IBM Cúram Social Program Management

Cúram - Pod Developers Guide

Version 6.0.5

IBM Cúram Social Program Management

Cúram - Pod Developers Guide

Version 6.0.5

Note
Before using this information and the product it supports, read the information in "Notices" on page 43

Revised: May 2013

This edition applies to IBM Cúram Social Program Management v6.0 5 and to all subsequent releases unless otherwise indicated in new editions.

Licensed Materials - Property of IBM.

Contents

Figures

Tables

Chapter 1. Introduction

1.1 Purpose

This guide is a cookbook for Developers who want to create Pods. The guide will coach 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 will require knowledge of the widget development process.

1.2 Audience

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

1.3 Prerequisites

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

1.4 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)
Cúram User Experience Guidelines	Design guidelines for UI components including Pods. See Chapter 6: Home Pages and Pods

Chapter 2. A Technical Overview

2.1 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 additional features are activated, such as the ability to be re-positioned 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.

2.2 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 they load the page.

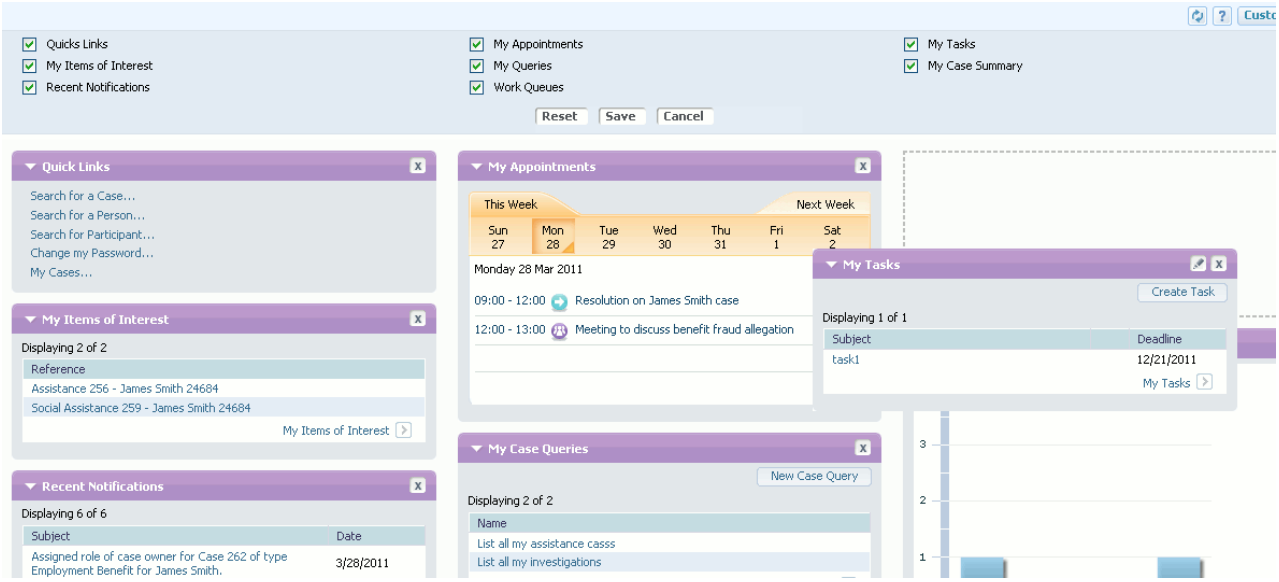

Figure 1. Pod-Container

2.3 How does it work?

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

2.3.1 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.

2.3.2 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 one of the save API's processes the data from the Pod-Container. The `PodContainer.vim` provides a reusable interface to the Pod infrastructure, simply add the `PodContainer.vim` to your UIM page and you will have a fully functioning interface.

2.3.3 PodLoader

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

2.3.4 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
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

2.3.5 Loading the Page

At the display phase the server interface invokes the `loadData()` method on the `PodContainer` class. The `PodContainer` uses the `PodContainerManager` to identify all the Pods to be displayed on the page using 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 invokes 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.

2.3.6 Rendering the page

The Rendering of the page is handled by a collection of Renderers. The rendering begins with the PodContainerRenderer which recevies 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 using markers in the data it receives. Each renderer returns its own content which it either generates itself or generates with the help of other renderers. This pattern of delegation is repeated unit all content is rendered.

The Curam Widget Development Guide discusses in detail the rendering framework and how renderers interact.

