Adding Intelligence to Media

XMP FILEINFO SDK

PROGRAMMER'S GUIDE

April 2010

Adobe®

XMP FileInfo SDK Programmer's Guide.

Contents

Preface

The File Info dialog is a standard dialog that many Adobe® applications use to provide access to file metadata information. If you define your own metadata schema and properties using the Adobe XMP format, you can customize the dialog by creating and adding custom panels that appear beside the built-in panels and provide access to your metadata properties.

About this document

This document, the *XMP FileInfo SDK Programmer's Guide*, provides guidance for programmers wishing to define custom panels for the File Info dialog, which is shared among Adobe CS5 applications. A custom panel allows users to display and modify specific XMP metadata properties that are important to their workflow.

This software is provided by Adobe as a free Software Development Kit (SDK). This guide emphasizes tutorial material. The complete API reference is provided as separate HTML documentation in the SDK.

This document assumes that the reader is familiar with Adobe XMP and Flex® technology, and with Eclipse-based development environments.

For further information about XMP and Flex, see:

- http://www.adobe.com/devnet/xmp/
- http://www.adobe.com/devnet/flex/

How this document is organized

This document has the following sections:

- Chapter 1, "The XMP FileInfo SDK," gives an overview of the XMP FileInfo SDK, and of the sample panels that are provided with it.
- Chapter 2, "Defining Custom File Info Panels," provides a tutorial walkthrough that demonstrates how to set up an XMP Custom Panel project and create a custom panel using Adobe Flash® Builder™, a full-featured development environment.
- Chapter 3, "The Generic Panel Tool," describes how to use the Generic Panel tool to quickly and easily provide access to your own custom XMP schema.
- Chapter 4, "File Info Panel Concepts," provides details of how custom panels are specified using the XMP FileInfo SDK, and discusses related issues such as Flash security.
- Chapter 5, "Using the Flex SDK," describes how to build a simple File Info panel using the test editor, a free Flex SDK and Apache Ant open-source software build tool.
- Chapter 6, "Porting Guide," provides compatibility details between this release of the SDK and the previous release, to help you in porting existing applications.

Conventions used in this document

The following type styles are used for specific types of text:

Typeface Style	Used for:
`Monospaced Regular`	XML code and other literal values, such as value types and names in other languages or formats
`Monospaced bold`	Emphasis or points of interest in example code.
`Monospaced italic`	Placeholders or variables in code or paths, to be replaced by the user with appropriate values.

1 The XMP FileInfo SDK

This chapter introduces the XMP FileInfo Software Developers Kit (SDK) and the sample panels that it provides.

Overview

The File Info dialog is a standard dialog that many Adobe applications use to provide access to file metadata information. Some Adobe applications, however, including Adobe Bridge and audio-video applications such as Adobe Premiere® Pro, do not use the File Info dialog to display metadata. Instead, they have their own metadata palettes that rely on an XML representation of XMP properties.

XMP is, by definition, extensible; you can create custom properties in addition to those that are predefined in public schemas. If you define your own metadata schema and properties using the Adobe XMP format, there are three ways to extend the presentation of XMP metadata in Adobe applications, so that they show the custom properties you have defined:

1. You can customize the FileInfo dialog, using the SDK and Flash Builder to create and add custom panels that appear beside the built-in panels and provide access to your metadata properties. This document describes in detail how to do this.

   When you create a custom panel for the File Info dialog, you can simply associate components with your custom properties, and their values are displayed in exactly the same way as predefined properties.

   This is the most powerful and flexible way to present your custom metadata in those applications that support the FileInfo dialog.

2. Custom schema files, in an XML format, provide custom schemas to the Metadata panel in Adobe Bridge and in audio-video application such as Premiere, which do not support the File Info dialog.

   You can use this same XML representation to define your own XMP properties, and include it in your project, so that the values that are edited in your File Info panel are also made available to applications that do not use the File Info dialog. Deliver the schema file along with your custom panel definition in the delivery location for your platform, `. . . /Custom File Info Panels/3.0/custom/`. The applications that do not use the File Info dialog look in this location for the XML file, while ignoring the custom panel definitions.

3. Custom XMP properties that you define in XML can also be picked up dynamically and rendered automatically in the File Info dialog by the Generic Panel, a tool included with the XMP FileInfo SDK; see Chapter 3, "The Generic Panel Tool".

   The Generic Panel offers only limited support for properties defined in XML; it can display only simple XMP properties and comma-separated array lists, with limited layout capabilities. For more complex workflows, it is recommended that you create a custom panel using Flex components.

Using the XMP FileInfo SDK

Adobe provides the free XMP FileInfo SDK with documentation, tools, and sample code to help you get started with creating your own custom panels for the File Info dialog in Adobe Creative Suite® 5 applications.

▶ You can download the XMP FileInfo SDK from http://www.adobe.com/devnet/xmp/

This SDK is an independent companion to the general XMP Toolkit SDK (also available at the same location) which provides an API for working with XMP metadata programmatically. The XMP FileInfo SDK is used to extend CS5 applications, whereas the XMP Toolkit SDK is used to build XMP support into other products and applications. The API is not required for building custom panels, but the documentation, especially the *XMP Specification Part 1, Data and Serialization Model*, is recommended for understanding XMP models and concepts.

The XMP FileInfo SDK contains these folders and files (locations are relative to the download location, *SDKroot*):

`Docs/XMP FileInfo SDK.pdf`	This document.
`Docs/API/index.html`	The home page of the HTML reference documentation for the XMP component set.

`tools/`
`  com.adobe.xmp.sdk.fileinfo_1.0.2.jar`
`  com.adobe.xmp.sdk.fileinfo_fb4_1.1.0.jar`
	These JAR files contain the Custom Panel Eclipse plug-in for Flex Builder 3 and Flash Builder 4, or for their Eclipse plug-in versions. (For the standalone version of the Flex/Flash Builder, you must also install the Ant plug-in, which must be obtained separately.)
	
