIES"
    status="ENABLED"
    value="PT9002"
  >
    <locale language="en" sort_order="0" >
      <description>pods.podloaders.MyFavouriteMovies</description>
      <annotation/>
    </locale>
  </code>
```

Create a new PodLoader

Next, you add the PodLoader class to your loaders package remembering to reference the new codetable value you created in the *PodType* codetable when you construct the new Pod by using the PodBuilder.

Creating a PodLoader class:

```
001  package pods.podloaders;
002
003  import java.util.Map;
004  import org.w3c.dom.Document;
005  import org.w3c.dom.Node;
006  import curam.cefwidgets.docbuilder.impl.PodBuilder;
007  import curam.cefwidgets.pods.pod.impl.PodLoader;
008  import curam.codetable.PODTYPE;
009
010  public class MyFavouriteMovies extends PodLoader {
011
012    @Override
013    public Node createPod(Document document, Map<String,Object> contexts) {
014      try{
015        PodBuilder moviesPod =
016          PodBuilder.newPod(document, PODTYPE.MYFAVMOVIES);
017        moviesPod.setTitle("My Favourite Movies");
018        return moviesPod.getWidgetRootNode();
019      }catch(Exception e){
020        throw new RuntimeException(e);
021      }
022    }
023  }
```

Log into the Administrator application and add the new Pod to the Pod-Container in the same way you did in the previous section.

Open the Pod page and ensure that the Pod is visible.

Create the list

Now that you have a Pod in place you can add content to it. The `PodBuilder` class provides an `addContent(...)` method to add the content to a Pod. In the movies example you are going to delegate to the list widget which can generate a HTML *table*.

To start you need to provide the movies for a list. The related information sample below contains a full program-listing for a Java class that act as a simple read-only DB of your favorite movies. Add this class to a package in the project where it can be accessed by our PodLoader.

Next, you create a list in our PodLoader and populate it with the favorite movies. In the PodLoader add the following code to the `createPod` method before the return statement.

Adding a list to a Pod:

```
001  public Node createPod(Document document, Map<String,Object> contexts) {
002    try{
003      PodBuilder moviesPod =
004        PodBuilder.newPod(document, PODTYPE.MYFAVMOVIES);
005      moviesPod.setTitle("My Favourite Movies");
006
007      MoviesDB moviesDB = new MoviesDB();
008
009      Collection<MoviesDB.Movie> favMovieCollection =
010        moviesDB.getAllMovies();
011      Iterator<MoviesDB.Movie> movieList =
012        favMovieCollection.iterator();
013
014      // Create the list
015      ListBuilder myFavouriteMovies =
016        ListBuilder.createList(1, document);
017
018      int row = 1;
019      while(movieList.hasNext()) {
020        Movie movie = movieList.next();
021        String movieName = movie.title;
022        myFavouriteMovies.addRow();
023        myFavouriteMovies.addEntry(1, row++, movieName);
024      }
025
026      RendererConfig contentRenderer = new RendererConfig(
027          RendererConfigType.STYLE, "single-list");
028      moviesPod.addContent(myFavouriteMovies, contentRenderer);
029
030      return moviesPod.getWidgetRootNode();
031    }catch(Exception e){
032      throw new RuntimeException(e);
033    }
```

Compile your PodLoader class and reload the Pod page. The 'My Favourite Movies' Pod are updated with the list of your favourite movies.

In the next section you can look in more detail at how the list was created.

Related reference

Sample: My favorite movies Pod-Loader on page 39

Deconstructing the code

Constructing the list

A Pod does not need to know what its content will be. At run time the Pod delegates to other widgets to produce the HTML that renders the content. Your movies Pod is a list of movie names and it reuses another widget to return a HTML table containing the list data. Like the PodBuilder the ListBuilder is an API for creating lists that conform to the schema for a renderer called `ListBodyRenderer`. The ListBuilder generates a fragment of XML that describes a list and at run time the `ListBodyRenderer` translates this XML into the HTML that can be added to the body of a Pod. To build the Pod content for a Pod the PodLoader use the ListBuilder to produce the list of movies.

The first step in creating a list is to construct a new `ListBuilder` object. The constructor on line 16 accepts an *int* value which is the number of columns in the list. The second parameter is a `org.w3c.Document`. The document parameter represents the overall PodContainer to which a Pod is added. The document object is used to create the new Nodes that represent a Pod and its content. Those Nodes is appended to some part of the document object.

```
015      ListBuilder myFavouriteMovies =
016        ListBuilder.createList(1, document);
```

Adding rows

Next, iterate over the movies. For each movie you add a new row (line 22).

```
019      while(movieList.hasNext()) {
020        Movie movie = movieList.next();
021        String movieName = movie.title;
022        myFavouriteMovies.addRow();
023        myFavouriteMovies.addEntry(1, row++, movieName);
024      }
```

Creating content in the cells

You use the `addEntry(...)` method to add content to cells. This method accepts a column, a row and a Java Object, which represents the content to be added to the cell.

Table 5: ListBuilder.addEntry(...) parameters

Parameter	Type	Description
col	int	The column index, offset 1.
row	int	The row index, offset 1.
content	Object<?>	A Java Object that represents the content. The List Renderer can accept a number of different types including CodetableItems and LocalizedString objects which it processes for display. (See Javadoc for ListBuilder)

In the movies Pod you want to add a list of movie names so you pass a Java String in the *content* parameter. On lines 19 to 24 we iterate over the collection of movies.