2.3.7 Saving the Page

At the action phase the server interface saves any changes the user made to the Pod selection and layout of the container back to the database again by way of 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).

2.3.8 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.

2.3.8.1 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 re-use of Pods across Pod containers.

Pod Height

The height of each Pod is determined by its content. A Pod's height will extend to display it's 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 you consider the type of Pods you will be 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.

2.3.9 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 of how to do this.

2.3.10 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 using the customization-console
- removing Pods using the customization console
- removing Pods 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 listed above a record of the current configuration of the page is saved. When the page reloads this saved configuration will be re-displayed.

2.3.11 Developing new Pods

In addition to re-using the Pods provided in the product an Organization may want to create new Pods. The Pod framework provides the ability to create new Pods with custom content. This guide will present examples of how this can be done.

Chapter 3. Getting Started

3.1 Introduction

Before creating a Pod we will 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.

3.2 Creating a page with a Pod container

Starting with a page that is mapped to a section and tab in the application (See WebClient reference guide for details of how to map pages in the application), add a Pod Container to the page by including the PodContainer.vim file as in the example below.

```
<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>
```

Figure 2. MyPodContainer.uim

3.3 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 will be used to uniquely identify the Pod page.

```
MyPodContainer=MyPodContainer
```

Figure 3. Constant.properties

3.4 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. 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>
```

Figure 4. PAGECONFIG.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>
```

Figure 5. USERPAGECONFIG.DMX

3.5 Testing the page

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

When our new Pod page loads it will be empty except for few buttons in the top right corner. The container is empty because we have not added any Pods to the page. Clicking the Customize button will open the customization-console. When the console opens it will be empty except for the action buttons. Again, because we have not assigned any Pods to the container there will be no Pods to select.

- The *Save* button will store the current users customizations.
- The *Reset* button will delete 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 chapter we will create a very simple Pod and add it to the container.

Chapter 4. Hello World Pod

4.1 Introduction

In this chapter we are going to create a basic Pod with a title and some text. We will also use the Admin Wizard to add our new Pod to our Pod page.

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

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

4.2 Declaring a new Pod

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

```
<?xml version="1.0" encoding="UTF-8"?>
<codetables package="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>
```

Figure 6. CT_PodType.ctx, declaring a 'Pod-Type'

4.3 Declaring a new PodLoader

Next we need to declare our PodLoader. The PodLoader is the java class that will generate the fragment of XML that will populate our Pod. The `CT_PodLoaderBindings.ctx` codetable entry binds a Pod-Type to a PodLoader. When the infrastructure processes our Pod it will look 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.

```
<?xml version="1.0" encoding="UTF-8"?>
<codetables package="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>
```

Figure 7. CT_PodLoaderBindings.ctx, declaring a 'PodLoader'

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

4.4 Creating a Pod using a PodLoader

The next step is to create our 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 the example below into a class named `HelloWorld` in the package pods.loaders.

```
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  }
```

Figure 8. A very simple PodLoader

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 paramtets to the Pods. Currently this is not support.

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 returned must match a predefined schema. The PodBuilder class provides an API to create our 'pod' Node in the correct format.

In the example 4.3 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 consturctor. 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.

4.5 Adding a Pod to the Pod Container

The last piece of the jigsaw is adding the Pod to the Pod-Container. We will use the wizard provided in the Administrator application. We must login to the Administrator application, so we will need the username and password assigned to this application. When we have logged in we 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 we can see the MyPodContainer page that we created in Chapter 2 in the list of Personal Pages. Selecting edit will open the wizard for maintaining our Personal Page. The first step lists all the Pods available for selection. In this list we find our Pod 'HelloWorld!'. Select the Pod and click next on the remaining steps finally saving the record. We have now added our Pod to our Pod Container. Log out of the Administrator application and log into the application that contains the Pod page.

4.6 Viewing the Pod

Now lets see our Pod in action. Login to the application and go to the Pod page. When the page loads it will be empty except for the buttons in the top right corner. Click the customize button to open the *customization-console*. We can see our Pod listed in the console. Select the checkbox beside the Pod and choose save. The page will reload with our Pod defaulted to the top right corner.

You will 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 chapter we will create another Pod with some content and take a closer look at the PodBuilder class.

4.7 Review

Lets take a look back on what we did.

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

In the next chapter we will create a new Pod with some more interesting content.