	To install the plug-in, place the JAR file in your `eclipse/plugins` or Flex/Flash Builder `plugins` folder; see "Installing and using the XMP Custom Panel plug-in" on page 10.

`  Generic/`	The Generic Panel, a ready-made panel that automatically generates the proper components to show a custom schema specified as XML. See Chapter 3, "The Generic Panel Tool."
`  FileInfoFoundation.swc` `  build.xml`	The standalone XMP component library and Ant build script, for use with the free Flex SDK (without Flash Builder); see Chapter 5, "Using the Flex SDK."

`Samples/`	Sample panels that make use of the XMP components; see "SDK sample panels" on page 11.
`panels-src/`	▶ The `panels-src` folder contains the Eclipse projects used to create the panels. You can import them into your Flash Builder workspace.
`panels/`	▶ The `panels` folder contains the complete example panels, the Flash files that result from compiling the projects. You can add the sample panels directly to the user custom-panel folder for your platform an use them in Creative Suite 5: **WINDOWS XP:** `C:\Document and Settings\<username>\Application Data\Adobe\XMP\Custom File Info Panels\3.0\panels` **WINDOWS 7/VISTA:** `C:\Users\<username>\AppData\Roaming\Adobe\XMP\Custom File Info Panels\3.0\panels` **MAC OS:** /Users/*<username>*/Library/Application Support/Adobe/XMP/Custom File Info Panels/3.0/panels

Prerequisites

To use the XMP FileInfo SDK and sample code, you must have the following:

- A Flex development environment, such as Adobe Flash Builder, or Eclipse with the Flash Builder plug-in, is recommended. See "The panel development environment" on page 10.

  —or—

  You can use the free Flex SDK with any development environment of your choice; see Chapter 5, "Using the Flex SDK."

- An installation of Apache Ant, either standalone, or as a plug-in for your development environment.
  - For information on Apache Ant, see http://ant.apache.org/.
  - You can obtain the Apache Ant Eclipse plug-in from http://sourceforge.net/projects/r2tech-eclipse/.

- The XMP Custom Panel plug-in for your development environment. This is provided with the XMP FileInfo SDK. See "Installing and using the XMP Custom Panel plug-in" on page 10.

  —or—

  If you choose to use the free Flex SDK, you must use the standalone component library and Ant build file; see Chapter 5, "Using the Flex SDK."

- An installed Adobe Creative Suite 5 application that contains the File Info dialog, such as Adobe Bridge CS5 or Adobe Photoshop® CS5.
- If you plan to create File Info panels for CS4 (or earlier), read Chapter 6, "Porting Guide".

The panel development environment

The File Info dialog is based on Adobe Flex™ technology. To extend it, you can use Adobe Flash Builder, a sophisticated development environment which is available as a standalone application or as an Eclipse plug-in. At minimum, you can use the Flex SDK, which is available from Adobe free of charge. For information on obtaining and using these tools, see http://www.adobe.com/devnet/flex/.

You develop the source file for a panel definition using MXML and ActionScript® with the class libraries provided by the Custom Panel plug-in, part of the XMP FileInfo SDK.

MXML is an XML language that you use to lay out user-interface components for Flex applications. MXML tags correspond to ActionScript classes or properties of classes. When you compile your Flex application, Flex parses your MXML tags and generates the corresponding ActionScript classes. It then compiles these ActionScript classes into SWF byte code which it stores in a Flash® (SWF) file.

The XMP FileInfo SDK provides an Apache Ant build file for compiling your panel definitions without the use of Flash Builder. To use it, you must configure the development environment with this open-source software build tool.

Installing and using the XMP Custom Panel plug-in

The XMP FileInfo SDK includes a component library (`FileInfoFoundation.swc`) that extends the basic Flex component set with XMP components that are specialized for use in panels within the File Info dialog (see Chapter 4, "File Info Panel Concepts").

The XMP component library is provided along with an XMP Project Wizard as a Java archive; there are separate JAR files for Flex Builder 3 and Flash Builder 4:

SDKroot`/tools/com.adobe.xmp.sdk.fileinfo_1.0.2.jar` (Flex Builder 3)
SDKroot`/tools/com.adobe.xmp.sdk.fileinfo_fb4_1.1.0.jar` (Flash Builder 4)

Place the appropriate JAR file in your `eclipse/plugins` or Flash/Flex Builder `plugins` folder. (The library can also be used standalone with the free Flex SDK; see Chapter 5, "Using the Flex SDK.")

The plug-in defines a new Flex project type, XMP Custom Panel, which contains the XMP library. To create a project of this type:

1. Start the Eclipse or Flash Builder IDE.
2. Create a new project (**File > New > Project**, or right-click in Flex Navigator and choose **New > Project**).
3. If the plug-in has been installed correctly, **XMP** appears in the list of project types. Expand **XMP** and choose **Custom Panel**.
4. Complete the Project Wizard steps to create a new Flex Project.
5. Edit the `build/build.properties` file to make sure that all locations reflect your installation, if it is different from the defaults. See details in "Getting started" on page 21.

The XMP FileInfo SDK also includes sample panels to help you get started using the specialized component set; see "SDK sample panels" on page 11.

SDK sample panels

The sample panels that are provided with the XMP FileInfo SDK are described in the following sections:

- "BasicControlSample panel" on page 11 demonstrates all of the XMP-aware components that are part of the SDK and than can be used in a custom panel.
- "Language panel" on page 15 demonstrates the use of XMP event handlers, and XMP file I/O.
- "YahooSearchSample panel" on page 16 demonstrates the use of an external service, accessed using HTTP.
- "FlashFlex panel" on page 18 demonstrates how to embed a previously-defined Flash file or previously-defined Flex component into a panel definition. It also shows how to extend the XMP component set to create customized components.