```
023        myFavouriteMovies.addEntry(1, row++, movieName);
```

Adding the list to a Pod

Now that the list is populated you insert it into the body of the Pod.

The `addContent(...)` method provides the mechanism for adding the Pod content. The method accepts as its first parameter either a `org.w3c.Node` or a `WidgetDocumentBuilder` object (which internally is converted to a Node using the `getWidgetRootNode` operator of the WidgetDocumentBuilder object).

The second parameter is a configuration for a Renderer that creates the HTML for our Pod content. The `RendererConfig` object specifies the type of configuration (Style or Domain) and name of a renderer configuration entry. Configuring renderers is covered in detail in the Curam Widget Development Guide.

Table 6: PodBuilder.addContent(...) parameters

param	type	descrption
content	Node | WidgetDocumentBuilder	The Node object is appended to the instance of org.w3c.Document that was passed to the constructor of the PodBuilder.
rendererConfig	RendererConfig	The RendererConfig object nominates the Renderer that processes the *content* parameter.

```
026      RendererConfig contentRenderer = new RendererConfig(
027          RendererConfigType.STYLE, "single-list");
028      moviesPod.addContent(myFavouriteMovies, contentRenderer);
```

The movies Pod uses the `ListBodyRenderer` which is invoked using a Style configuration called "single-list". On line 28 we add the list widget with the renderer configuration for a list to the body of the Pod.

The Pod is now complete. The content of your movies list is defined in the ListBuilder object which is added to the Pod. The ListBodyRenderer generates the HTML table which is appended to out Pod body.

1.6 Adding a Pod filter

In this section you explore Pod filters. You look at the existing filters available and you use one to add a filter to the movies Pod.

What is a Pod filter?

A Pod can optionally include a Pod filter. The filter allows a user to refine the information that is presented in the Pod. For example, some Pods display reports as charts that are based on periods of time. A Pod filter may present a selection of time periods which the user can select to redraw the Pod with a different chart representing the selected time period.

Types of filter

The ChoiceRenderer is a generic renderer for a number of filter style renderers, such as checkboxes, radiobuttons, and dropdowns. The ChoiceRenderer delegates to a specific renderer depending on what displayType is selected by the ChoiceBuilder.

The following table lists the existing filter renderers. The type and displayType combine to select a specific renderer.

Table 7: Filter Types

Filter	CT*	Type	Display Type	Renderer
Checkbox	Y	multiple	n/a	CTCheckboxSelectRenderer
Radiobutton	Y	single	n/a	CTRadiobuttonSelectRenderer
Radiobutton	N	db-single	n/a	RadiobuttonSelectRenderer
Dropdown	Y	single	dropdown	CTDropdownSelectRenderer
Dropdown	N	single	listdropdown	ListDropDownSelectRenderer

Note: CT *, Denotes a filter based on the values in a specific codetable file.

Adding a Drop Down Filter

To demonstrate the use of filters you can create a filter for our movies Pod. The filter selects movies by genre. As you did in the last section you insert the complete code sample first to see the Pod in action, then you step through the code to see what you did to create the filter.

Replace the original createPod method in the `MyFavouriteMovies` PodLoader with the version in the Sample: My Favourite Movies Pod-Loader for Pod filter topic in the links provided. Compile the PodLoader and start the Application.

When the page loads the Pod is updated to include a filter feature denoted by the pen icon on the title bar.

Open the filter by clicking on the pen icon. Select a genre from the drop-down. Use the Save button to save the selection and reload the list. The list only returns movies that match the selected genre in the dropdown.

Lets look at the steps you took to create the filter.

Related reference

Sample: My Favourite Movies Pod-Loader for Pod filter on page 40
Sample: Hello World Pod-Loader on page 39

Creating the Pod Filter

To add a filter to the Pod, you need to use the `PodBuilder.addFilter(...)` method which accepts a parameter of type `PodFilter`. The `PodFilter` object specifies the id of the filter and the renderer configuration that is used to invoke the render that creates the filter.

In our example you are creating a filter with the id "genre" and we are using a renderer called the ChoiceRenderer to render the content of the filter.

Creating the Pod Filter

```
010      RendererConfig filterRenderer =
011        new RendererConfig(RendererConfigType.DOMAIN, "CT_CHOICE");
012
013      // Create the PodFitler
014      PodFilter genreFilter =
015        new PodFilter("genre", document, filterRenderer);
```

On line 10-11 you create a renderer configuration which is mapped to a domain 'CT_CHOICE'. This configuration invokes a renderer called `ChoiceRenderer`. You then create a PodFilter object passing an id, the document instance of the PodLoader and the renderer configuration.

Creating the options

Now that you have the basic framework of a filter you need to add the choices. The filter can be described as a set of options and a set of selections, which are a subset of the options. Collectively you refer to these as the 'choices'. As you are using the ChoiceRenderer to create the drop-down list, so you can use the ChoiceBuilder to create the content that you pass to the ChoiceRenderer. The ChoiceBuilder accepts a HashMap which is the set of id's and values. In this case the values are the list of genres that are displayed in the drop down.

In this simple example you use the lower case version of the value as the id.

Creating the set of Choices for the genre drop-down:

```
018      HashMap<String, String> genres = new HashMap<String, String>();
019      genres.put("all", "- All -");
020      genres.put("horror", "Horror");
021      genres.put("drama", "Drama");
022      genres.put("romance", "Romance");
023      genres.put("comedy", "Comedy");
024      genres.put("action", "Action");
025
026      // Create the options and selections using the ChoiceBuilder.
027      ChoiceBuilder choices =
028        ChoiceBuilder.newInstance(genres, document);
```

Creating the selections

The next step is adding the selected values. In most cases, you want this to be the last saved selections. You can retrieve these values because they are saved for each filter every time a save action occurs on the container. The `PodLoader` class provides a `getPodFilterById(...)` which returns the selected values for each Pod filter.

Retrieving the saved selections and adding them to the Pod filter:

```
031      Node genreSelectionNode =
032        getPodFilterById(PODTYPE.MYFAVMOVIES, "genre", document);
033
034      // Convert the Node to an ArrayList.
035      ArrayList<String> selectedGenres =
036       PodFilter.convertSelectionsNodeToArrayList(genreSelectionNode);
037
038      // Create a default genre selection.
039      if (selectedGenres.isEmpty()){
040        selectedGenres.add("all");
041      }
042      choices.addSelection(selectedGenres.get(0));
```

On line 32, you use the `getPodFiltersById(...)` method to return the saved selections for the 'genre' filter on the 'MYFAVMOVIES' Pod. The values are returned as a Node object in the raw format that they were encoded and stored as. The `PodFilter.converSelectionsNodeToArrayList(Node)` utility is used to convert the values into a list of String values. On line 42, you add the selected value, in this case it is the only value that is returned in the array.

Setting the type of filter

In our example, you are using the ChoiceRenderer to create a dropdown list. The ChoiceRenderer delegates to a specific renderer that depends on what displayType is selected by the ChoiceBuilder. You are creating a drop down list, which is not based on a codetable, so you selected "listdropdown" for the display type.

Setting the type of filter:

```
043      choices.setTypeOfDisplay("listdropdown");
```

Adding a label and CSS styling

Optionally you can add a label to the filter by passing a String * to the `addFitlerLabel(...)` method. Custom styling can also be applied to the filter by passing CSS class names to the `addCSSClasses(...)`

Note: * The filter label is configured for localization. The String passed to the addFilterLabel method is assumed to be a key in a properties file associated with the Pod. If no property value is returned by the key, the key is used as the label.

Adding a PodFilter to a Pod:

```
048      genreFilter.addFilterLabel("Genre");
049      genreFilter.addCSSClasses("genre-filter");
```

Add the Filter to the Pod

Next, you add the filter to the Pod by passing it to the `PodFilter.addFilter(...)` method.

To add a PodFilter to a Pod:

```
050      moviesPod.addFilter(genreFilter);
```

Filtering your Pod

The final task is to filter the content of the Pod. In the movies example you want to filter out all movies where the genre does not match the currently selected one.

To filter the movies by genre:

```
067         if (selectedGenre.equals(movie.genre)
068             || selectedGenre.equals("all")){
069
070           myFavouriteMovies.addRow();
071           myFavouriteMovies.addEntry(1, row++, movieName);
072         }
```

So that completes the filter. When the Pod is loaded for the first time no value is stored for the filter. Every subsequent save stores the filter value, even if that is an empty String. When the Pod reloads it uses the saved value to filter the list of Movies, and it also passes the stored value back to the filter for display so that you can see what filter is being applied.

Using the `PodBuilder`, `PodFilter` and `ChoiceBuilder` has meant that there was no requirement to create Renderers. The builder classes allow you to reuse existing renderers. There are however be occasions where you want to create a custom filter type. In the next section, see how to create a new filter renderer.

1.7 Creating new Pod filters

In this section, you are going to create a new filter for a Pod to demonstrate how to add form items to Pods.

To complete this section you need to create a Renderer so you need to be familiar with building Renderers and topics such as source paths, target paths and marshallers. These are covered in the Curam Widget Development Guide. This section assumes you have a good working knowledge of the renderers.

Start with some simple definitions which you should already be familiar with from the Curam Widget Development Guide

- **Renderer**
  A Java class that generates HTML markup.
- **Marshaller**
  A Java class used to access properties of a server interface and pre-processes data retrieved from a field
- **Source Path**
  A pointer used when accessing server interface properties.
- **Target Path**
  A pointer used for accessing the content of form fields.

In this example you create a simple text filter that filters by movie title. To create a new filter you are going to...

- Create a Pod filter Renderer
  - Map the source path

- Map the target path
- Create the text box

- Create a configuration for the Pod filter Renderer.
- Update our movies PodLoader.
  - Create a new PodFilter that uses our new filter Renderer.

Create a Pod filter Renderer

The related information contains a program listing for a `PodTextFilterRenderer`. Add this class to your component in the webclient project in a package that is named *`sample`* under the *`javasource`* folder.

Below we step through the important code.

Related reference

Sample: PodTextFilterRenderer for new Pod filter example on page 41

Preparing to delegate

You start our Renderer by creating a `FieldBuilder`. You do this because our Renderer is not going to do all the work. It delegates the task of rendering the input box to an existing Renderer. The `FieldBuilder` stores up the settings that you pass to that Renderer.

Setting up a FieldBuilder

```
025      Field field = ((Field)component);
026
027      final FieldBuilder fieldBuilder =
028        ComponentBuilderFactory.createFieldBuilder();
029      fieldBuilder.copy(field);
```

Setting a source path

In the following code extract, you extend the source path received to access the text for the filter. The text is stored in a Document Node named *text-filter* (you create that later in the PodLoader). You use the data accessor to retrieve the text that is added to the input box.

Setting the source path

```
032      String sourcePathExt = "text-filter";
033      Path sourcePath =
034        field.getBinding().getSourcePath().extendPath(sourcePathExt);
035      fieldBuilder.setSourcePath(sourcePath);
```

Setting a target path

Next, you extend the target path. You need to extend the target path to ensure the form item value is processed by the Marshaller attached to the Pod-Container. The Marshaller is configured to

process a number of specific target paths. The following example shows how to extend the target path in the correct format.

To set the target path:

```
038      String targetPathExt =
039        "choice/" + field.getID() + "/selected-options";
040      Path targetPath =
041        field.getBinding().getTargetPath().extendPath(targetPathExt);
042      fieldBuilder.setTargetPath(targetPath);
```

Note: The `PodFiltersRenderer` passes an Id value to the Renderer it invokes. The Id is the concatenation of a *podID* and *filterID* in the format *podID/filterID. The Id value is retrieved by the called renderer using the* `getID()` *method. That renderer uses the Id to uniquely identify itself.*

Format of a Pod filter target path