Chapter 5. Creating a Pod with a list

5.1 Introduction

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

Lets begin with a new Pod which we will add to our Pod-Container in the same way we added the Hello World! Pod in the previous chapter. We will use a movies theme for our examples, so lets create a Pod with a short list of our favourite movies.

5.2 Creating a new list Pod

5.2.1 Register new Pod

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

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

Figure 9. 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>pods.podloaders.MyFavouriteMovies</description>
      <annotation/>
    </locale>
  </code>
```

Figure 10. Adding PodLoader binding

5.2.2 Create a new PodLoader

Next we add our PodLoader class to our loaders package remembering to reference the new codetable value we created in the *PodType* codetable when we construct our new Pod using the PodBuilder.

```
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  }
```

Figure 11. Creating a PodLoader class

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

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

5.2.3 Create the list

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

To start we will need to provide the movies for our list. Appendix A.1 contains a full program-listing for a Java class that will act as a simple read-only DB of our favourite movies. Add this class to a package in the project where it can be accessed by our PodLoader.

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

```
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    }
```

Figure 12. Adding a list to a Pod

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

In the next section we will look in more detail at how the list was created.

5.3 Deconstructing the code

5.3.1 Constructing the list

A Pod does not need to know what its content will be. At runtime the Pod will delegate to other widgets to produce the HTML that will render the content. Our movies Pod is a list of movie names and it will re-use 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 runtime the `ListBodyRenderer` will translate this XML into the HTML that can be added to the body of a Pod. To build the Pod content for our Pod the PodLoader will use the ListBuilder to produce the list of movies.

The first step in creating our 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 our Pod will be added. The document object is used to create the new Nodes that represent our Pod and its content. Those Nodes will be appended to the some part of the document object.

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

5.3.2 Adding rows

Next we iterate over our movies. For each movie we 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      }
```

5.3.3 Creating content in the cells

We 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	Descrption
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 will process for display. (See Javadoc for ListBuilder)

In our movies Pod we want to add a list of movie names so we 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);
```

Figure 13. Add the movie names to our list

5.3.4 Adding the list to a Pod

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

The `addContent(...)` method provides the mechanism for adding our Pod content. The method accepts as it's 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 will create 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 will process the *content* parameter.

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

Figure 14. Adding the list to the Pod

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 our movies list is defined in the ListBuilder object which is added to our Pod. The ListBodyRenderer will generate the HTML table which will be appended to out Pod body.

Chapter 6. Adding a Pod filter

6.1 Introduction

In this chapter we will explore Pod filters. We will look at the existing filters available and we will use one to add a filter to our movies Pod.

6.2 What is a Pod filter?

A Pod can optionally include a Pod filter. The filter allows a user to refine the information 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 re-draw the Pod with a different chart representing the selected time period.

6.3 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 table below 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.

6.4 Adding a Drop Down Filter

To demonstrate the use of filters we are going to create a filter for our movies Pod. The filter will select movies by genre. As we did in the last chapter we will insert the complete code sample first to see the Pod in action, then we will step through the code to see what we did to create the filter.

Appendix A.4 contains a program listing, replace the createPod method in the `MyFavouriteMovies` PodLoader with the version from the Appendix. Compile the PodLoader and launch the Application.

When the page loads the Pod will be 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 will only return movies that match the selected genre in the dropdown.

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

6.4.1 Creating the Pod Filter

To add a filter to the Pod we 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 will be used to invoke the render that will create the filter. In our example we are creating a filter with the id "genre" and we will be using a renderer called the ChoiceRenderer to render the content of the 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);
```

Figure 15. Creating the Pod Filter

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

6.4.2 Creating the options

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

```
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);
```