Installing the samples

You can find the sample code in the `samples` folder of the root location where you downloaded the XMP FileInfo SDK. There are two subfolders:

- The `panels-src` folder contains the Eclipse/Flash Builder projects that were used to develop the sample panels. You can import these into your Eclipse or Flash Builder workspace (**File > Import > Flex Project**). See "Installing and using the XMP Custom Panel plug-in" on page 10.
- The `panels` folder contains the compiled Flash files for the sample panels. To see the panels in the File Info dialog of an installed Adobe application, add one or more of these subfolders directly to your XMP `panels` directory in the user's Adobe XMP application-data location for your platform:

  IN WINDOWS XP: `C:\Documents and Settings\<username>\Application Data\Adobe\XMP\Custom File Info Panels\3.0\panels\`

  IN WINDOWS 7/VISTA: `C:\Users\<username>\AppData\Roaming\Adobe\XMP\Custom File Info Panels\3.0\panels\`

  IN MAC OS: `/user/<username>/Library/Application Support/Adobe/XMP/Custom File Info Panels/3.0/panels/`

- Create a Flash trust file for the panels you have installed. This is a simple text file with a `.cfg` file extension, that identifies Flash files that have been determined to be safe. See "Trust files for custom panels" on page 53. Without a trust file, your panel does not appear in the File Info dialog.

BasicControlSample panel

This sample demonstrates the main components available for custom File Info panels. This is very much like the finished project as created by the tutorial example in Chapter 2, "Defining Custom File Info Panels".

The sample folder `BasicControlSample` contains these folders and files:

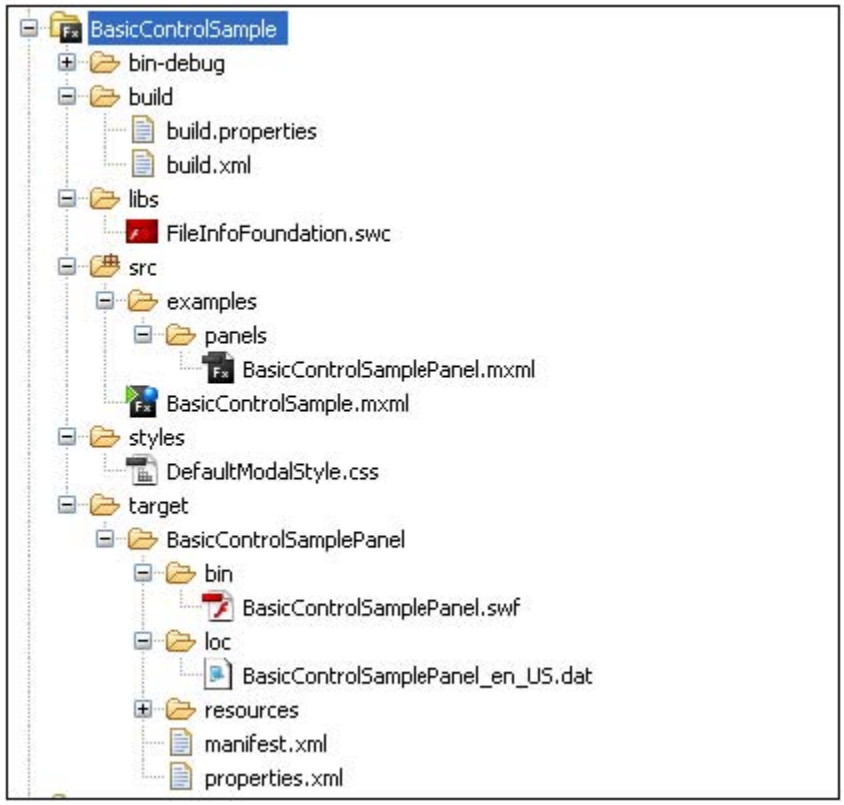
- `src/`: This folder contains the source files for the panel definition, including the following:

- `PanelTest.mxml`: A generated file that provides a testing environment within the development environment. When run, it displays a simple dialog that includes the new panel defined here.
- `examples/panels/BasicControlSamplePanel.mxml`: The source file for the panel definition. When you create a project with the Project Wizard, this is automatically supplied with some starter code that defines a basic panel with one form entry.

- `styles/`: This folder contains a style sheet that helps display the panel within the development environment, so that it replicates the look expected within the File Info dialog in a CS5 application. (You might see style-sheet warnings for the project, as not all available styles are used in the samples, and may not be used when developing your own panels.)
- `build/`: This folder contains the Apache Ant build script that allows you to compile the source MXML into a Flash file for delivery.
  - The `build.xml` file is the makefile that instructs Ant in how to compile the source.
  - The `build.properties` file is used to locate the Flex SDK, and publish the panel to the correct user folder. Note that within this file, all paths use forward slash (/) notation, regardless of operating system.

  NOTE: You can also use these to build a panel in a standalone Flex SDK environment; see Chapter 5, "Using the Flex SDK." You must keep these two files together.
- `target/BasicControlSample/`: This folder contains the delivery files. Everything in this folder is included in the export of this custom panel, including the following:
  - `manifest.xml`: The manifest that lists the contents of the panel definition, including, for example, the name of the panel. For details, see "Panel manifests" on page 47.
  - `bin/BasicControlSamplePanel.swf`: The compiled Flash movie (SWF) file for the panel.
  - `loc/BasicControlSamplePanel_en_US.dat`: A translation dictionary for localized strings (ZStrings) used in the panel.
  - `properties.xml`: Additional XMP metadata fields specific to this sample, defined in XML format. See "Showing custom XMP properties" on page 32.