```
choice/ podId
         /
         filterId
          /selected-options
         /option-value
          |--1--|
         --2--|
         ---3----|
          --------4-------|
         -----5------|
```

The extended target path is broken in to what are known as steps which are divided by the '/' character. Each step in our target path is defined here.

Table 8: Target Path break down

Step	Description
1	This acts as the marker for the marshal. The *'choice'* text indicates that this field is to be processed by the Pod-Container.
2	Contains the unique identifier (as specified in the PodType codetable) for the Pod to which the filter is attached. For example, PT9001
3	Contains the unique identifier for the filter attached to the Pod. This Id is created when the `PodFilter` is constructed in the PodLoader.
4	The *selected-options* step indicates that this is a filter. Knowing this, the infrastructure processes the form values as a Pod filter.
5	The *option-value* step is optional and is used to uniquely identify selections in multi-select filters. For example, a checkbox filter can select more than 1 value, so each option gets an *option-value* step to distinguish it from its siblings.

In our code extract, you extended the target path using the id passed from the `PodFiltersRenderer` to map our text input form item. At runtime its value will be...

Format of a target path for My Favourite Movies Pod text filter

```
choice/PT9001/title/selections
```

Creating the input field

The last section of the renderer creates the input field.

It actually delegates the task to an existing Renderer which can create the input field. The `TextRenderer` is mapped to the *TEXT_NODE* Domain, so you simply set the Domain on our `FieldBuilder` instance and call the `render` function on that. The `TextRenderer` creates the form item and returns the input box which is appended to the HTML document.

Rendering the input box

```
045      fieldBuilder.setDomain(context.getDomain("TEXT_NODE"));
046      DocumentFragment textFilter =
047        fragment.getOwnerDocument().createDocumentFragment();
048      context.render(fieldBuilder.getComponent(), fragment, contract);
049
050      fragment.appendChild(textFilter);
```

Create a configuration for the Pod filter Renderer

In the StylesConfig.xml in your component, add the following entry. The 'style' name is used in the PodLoader to configure the `PodFilter` to use a new `PodTextFilterRenderer`.

You need to execute the build target for the client to add this configuration.

Style configuration for Pod filter Renderer:

```
<sc:style name="pod-text-filter">
    <sc:plug-in
      class="sample.PodTextFilterRenderer"
      name="component-renderer"
    />
  </sc:style>
```

Create a new PodFilter in the PodLoader

After creating a filter, all that remains is to start it in the PodLoader and use the saved value to filter the list of Movies. The related information contains the updated version of the `createPod(...)` method.

The code extract here shows the specific code that creates the text filter and adds it to the Pod.

Adding the Pod Text filter:

```
009      // Create the configuration for the filter renderer.
010      RendererConfig titleFilterRenderer =
011        new RendererConfig(RendererConfigType.STYLE, "pod-text-filter");
012
013      // Create the filter.
014      PodFilter titleFilter =
015        new PodFilter("title", document, titleFilterRenderer);
016      titleFilter.addFilterLabel("Title");
017
018      // Retrieve the saved filter value and extract to an array
019      Node titleTextNode =
020        getPodFilterById(PODTYPE.MYFAVMOVIES, "title", document);
021      ArrayList<String> titleTextArray =
022        PodFilter.convertSelectionsNodeToArrayList(titleTextNode);
023
024      // Create the Node that the filter Renderer expects and add the
025      // saved filter text to it.
026      String titleFilterText = "";
027      if (!titleTextArray.isEmpty()) {
028        titleFilterText = titleTextArray.get(0);
029      }
030      Element titleFilterNode = document.createElement("text-filter");
031      titleTextNode = document.createTextNode(titleFilterText);
032      titleFilterNode.appendChild(titleTextNode);
033      titleFilter.addFilter(titleFilterNode);
034
035      // Add the title filter to the Pod
036      moviesPod.addFilter(titleFilter);
```

Create a new filter:

In lines 10-11, create the configuration for the new filter by referencing the style that was created in the StylesConfig.xml. You pass this to the PodFilter constuctor along with the id of the filter, 'title' in this case.

Retrieve saved filter values:

In lines 19-22, use the utility functions to return the saved values for the 'title' filter and convert them to an array for ease of use.

Create input to Renderer:

In lines 19-33, create the text Node that is passed to our Renderer. The Renderer is expecting a Node named "text-filter" so you create this and add the filter text to it. You add the Node to our `PodFilter` object using the `addFilter(...)` method.

Add the filter to the Pod:

Finally, pass the `PodFilter` object to the `addFilter(...)` method of our `PodBuilder` object.

When you iterate over the movies, you only select movies whose title contains the substring that was returned from the filter. When you put it all together you can load the Pod, select the pen icon to open the filter, choose a genre and click save. The page redraws with the new filtered list.

Related reference

Sample: My Favorite Movies Pod-Loader for new Pod filter on page 42

1.8 Localization in Pods

In this section you are going to look at building Pods in a localizable manner. The examples that are provided use non-locale-specific properties file, these can be supplemented with locale-specific versions to return translated text if required. The Curam Widget Development Guide has a Chapter on Internationalization and Localization for widgets which covers this topic in more detail and the Curam Regionalization Guide discusses building a locale aware product.

To demonstrate the features built into the framework of Pods to support localization we will update our movies Pod to source various fields from property resources.

The textresource property

For each of the existing renderers that are used with Pods a 'textresource' attribute can be set that defines a resource property file. The code extract in the example shows a renderer reading a property from a text resource file. The file name is passed in the XML received by the renderer. (Refer to the example).

A Renderer reading a property from a text resource file:

```
private static final String RESOURCE_FILE_PATH = "@textresource";
  ...
  ...
  String textResource = context.getDataAccessor().get(
    field.getBinding().getSourcePath().extendPath(
      RESOURCE_FILE_PATH));
  Path textPath =
    ClientPaths.GENERAL_RESOURCES_PATH.extendPath(textResource);
  ...
  ...
  final String saveButtonText =
    context.getDataAccessor().get(
      textPath.extendPath("button.save.text"));
```

In the example above the Renderer is expecting to receive the name of the text resource file in the 'textresource' attribute of the document Node it receives.

Example of a document Node input to a Pod renderer

```
<pod textresource="sample.i18n.MyFavouriteMovies" ...>
    <config>
      ...
    </config>
    <data>
      ...
    </data>
  </pod>
```

The Renderer uses the `ClientPaths` class to create a pointer to the text resource file. The value of the property is retrieved by extending the path into the file to point at the specific property. The path extension is the property key. The value that is returned is the property value. If the request is made for a specific locale, and the resource file for that locale is provided then ClientPaths class accesses the property in the appropriate resource file.

MyFavouriteMovies.properties

```
pod.title=My Favourite Movies
  pod.filter.genre.label=Genre
  ....
```

The location of the properties file must be on the classpath of the client project. Adding the properties file to the javasource folder achieves this. The convention is to add property files to a folder called i18n to differentiate them.

Setting the text resource

A number of Renderers for producing standard content types in Pods are provided. Each of these Renderers has an associated Builder class that acts as an API for the Renderer to simplify the task of generating content to pass to the Renderer.

Table 9: Builders & Renderers

Builder	Renderer
PodBuilder	PodBodyRenderer
ListBuilder	ListBodyRenderer
PodListBuilder	PodListBodyRenderer
LinkBuilder	LinkRenderer
PodBuilder	PodBodyRenderer

The builder classes provide a `setTextResource(String)` method. At run time each instance of the Renderer uses the properties file received in the 'textresource' attribute to retrieve values that can be localized. Refer to the next section.

Localizing the My Favorite Movies Pod

In this section you update the Movies Pod to read the values from properties files instead of using hardcoded Strings. Start with a simple example, localizing the Pod title. You create a properties file with a title property and then update the PodBuilder to reference this property.

Note: The full listing for the createPod method for all examples that follow can be found in the related reference information.

Related reference

Sample: My Favourite Movies Pod-Loader for localization on page 43

Localizing the Pod

Create a new file called *`MyFavouriteMovies.properties`* in a folder called 'i18n' under the javasource/sample folder in the webclient project (If you have not already created that folder

you can do so now). In the file add the key *pod.title* with the value 'My Top Movies' which will distinguish it from the current title.

MyFavouriteMovie.properties:

```
pod.title=My Top Movies
```

Update the code used to construct our Pod by setting a text resource and use the property key for the title of the Pod.

MyFavouriteMovies.java, sourcing the Pod title from a properties file

```
005      moviesPod.setTextResource("sample.i18n.MyFavouriteMovies");
006      moviesPod.setTitle("pod.title");
```

Compile the PodLoader class, build the client target and launch the application. When the Pod is loaded you will see the new title "My Top Movies" which has been read from the properties file.

Now we have a localizable Pod title.

Localizing the filter

Next, you add localizable text to the filter labels. The Pod filter is tied to the Pod so it inherits the same resource file that you give to the Pod. In the same way that you did for the Pod title, you use a property key for the labels and add the property value to the properties file.

MyFavouriteMovies.properties:

```
pod.title=My Top Movies
  pod.filter.title.label=Movie Title:
  pod.filter.genre.label=Select Genre:
```

MyFavouriteMovies.java, by using the properties file for labels

```
...
017      titleFilter.addFilterLabel("pod.filter.title.label");
  ...
078      genreFilter.addFilterLabel("pod.filter.genre.label");
  ...
```

When you load the Pod you see that the label on the filter is changed to the value specified in the properties file.

Localizing the movie list

Take one more example. This time you use a properties file with a list of movies. To do so, you add a title to the list that is sourced from a properties file.

- Create a new properties file MoviesList.properties and add it to the i18n folder.
- Build the client to publish the properties.
- Update the list to use the properties file and add a column title as a property key. See the example.

Adding a column title

```
091      myFavouriteMovies.setTextResource("sample.i18n.MoviesList");
092      myFavouriteMovies.addColumnTitle(1, "list.col1.title");
```

Sharing properties files

The last example of localizing the list illustrates the value of sharing properties files. If you think about how a Pod is made up of various widgets, the complexity of which might extend to any number of widgets, then having 1 property file per widget would be difficult to maintain. For this reason, it makes sense to share the properties files for aggregated widgets such as Pods even though it is not technically necessary to do so.

In the example here, instead of creating a new properties file for the movies list widget, you can reuse the MyFavouriteMovies.properties file. Using this technique you have a single resource for all properties that are associated with the 'MyFavouriteMovies' Pod.

1.9 Sample program listings

This section contains the sample program listings for the My favorite movies pod.

Sample: The movies DB: A Java class serving our favorite movies

This class is the helper for the examples. It is a simple read-only Java DB for our favorite Movies.