Figure 16. Creating the set of Choices for the genre drop-down

6.4.3 Creating the selections

The next step is adding the selected values. In most cases you will want this to be the last saved selections. We 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 will return the selected values for each 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));
```

Figure 17. Retrieving the saved selections and adding them to the Pod filter

On line 32 we 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 we add the selected value, in this case it is the only value returned in the array.

6.4.4 Setting the type of filter

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

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

Figure 18. Setting the type of filter

6.4.5 Adding a label and CSS styling

Optionally we 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. Localization is covered in Chapter 8: Localization in Pods

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

Figure 19. Adding a PodFilter to a Pod

6.4.6 Add the Filter to the Pod

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

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

Figure 20. Adding a PodFilter to a Pod

6.4.7 Filtering your Pod

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

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

Figure 21. Filter the movies by genre

So that completes our filter. When the Pod is loaded for the first time no value will be stored for the filter. Every subsequent save will store the filter value, even if that is an empty String. When the Pod reloads it will use the saved value to filter the list of Movies, and it will also pass the stored value back to the filter for display so that we 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 us to reuse existing renderers. There will however be occasions where you want to create a custom filter type. In the next chapter we will demonstrate how to create a new filter renderer.

Chapter 7. Creating new Pod filters

7.1 Introduction

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

To complete this section you will need to create a Renderer so you will 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 chapter will assume you have a good working knowledge of the renderers.

We will 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 we will create a simple text filter that filters by movie title. To create our new filter we 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.

7.2 Create a Pod filter Renderer

Appendix A.3 contains a program listing for a `PodTextFilterRenderer`. Add this class to your component in the webclient project in a package named `sample` under the `javasource` folder. Below we will step through the important code

7.2.1 Preparing to delegate

We start our Renderer by creating a `FieldBuilder`. We do this because our Renderer isn't going to do all the work. It will delegate the task of rendering the input box to an existing Renderer. The `FieldBuilder` will store up the settings that we pass to that Renderer.

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

Figure 22. Setting up a FieldBuilder

7.2.2 Setting a source path

In the code extract in Example 7.2 we have extended the source path received to access the text for the filter. The text will be stored in a Document Node named *text-filter* (we will create that later in the PodLoader). We use the data accessor to retrieve the text that will be added to the input box.

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

Figure 23. Setting the source path

7.2.3 Setting a target path

Next we extend the target path. We 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. Example 7.3 shows how to extend the target path in the correct format.

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

Figure 24. Setting the target path

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 will use the Id to uniquely identify itself.*

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

Figure 25. Format of a Pod filter target path

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 below.

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.

Table 8. Target Path break down (continued)

Step	Description
2	Contains the unique identifier (as specified in the PodType codetable) for the Pod to which the filter is attached. E.g. 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 will process 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. E.g a checkbox filter can select more than 1 value, so each option gets an *option-value* step to distinguish it from it's siblings.

In our code extract in Example 7.3 we have extended the target path using the id passed from the `PodFiltersRenderer` to map our text input form item. At runtime its value will be...

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

Figure 26. Format of a target path for My Favourite Movies Pod text filter

7.2.4 Creating the input field

The last section of our renderer creates the input field. It actually delegates the task to an existing Renderer which can create the input field for us. The `TextRenderer` is mapped to the *TEXT_NODE* Domain, so we simply set the Domain on our `FieldBuilder` instance and call the `render` function on that. The `TextRenderer` will create the form item and return the input box which is appended to the HTML document.

```
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);
```

Figure 27. Rendering the input box

7.3 Create a configuration for the Pod filter Renderer

In the StylesConfig.xml in your component add the following entry. The 'style' name will be used in our PodLoader to configure the `PodFilter` to use our new `PodTextFilterRenderer`. We will need to execute the build target for the client to add this configuration.

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

Figure 28. Style configuration for Pod filter Renderer

7.4 Create a new PodFilter in the PodLoader

Now that we have created our filter all that remains is to invoke it in our PodLoader and use the saved value to filter our list of Movies. Appendix A.6 contains the updated version of the `createPod(...)` method. The code extract below shows the specific code that creates our text filter and adds it to the Pod.

```
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);
```

Figure 29. Adding the Pod Text filter

Create a new filter

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

Retrieve saved filter values

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

Create input to Renderer

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

Add the filter to the Pod

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

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

Chapter 8. Localization in Pods

8.1 Introduction

In this chapter we are going to look at building Pods in a localizable manner. The examples 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.

8.2 The textresource property

For each of the existing renderers used with Pods a 'textresource' attribute can be set that defines a resource property file. The code extract in Example 8.1 shows a renderer reading a property from a text resource file. The file name is passed in the XML received by the renderer (See example 8.2)