The project looks like this when imported into Flash Builder:

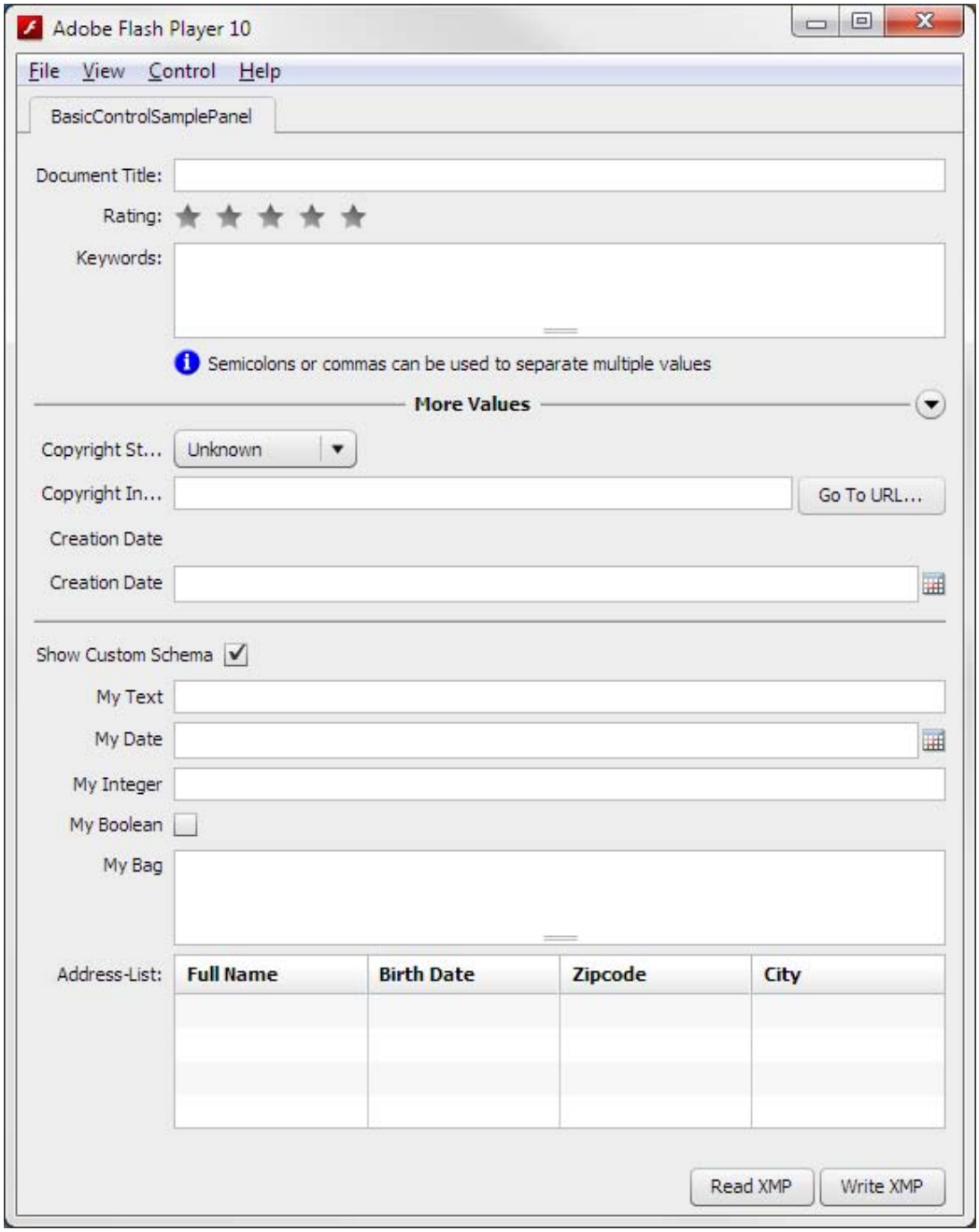


The source MXML file for the panel specifies the XMP components to display in the panel. Each component has an `xmpPath` property that associates it with an XMP metadata property.

The sample illustrates different types of controls that are suitable for displaying and controlling the values of different types of metadata fields with a variety of data types. Each component is capable of reading its associated value from the XMP Packet of the examined file, and writing it back if needed when the dialog is dismissed with the **OK** button. See details in Chapter 2, "Defining Custom File Info Panels".

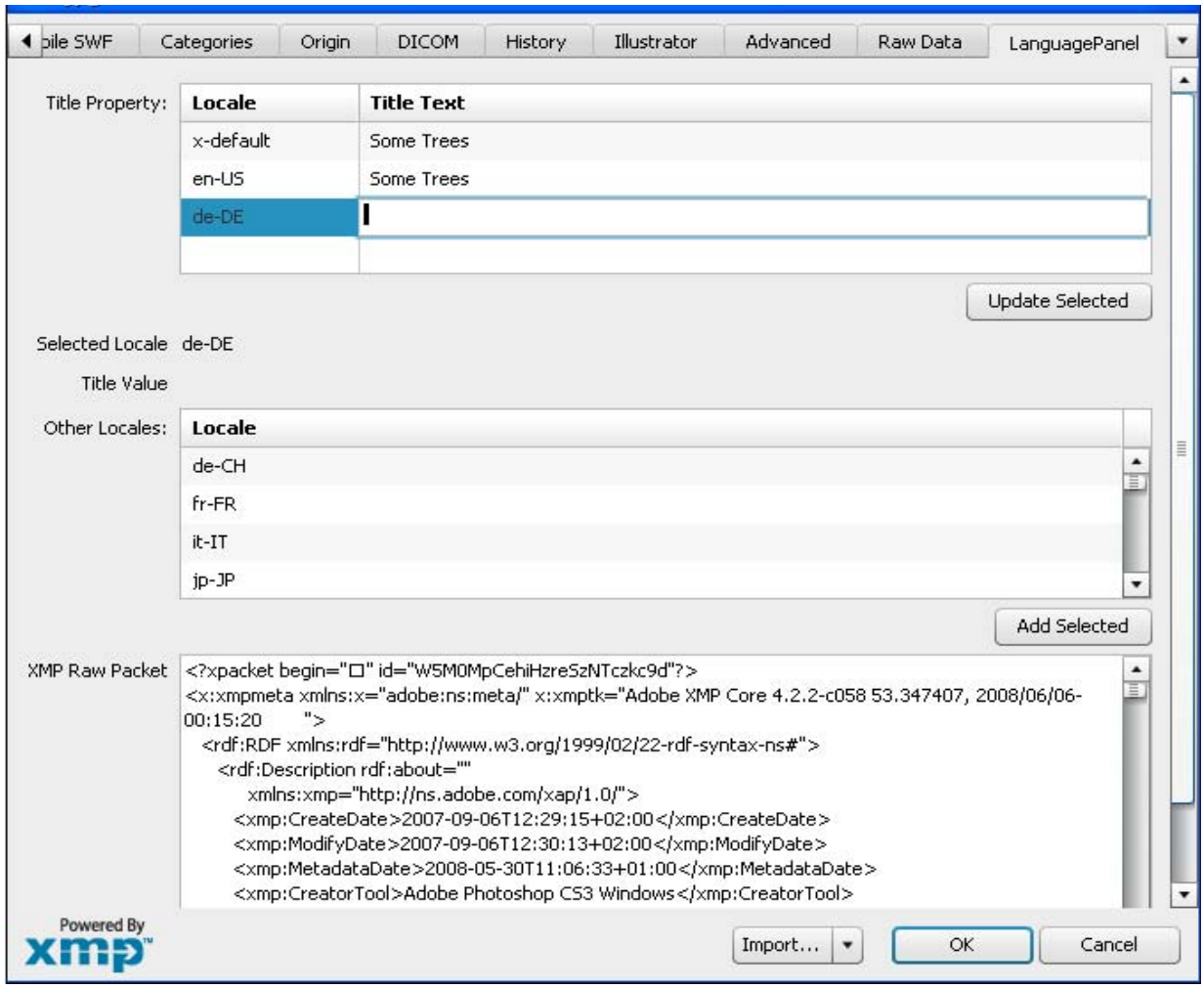
The sample also illustrates how to create a custom XMP metadata schema (specified in an XML file delivered along with your panel definition) and reference these fields in the custom panel. See "Showing custom XMP properties" on page 32.