```
package pods.podloaders;

  import java.util.Collection;
  import java.util.TreeMap;

  /** Simple read-only Java DB for a movie collection */
  public class MoviesDB {

    private TreeMap<Integer, Movie> allMovies;

    /** Constructor */
    public MoviesDB() {

      allMovies = new TreeMap<Integer, Movie>();
      allMovies.put(1, new MoviesDB.Movie(1,"The Dark Knight", "action",
        2008, "Christopher Nolan", "Christian Bale", 1));
      allMovies.put(2, new MoviesDB.Movie(2,"Casablanca", "romance",
        1942, "Michael Curtiz", "Humphrey Bogart", 3));
      allMovies.put(3, new MoviesDB.Movie(3,"Schindler's List", "drama",
        1993, "Steven Spielberg", "Liam Neeson", 7));
      allMovies.put(4, new MoviesDB.Movie(4,"Alien", "horror",
        1979, "Ridley Scott", "Sigourney Weaver", 1));
      allMovies.put(5, new MoviesDB.Movie(1, "The GodFather, Part II",
        "drama", 1974, "Francis Ford Coppola", "Marlon Brando", 6));
      allMovies.put(5, new MoviesDB.Movie(1, "Toy Story 3",
       "comedy", 2010, "Lee Unkrich", "Tom Hanks", 2));
      allMovies.put(6, new MoviesDB.Movie(6, "Toy Story 2",
          "comedy", 1999, "John Lasseter", "Tom Hanks", 0));

    }

    /** Return all movies as a Collection */
    public Collection<MoviesDB.Movie> getAllMovies(){
      Collection<MoviesDB.Movie> movieCollection =
        this.allMovies.values();
      return movieCollection;
    }
    /** Return a movie by its Id */
    public Movie getMovieById(Integer id) {
      return allMovies.get(id);
    }

    class Movie {

      public int id,year,oscars;
      public String title, genre, director, leadrole, url;

      public Movie(int id,String title,String genre,
        int year,String director,String leadrole, int oscars){
       this.id = id;
       this.title = title;
       this.genre = genre;
       this.year = year;
       this.director = director;
       this.leadrole = leadrole;
       this.oscars = oscars;

      }
    }
  }
```

Sample: Hello World Pod-Loader

This is the simplest Pod-Loader you can have

```
001  package pods.podloaders;
002
003  import java.util.Map;
004  import org.w3c.dom.Document;
005  import org.w3c.dom.Node;
006  import curam.cefwidgets.docbuilder.impl.PodBuilder;
007  import curam.cefwidgets.pods.pod.impl.PodLoader;
008  import curam.codetable.PODTYPE;
009
010  public class HelloWorld extends PodLoader {
011
012    @Override
013    public Node createPod(Document document, Map<String,Object> contexts) {
014      try{
015        PodBuilder helloWorld =
016          PodBuilder.newPod(document, PODTYPE.HELLOWORLD);
017        helloWorld.setTitle("Hello World");
018        return helloWorld.getWidgetRootNode();
019      }catch(Exception e){
020        throw new RuntimeException(e);
021      }
022    }
023  }
```

Sample: My favorite movies Pod-Loader

This version of the createPod method creates a list of movies using the MoviesDB class

```
001  public Node createPod(Document document, Map<String,Object> contexts) {
002    try{
003      PodBuilder moviesPod =
004        PodBuilder.newPod(document, PODTYPE.MYFAVMOVIES);
005      moviesPod.setTitle("My Favourite Movies");
006
007      MoviesDB moviesDB = new MoviesDB();
008
009      Collection<MoviesDB.Movie> favMovieCollection =
010        moviesDB.getAllMovies();
011      Iterator<MoviesDB.Movie> movieList =
012        favMovieCollection.iterator();
013
014      // Create the list
015      ListBuilder myFavouriteMovies =
016        ListBuilder.createList(1, document);
017
018      int row = 1;
019      while(movieList.hasNext()) {
020        Movie movie = movieList.next();
021        String movieName = movie.title;
022        myFavouriteMovies.addRow();
023        myFavouriteMovies.addEntry(1, row++, movieName);
024      }
025
026      RendererConfig contentRenderer = new RendererConfig(
027          RendererConfigType.STYLE, "single-list");
028      moviesPod.addContent(myFavouriteMovies, contentRenderer);
029
030      return moviesPod.getWidgetRootNode();
031    }catch(Exception e){
032      throw new RuntimeException(e);
033    }
```

Sample: My Favourite Movies Pod-Loader for Pod filter

This version of the createPod method adds a filter to the Movies Pod.

```
001  public Node createPod(Document document, Map<String,Object> contexts) {
002    try{
003      PodBuilder moviesPod =
004        PodBuilder.newPod(document, PODTYPE.MYFAVMOVIES);
005      moviesPod.setTitle("My Favourite Movies");
006
007      MoviesDB moviesDB = new MoviesDB();
008
009      // Create the configuration for the drop down filter.
010      RendererConfig filterRenderer =
011        new RendererConfig(RendererConfigType.DOMAIN, "CT_CHOICE");
012
013      // Create the PodFitler
014      PodFilter genreFilter =
015        new PodFilter("genre", document, filterRenderer);
016
017      // Create genre list
018      HashMap<String, String> genres = new HashMap<String, String>();
019      genres.put("all", "- All -");
020      genres.put("horror", "Horror");
021      genres.put("drama", "Drama");
022      genres.put("romance", "Romance");
023      genres.put("comedy", "Comedy");
024      genres.put("action", "Action");
025
026      // Create the options and selections using the ChoiceBuilder.
027      ChoiceBuilder choices =
028        ChoiceBuilder.newInstance(genres, document);
029
030      // Return the last saved selection for the filter with id "genre".
031      Node genreSelectionNode =
032        getPodFilterById(PODTYPE.MYFAVMOVIES, "genre", document);
033
034      // Convert the Node to an ArrayList.
035      ArrayList<String> selectedGenres =
036       PodFilter.convertSelectionsNodeToArrayList(genreSelectionNode);
037
038      // Create a default genre selection.
039      if (selectedGenres.isEmpty()){
040        selectedGenres.add("all");
041      }
042      choices.addSelection(selectedGenres.get(0));
043      choices.setTypeOfDisplay("listdropdown");
044
045      genreFilter.addFilter(choices.getWidgetRootNode());
046
047      // Add a filter label
048      genreFilter.addFilterLabel("Genre");
049      genreFilter.addCSSClasses("genre-filter");
050      moviesPod.addFilter(genreFilter);
051
052
053      Collection<MoviesDB.Movie> favMovieCollection =
054        moviesDB.getAllMovies();
055      Iterator<MoviesDB.Movie> movieList =
056        favMovieCollection.iterator();
057
058      // Create the list
059      ListBuilder myFavouriteMovies =
060        ListBuilder.createList(1, document);
061
062      int row = 1;
063      while(movieList.hasNext()) {
064        Movie movie = movieList.next();
065        String movieName = movie.title;
066        String selectedGenre = selectedGenres.get(0);
067        if (selectedGenre.equals(movie.genre)
068            || selectedGenre.equals("all")){
069
070          myFavouriteMovies.addRow();
071          myFavouriteMovies.addEntry(1, row++, movieName);
072        }
073      }
074
075      RendererConfig contentRenderer = new RendererConfig(
076          RendererConfigType.STYLE, "single-list");
077      moviesPod.addContent(myFavouriteMovies, contentRenderer);
078
079      return moviesPod.getWidgetRootNode();
```

Sample: PodTextFilterRenderer for new Pod filter example

The following renderer creates the text filter that you use to create new filters for Pods:

```
001  package sample;
002
003  import org.w3c.dom.DocumentFragment;
004  import curam.util.client.ClientException;
005  import curam.util.client.model.Component;
006  import curam.util.client.model.ComponentBuilderFactory;
007  import curam.util.client.model.Field;
008  import curam.util.client.model.FieldBuilder;
009  import curam.util.client.view.RendererContext;
010  import curam.util.client.view.RendererContract;
011  import curam.util.common.path.DataAccessException;
012  import curam.util.common.path.Path;
013  import curam.util.common.plugin.PlugInException;
014  import curam.widget.render.infrastructure.AbstractComponentRenderer;
015
016  /**
017   * Creates a text input for use with a Pod Filter
018   */
019  public class PodTextFilterRenderer extends AbstractComponentRenderer {
020
021    public void render(Component component, DocumentFragment fragment,
022      RendererContext context, RendererContract contract)
023      throws ClientException, DataAccessException, PlugInException {
024
025      Field field = ((Field)component);
026
027      final FieldBuilder fieldBuilder =
028        ComponentBuilderFactory.createFieldBuilder();
029      fieldBuilder.copy(field);
030
031      // Update the source path to point at the text node
032      String sourcePathExt = "text-filter";
033      Path sourcePath =
034        field.getBinding().getSourcePath().extendPath(sourcePathExt);
035      fieldBuilder.setSourcePath(sourcePath);
036
037      // Update the target path to use the Pod filter id
038      String targetPathExt =
039        "choice/" + field.getID() + "/selected-options";
040      Path targetPath =
041        field.getBinding().getTargetPath().extendPath(targetPathExt);
042      fieldBuilder.setTargetPath(targetPath);
043
044      // Use TextRenderer to create input box
045      fieldBuilder.setDomain(context.getDomain("TEXT_NODE"));
046      DocumentFragment textFilter =
047        fragment.getOwnerDocument().createDocumentFragment();
048      context.render(fieldBuilder.getComponent(), fragment, contract);
049
050      fragment.appendChild(textFilter);
051    }
052  }
```

Sample: My Favorite Movies Pod-Loader for new Pod filter

This version of the create Pod method includes the creation of the movie title filter

```
001  public Node createPod(Document document, Map<String,Object> contexts) {
002    try{
003      PodBuilder moviesPod =
004        PodBuilder.newPod(document, PODTYPE.MYFAVMOVIES);
005      moviesPod.setTitle("My Favourite Movies");
006
007      MoviesDB moviesDB = new MoviesDB();
008
009      // Create the configuration for the filter renderer.
010      RendererConfig titleFilterRenderer =
011        new RendererConfig(RendererConfigType.STYLE, "pod-text-filter");
012
013      // Create the filter.
014      PodFilter titleFilter =
015        new PodFilter("title", document, titleFilterRenderer);
016      titleFilter.addFilterLabel("Title");
017
018      // Retrieve the saved filter value and extract to an array
019      Node titleTextNode =
020        getPodFilterById(PODTYPE.MYFAVMOVIES, "title", document);
021      ArrayList<String> titleTextArray =
022        PodFilter.convertSelectionsNodeToArrayList(titleTextNode);
023
024      // Create the Node that the filter Renderer expects and add the
025      // saved filter text to it.
026      String titleFilterText = "";
027      if (!titleTextArray.isEmpty()) {
028        titleFilterText = titleTextArray.get(0);
029      }
030      Element titleFilterNode = document.createElement("text-filter");
031      titleTextNode = document.createTextNode(titleFilterText);
032      titleFilterNode.appendChild(titleTextNode);
033      titleFilter.addFilter(titleFilterNode);
034
035      // Add the title filter to the Pod
036      moviesPod.addFilter(titleFilter);
037
038      // Create the configuration for the drop down filter.
039      RendererConfig filterRenderer =
040        new RendererConfig(RendererConfigType.DOMAIN, "CT_CHOICE");
041
042      // Create the PodFitler
043      PodFilter genreFilter =
044        new PodFilter("genre", document, filterRenderer);
045
046      // Create genre list
047      HashMap<String, String> genres = new HashMap<String, String>();
048      genres.put("all", "- All -");
049      genres.put("horror", "Horror");
050      genres.put("drama", "Drama");
051      genres.put("romance", "Romance");
052      genres.put("comedy", "Comedy");
053      genres.put("action", "Action");
054
055      // Create the options and selections using the ChoiceBuilder.
056      ChoiceBuilder choices =
057        ChoiceBuilder.newInstance(genres, document);
058
059      // Return the last saved selection for the filter with id "genre".
060      Node genreSelectionNode =
061        getPodFilterById(PODTYPE.MYFAVMOVIES, "genre", document);
062
063      // Convert the Node to an ArrayList.
064      ArrayList<String> selectedGenres =
065       PodFilter.convertSelectionsNodeToArrayList(genreSelectionNode);
066
067      // Create a default genre selection.
068      if (selectedGenres.isEmpty()){
069        selectedGenres.add("all");
070      }
071      choices.addSelection(selectedGenres.get(0));
072      choices.setTypeOfDisplay("listdropdown");
073
074      genreFilter.addFilter(choices.getWidgetRootNode());
075
076      // Add a filter label
077      genreFilter.addFilterLabel("Genre");
078      genreFilter.addCSSClasses("genre-filter");
079      moviesPod.addFilter(genreFilter);
```

Sample: My Favourite Movies Pod-Loader for localization