```
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"));
```

Figure 30. A Renderer reading a property from a text resource file

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.

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

Figure 31. Example of a document Node input to a Pod renderer

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 returned is the property value. If the request is made for a specific locale, and the resource file for that locale has been provided then ClientPaths class will access the property in the appropriate resource file.

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

Figure 32. MyFavouriteMovies.properties

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

8.3 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 which 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 runtime each instance of the Renderer uses the properties file received in the 'textresource' attribute to retrieve values that can be localized. We'll see this in action in the next section.

8.4 Localizing the My Favourite Movies Pod

In this section we will update our Movies Pod to read the values from properties files instead of using hardcoded Strings. Lets start with a simple example, localizing the Pod title. We will create a properties file with a title property and then update our PodBuilder to reference this property.

Note: The full listing for the createPod method for all examples that follow can be found in Appendix A.8.

8.4.1 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 haven't 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.

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

Figure 33. MyFavouriteMovie.properties

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

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

Figure 34. MyFavouriteMovies.java, sourcing the Pod title from a properties file

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 Movied" which has been read from the properties file.

Now we have a localizable Pod title! Lets do some more...

8.4.2 Localizing the filter

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

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

Figure 35. MyFavouriteMovies.properties

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

Figure 36. MyFavouriteMovies.java, using the properties file for labels

When we load our Pod we will see that the label on the filter has changed to the value specified in the properties file.

8.4.3 Localizing the movie list

Lets take one more example. This time we are going to use a properties file with our list of movies. To do this we are going to add a title to the list which will be 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 Example 8.8)

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

Figure 37. Adding a column title

8.4.4 Sharing properties files

The last example of localizing the list illustrates the value of sharing properties files. If we think about how a Pod is made up of various widgets, the complexity of which could 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 same properties files for aggregated widgets such as Pods even though it is not technically necessary to do this.

In our example above, instead of creating a new properties file for the movies list widget, we can re-use the MyFavouriteMovies.properties file. Using this technique we have a single resource for all properties associated with the 'MyFavouriteMovies' Pod.

Appendix. Progam Listings

A.1 The Movies DB: A Java class serving our favourite 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;

      }
    }
  }
```

Figure 38. This class is the helper for our examples. It is a simple read-only Java DB for our favourite Movies. Feel free to add your personal favourites!

A.2 Chapter 4: Hello World Pod-Loader

```
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  }
```

Figure 39. This is the simplest Pod-Loader you can have

A.3 Chapter 5: My Favourite Movies Pod-Loader

```
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    }
```

Figure 40. This version of the createPod method creates a list of movies using the MoviesDB class

A.4 Chapter 6: My Favourite Movies Pod-Loader for Pod filter example

```
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      }
```

A.5 Chapter 7: PodTextFilterRenderer for new Pod filter example

```
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  }
```

Figure 42. This renderer will create the text filter that we use in Chapter 7 when we demonstrate how to create new filters for Pods

A.6 Chapter 7: My Favourite Movies Pod-Loader for new Pod filter example

```
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
```

A.7 Chapter 8: My Favourite Movies Pod-Loader for localization example

```
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
082
083       Collection<MoviesDB.Movie> favMovieCollection =
084         moviesDB.getAllMovies();
085       Iterator<MoviesDB.Movie> movieList =
086         favMovieCollection.iterator();
087
088       // Create the list
089       ListBuilder myFavouriteMovies =
090         ListBuilder.createList(1, document);
091       myFavouriteMovies.setTextResource("sample.i18n.MoviesList");
092       myFavouriteMovies.addColumnTitle(1, "list.col1.title");
093
094       int row = 1;
095       while(movieList.hasNext()) {
096         Movie movie = movieList.next();
097         String movieName = movie.title;
098         String selectedGenre = selectedGenres.get(0);
099         if (selectedGenre.equals(movie.genre)
100             || selectedGenre.equals("all")){
101
102           if (movieName.toUpperCase().indexOf(
103               titleFilterText.toUpperCase()) != -1) {
104             myFavouriteMovies.addRow();
105             myFavouriteMovies.addEntry(1, row++, movieName);
106           }
107         }
108       }
109
110       RendererConfig contentRenderer = new RendererConfig(
111           RendererConfigType.STYLE, "single-list");
112       moviesPod.addContent(myFavouriteMovies, contentRenderer);
113
114       return moviesPod.getWidgetRootNode();
115
116     }catch(Exception e){
117       throw new RuntimeException(e);
118     }
119   }
```

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law.

IBM Japan Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Dept F6, Bldg 1

294 Route 100

Somers NY 10589-3216

U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources.

IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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 the names and addresses used by an actual business enterprise is entirely coincidental.

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 IBM, 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. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/us/en/copytrade.shtml.

Java and all Java-based trademarks and logos are registered trademarks of Oracle and/or its affiliates.

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

Printed in USA