The sample panel looks like this when selected in a File Info dialog in an Adobe CS5 application:



Language panel

This sample creates a panel for the File Info dialog that displays contents of the XMP Packet for the examined file, and provides two data grids that allow the user to add or modify localized title values for the `dc:title` property.



Setting values in the panel does not actually write changes to the XMP until you explicitly update the file. To update the XMP in the file, select the modified field in the upper grid and click "Update selected." The update operation can add locale entries for the `dc:title` property, or update existing locale entries.

- To read a locale entry, it uses code like this:

```
xmpAccess.getLocalizedText("dc:title", "it-IT");
```

- To update the value of an existing locale entry or add a new one, it uses code like this:

```
xmpAccess.setLocalizedText("dc:title","it-IT","New String", 0);
```

The operation also updates the displayed XMP Packet, so that you can see the changes that have been made to the title localization values.

Defining XMP handlers

You can define read and write event handlers to change or extend the default read/write behavior that is already built into the XMP components through the `xmpPath` attribute. This sample demonstrates the use of a non-default XMP event handler.

Each time you open a specific XMP panel in the File Info dialog (either by opening the dialog or by switching to a new tab), that panel fires an `xmpRead` event that tells each XMP component to retrieve its property data from the XMP Packet, before the panel is actually displayed. You can define an `xmpRead` handler, and pass it as the `xmpRead` parameter to the `XMPForm` tag. The function will be called each time the panel is displayed.

The LanguagePanel sample does this as follows:

```
<fi:XMPForm
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:fi="com.adobe.xmp.components.*" width="100%" height="100%"
    label="LanguagePanel" xmpRead="xmpReadHandler(event)" >
```

For this sample, the `xmpRead` handler displays the contents of the XMP Packet below the two grids, showing the initial state of the localization values for the `dc:title` property:

```
private function xmpReadHandler(event:XMPEvent): void
{
    var packet:String = event.xmpAccess.getXMPPacket();
    rawPacket.text = packet;
}
```

You could also provide an `xmpWrite` handler, and pass it as a parameter to the `xmpForm`. This handler would be called when the user switches the panel or closes the dialog; you could use it, for example, to do some processing of the entries submitted in the form before saving the new metadata values to the XMP Packet.

You can associate a read or write handler with a form, as shown here, in order to modify the behavior of all components in that form; you can also associate a handler with a single component, if you want to modify only that component. For example, if you have a component with an `xmpPath` attribute that defines a default read/write handler, and you add custom `xmpReadHandler` methods to both that component and to the containing form, the handlers are called in this order:

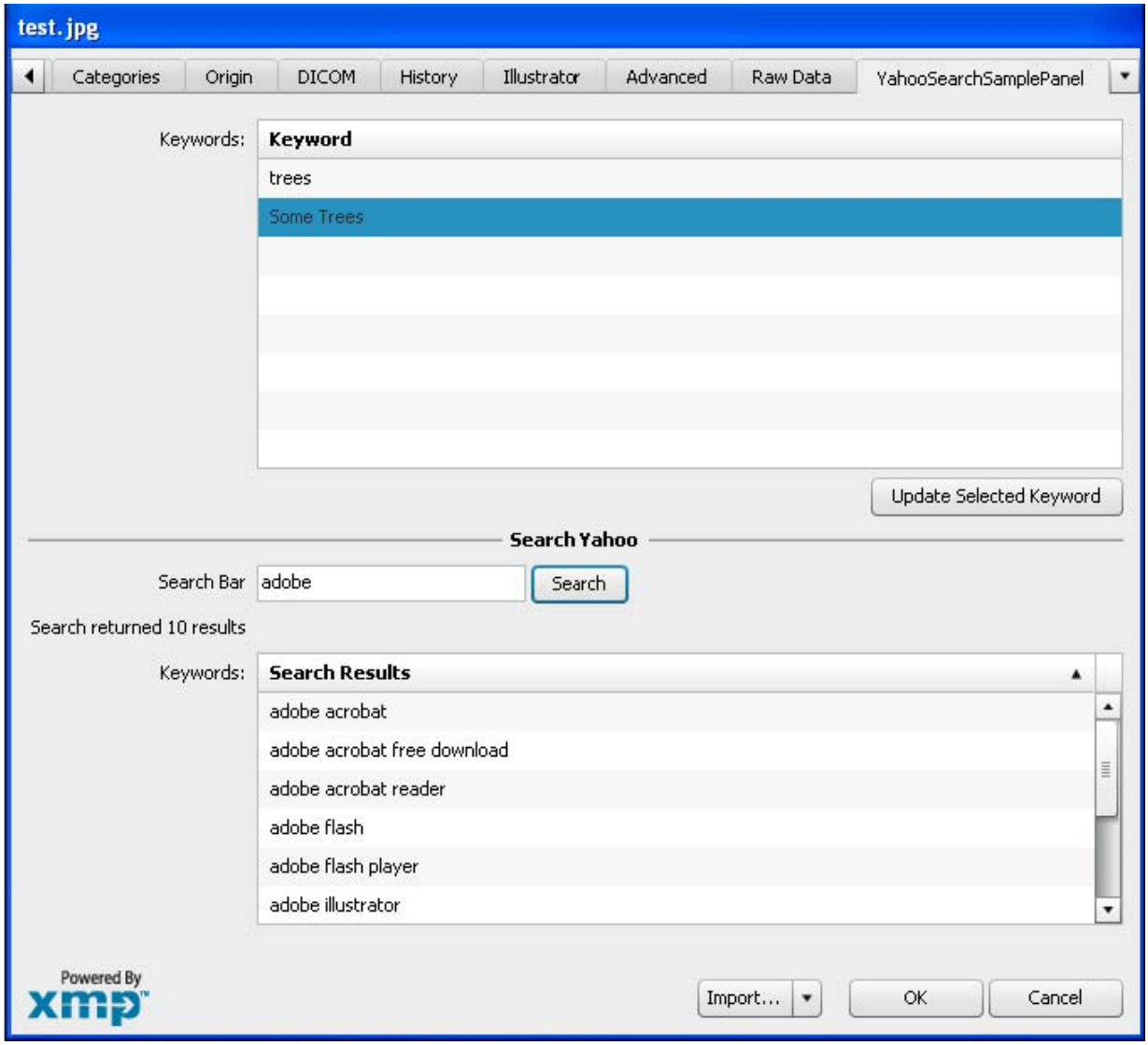
1. The `xmpReadHandler` for the Form.
2. The default read handler defined by the `xmpPath` attribute.
3. The `xmpReadHandler` for the component.

YahooSearchSample panel

This sample defines a panel that allows the user to search for a word or phrase using an external web service (Yahoo in this example). It has two sections:

- The lower section configures the search, calls the external service, and displays the search results. This search makes use of an external service; see "Creating an HTTP connection" on page 18.
- The upper section contains a component that is connected to the keyword metadata in the file (the "dc:subject" property). It shows the initial keyword values, and allows the user to enter new ones.

The user can select a search result from the lower panel, and drag it up into the Keywords list in the upper panel in order to associate it with the selected image as a metadata keyword.



To perform a search:

1. Enter a word or phrase in the Search Bar in the lower panel.
2. Click **Search**.
3. The Keywords list in this panel is populated with the results.

To add one of the returned search results as a new entry to the `keyword` field for the examined file:

1. Select the entry in the lower Keywords list.
2. Drag it into the upper Keywords list.
3. Click **Update Selected**. This updates the XMP metadata in the file.

Creating an HTTP connection

This sample demonstrates connecting to an external service to retrieve search results for the string entered by the user. To do this, it adds an `HTTPService`.

```
<mx:HTTPService id="connect"
    contentType="application/x-www-form-urlencoded"
    concurrency="last"
    method="GET"
    resultFormat="xml"
    />
```

For documentation of this component, see http://livedocs.adobe.com/flex/3/langref/mx/rpc/http/mxml/HTTPService.html.

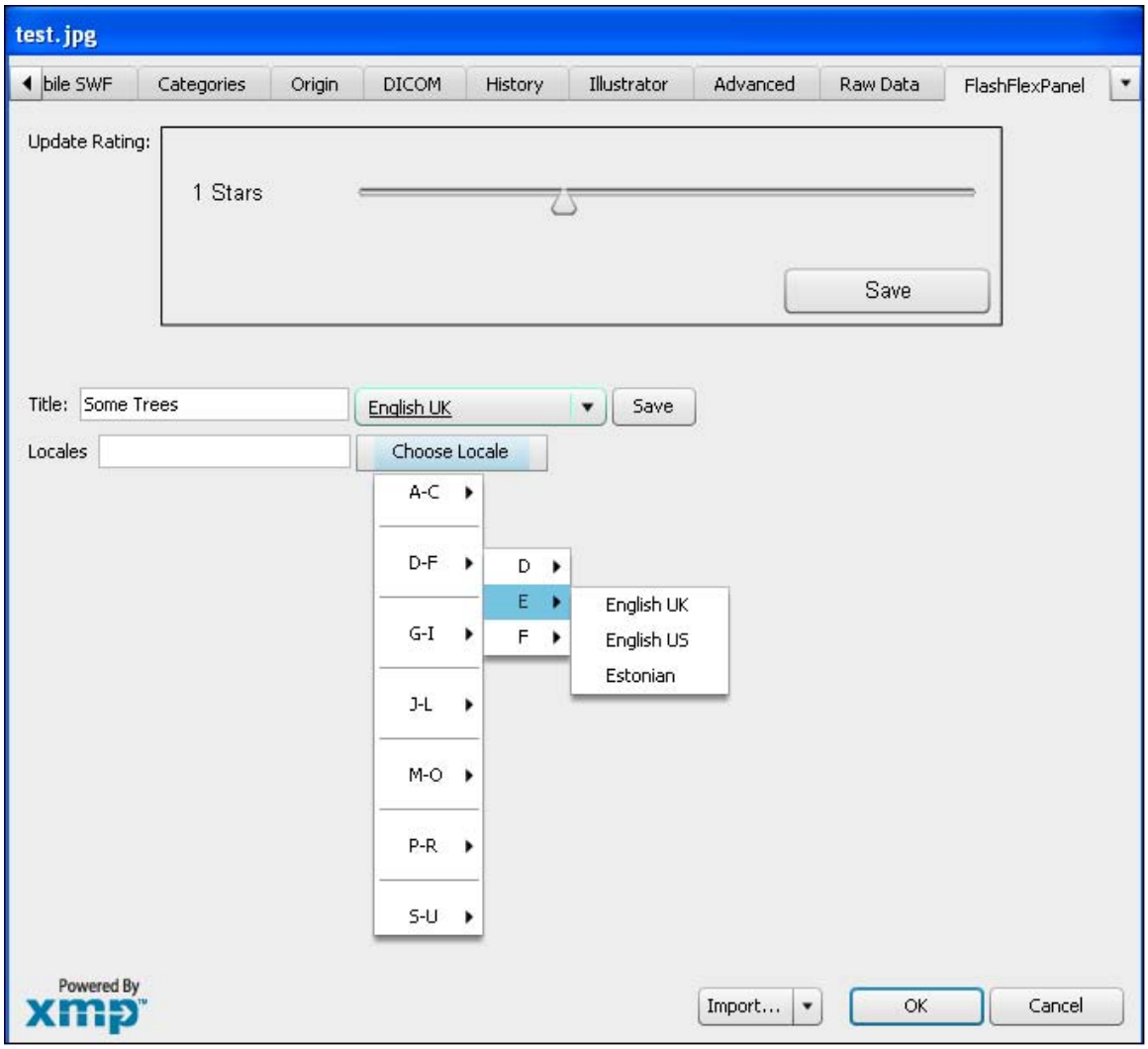
The sample creates a `textInput` form item with a button listener, which establishes a connection and retrieves the results from Yahoo:

```
private function searchYahoo(searchString: String): void
{
    connect.addEventListener("result", retrieveResults);
    connect.url = "http://ff.search.yahoo.com/gossip?output=xml&command=" +
        searchString;
    connect.send();
    searchArray.refresh();
}

private function retrieveResults(event:ResultEvent): void
{
    var xmlNode:XMLNode = XMLNode(event.result);
    if (xmlNode){
        searchStatus = "Search returned " + xmlNode.childNodes.length + " results";
        for (var i:int = 1; i < xmlNode.childNodes.length; i++){
            var newObj:Object = {value: xmlNode.childNodes[i].attributes.k};
            searchArray.addItem(newObj);
        }
    }
}
```