```
001  public Node createPod(Document document, Map<String,Object> contexts) {
002    try{
003      PodBuilder moviesPod =
004        PodBuilder.newPod(document, PODTYPE.MYFAVMOVIES);
005      moviesPod.setTextResource("sample.i18n.MyFavouriteMovies");
006      moviesPod.setTitle("pod.title");
007
008      MoviesDB moviesDB = new MoviesDB();
009
010      // Create the configuration for the filter renderer.
011      RendererConfig titleFilterRenderer =
012        new RendererConfig(RendererConfigType.STYLE, "pod-text-filter");
013
014      // Create the filter.
015      PodFilter titleFilter =
016        new PodFilter("title", document, titleFilterRenderer);
017      titleFilter.addFilterLabel("pod.filter.title.label");
018
019      // Retrieve the saved filter value and extract to an array
020      Node titleTextNode =
021        getPodFilterById(PODTYPE.MYFAVMOVIES, "title", document);
022      ArrayList<String> titleTextArray =
023        PodFilter.convertSelectionsNodeToArrayList(titleTextNode);
024
025      // Create the Node that the filter Renderer expects and add the
026      // saved filter text to it.
027      String titleFilterText = "";
028      if (!titleTextArray.isEmpty()) {
029        titleFilterText = titleTextArray.get(0);
030      }
031      Element titleFilterNode = document.createElement("text-filter");
032      titleTextNode = document.createTextNode(titleFilterText);
033      titleFilterNode.appendChild(titleTextNode);
034      titleFilter.addFilter(titleFilterNode);
035
036      // Add the title filter to the Pod
037      moviesPod.addFilter(titleFilter);
038
039      // Create the configuration for the drop down filter.
040      RendererConfig filterRenderer =
041        new RendererConfig(RendererConfigType.DOMAIN, "CT_CHOICE");
042
043      // Create the PodFitler
044      PodFilter genreFilter =
045        new PodFilter("genre", document, filterRenderer);
046
047      // Create genre list
048      HashMap<String, String> genres = new HashMap<String, String>();
049      genres.put("all", "- All -");
050      genres.put("horror", "Horror");
051      genres.put("drama", "Drama");
052      genres.put("romance", "Romance");
053      genres.put("comedy", "Comedy");
054      genres.put("action", "Action");
055
056      // Create the options and selections using the ChoiceBuilder.
057      ChoiceBuilder choices =
058        ChoiceBuilder.newInstance(genres, document);
059
060      // Return the last saved selection for the filter with id "genre".
061      Node genreSelectionNode =
062        getPodFilterById(PODTYPE.MYFAVMOVIES, "genre", document);
063
064      // Convert the Node to an ArrayList.
065      ArrayList<String> selectedGenres =
066       PodFilter.convertSelectionsNodeToArrayList(genreSelectionNode);
067
068      // Create a default genre selection.
069      if (selectedGenres.isEmpty()){
070        selectedGenres.add("all");
071      }
072      choices.addSelection(selectedGenres.get(0));
073      choices.setTypeOfDisplay("listdropdown");
074
075      genreFilter.addFilter(choices.getWidgetRootNode());
076
077      // Add a filter label
078      genreFilter.addFilterLabel("pod.filter.genre.label");
079      genreFilter.addCSSClasses("genre-filter");
080      moviesPod.addFilter(genreFilter);
081
```

Notices

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the Merative website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of Merative

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of Merative.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

Merative reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by Merative, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

MERATIVE MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Merative or its licensors may have patents or pending patent applications covering subject matter described in this document. The furnishing of this documentation does not grant you any license to these patents.

Information concerning non-Merative products was obtained from the suppliers of those products, their published announcements or other publicly available sources. Merative has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-Merative products. Questions on the capabilities of non-Merative products should be addressed to the suppliers of those products.

Any references in this information to non-Merative websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those

websites are not part of the materials for this Merative product and use of those websites is at your own risk.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

The licensed program described in this document and all licensed material available for it are provided by Merative under terms of the Merative Client Agreement.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to Merative, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. Merative, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. Merative shall not be liable for any damages arising out of your use of the sample programs.

Privacy policy

The Merative privacy policy is available at https://www.merative.com/privacy.

Trademarks

Merative ™ and the Merative ™ logo are trademarks of Merative US L.P. in the United States and other countries.

IBM®, the IBM® logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

Adobe™, the Adobe™ logo, PostScript™, and the PostScript™ logo are either registered trademarks or trademarks of Adobe™ Systems Incorporated in the United States, and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis.

Microsoft™, Windows™, and the Windows™ logo are trademarks of Microsoft™ Corporation in the United States, other countries, or both.

UNIX™ is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.