FlashFlex panel

If you have previously developed components or Flash content that you would like to integrate into a File Info panel, this sample shows how to go about it.

This sample demonstrates how to embed a Flash file in a panel definition, using a Flash `LocalConnection` object to communicate between Flex and Flash, and how to extend the XMP component set to create customized components.



The top of the panel contains a slider component with which to update an XMP rating value (rather than the `XMPRating` component used in the `BasicControlSample`), and a **Save** button defined by an embedded Flash (SWF) file. Update the rating value and click the button to send a message from the Flash file to the custom panel, so that the updated rating value is saved immediately into the XMP Packet (rather than waiting for the File Info dialog to be dismissed).

The form item in the panel definition (`FlashFlexPanel.mxml`) uses the `@Embed()` function with the `SWFLoader` component (part of the Flex library) to embed the Flash file that defines the button:

```
<mx:VBox>
    <fi:XMPFormItem label="Update Rating:" labelTooltip="Update Rating using the slider
           and click 'Save' button to write changes" >
        <mx:SWFLoader id="myLoader" source="@Embed(source='FlashFlex.swf')"
           creationComplete="initApp()" width="500" height="150" />
    </fi:XMPFormItem>
</mx:VBox>
```

The FLA file that implements this button, and the Flash (SWF) file that results from compiling it, are included among the source files for the sample. Once it is embedded, the subsidiary file becomes part of the panel's Flash file, and does not need to be included among the delivery files.

The lower part of the panel demonstrates a different way to implement the localized-title functionality shown in the `Language` sample, using customized components. The source files `CustomLocaleDrop` and `CustomMenuBar` define the two customized components.

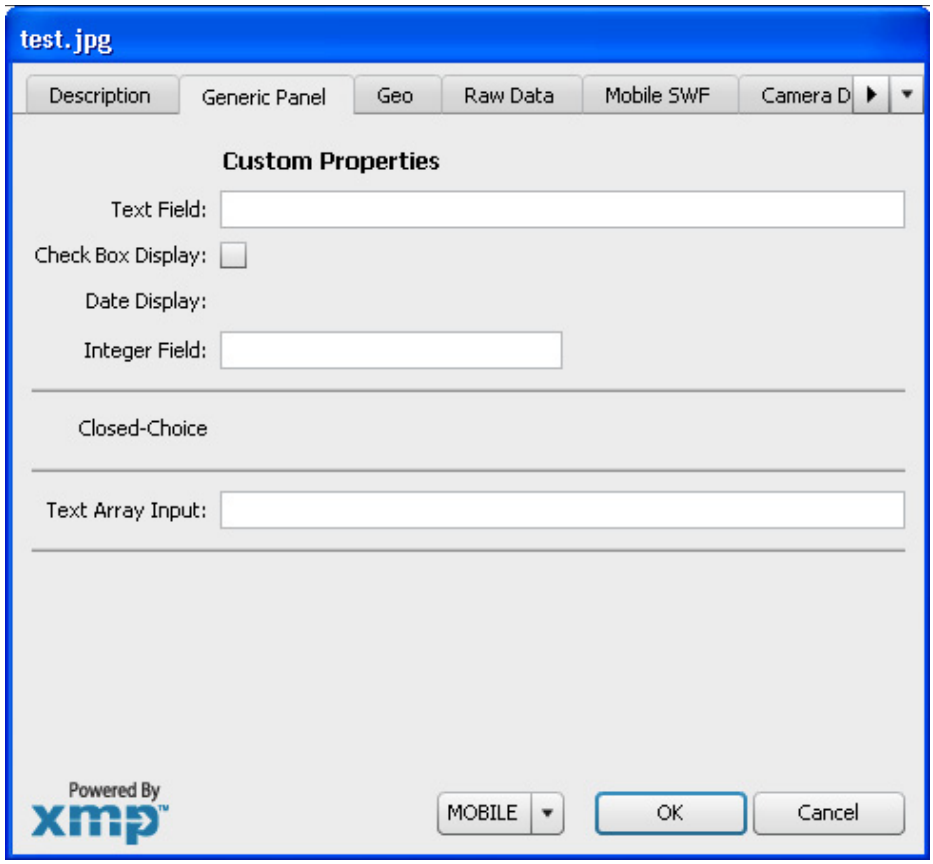
- `CustomLocaleDrop` shows a drop-down menu customized to contain entries for all Adobe-supported locales. It associates each locale name (the `label`, used as the display string) with the locale code (the `data`, used as the return value when an item is selected).
- `CustomMenuBar` shows an example of a nested drop-down menu, separating Adobe-supported locales alphabetically. The function `menuHandler` responds to a selection by returning the selected locale string to the calling object.

Generic panel

This panel reflects a custom XMP schema; the components are generated at run time from an XML file that specifies the properties and values. It is not a template, like the other samples, but a ready-made panel that you can use directly to provide access to your own XMP properties. This panel has limited customizability, but can get you started quickly creating simple custom panels without the need for a Flash Builder project.

You can easily customize the panel to show your own schema, which you provide in XML format. You then include the panel as it is, along with the customized support files, in the panel delivery location. For details of how to use and customize this panel, see Chapter 3, "The Generic Panel Tool."

The sample schema provided with the panel demonstrates properties of all types. When displayed with this sample `properties.xml` file, the panel looks like this:



2 Defining Custom File Info Panels

This chapter provides hands-on examples and advice for defining custom panels for the File Info dialog using Adobe Flash Builder, which is available as a standalone application or as an Eclipse plug-in.

- "Getting started" on page 21 describes how to set up the development environment and create a project.
- "Customizing a panel" on page 25 shows how to add components of various kinds to your custom panel, and associate them with metadata properties.
- "Showing custom XMP properties" on page 32 shows how to define your own XMP namespace and add its properties to your panel.
- "Defining panel preferences" on page 35 shows how to define your own persistent user preferences for a custom panel.

Getting started

This section describes how to set up the XMP Custom Panel project in which you build a custom panel. You can use Eclipse with the Flash Builder plug-in, or Adobe Flash Builder with the Apache Ant plug-in installed. See "The panel development environment" on page 10.

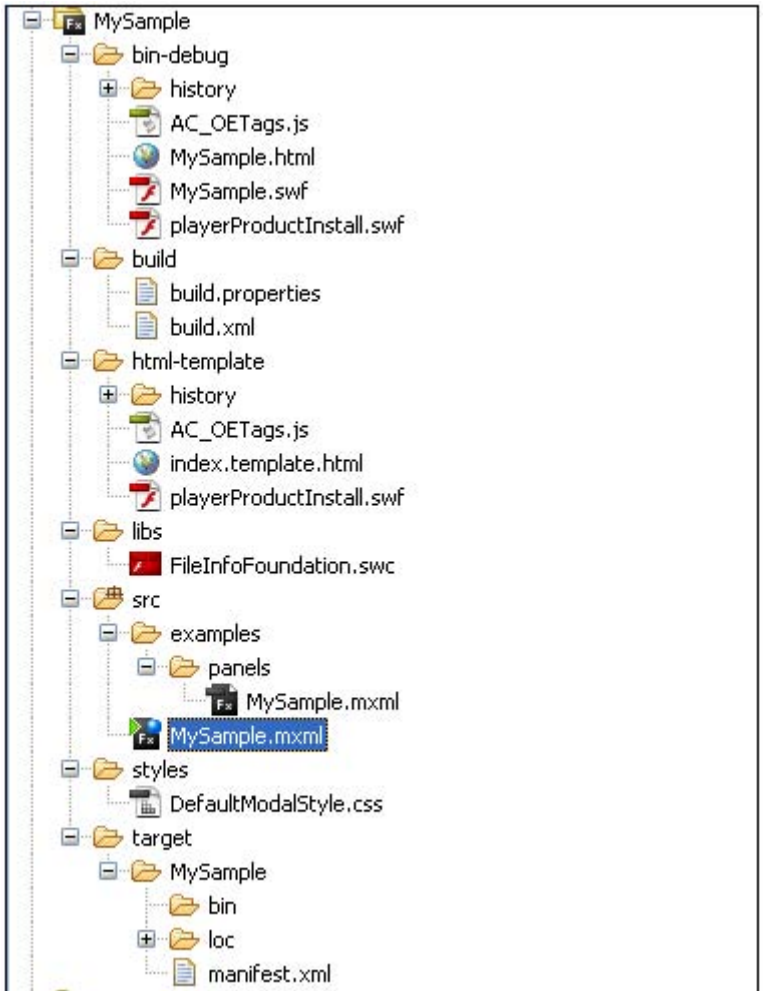
Set up the project

Use the Project Wizard to create a new project:

1. Start the Eclipse or Flash Builder development environment, and choose a suitable workspace.
2. Bring up the Project Wizard by right-clicking in the Package Explorer or Flex Navigator and choosing **New > Project**.
3. From the list of project types, choose **XMP > Custom Panel** and click **Next**.
4. Name the project `MySample` and click **Next**.
5. Accept the defaults in the remaining pages, and click **Finish**. This creates the basic file structure in your workspace directory, in the subfolder `MySample`.
6. Open the file `build/build.properties`. Check that the paths to the Flex SDK (`flex-sdk-dir`) and to the XMP application-data location (`build.target`) are correct for your platform and installation. For example, in Windows, the default locations are:

```
flex-sdk-dir      C:/Program Files/Adobe Flash Builder/sdks/3.3.0
publish.target    C:/Documents and Settings/<username>/Application
                  Data/Adobe/XMP/Custom File Info Panels/3.0/panels
panel-dir         MySample
panel-library     MySample
panel-classes     example.panels.MySample
projectName       MySample
```

This is what the folder structure looks like:



Examine initial panel components

The `src/examples/panels/` folder contains the working source file for the panel, `MySamplePanel.mxml`, which is already populated with the base `XMPForm` component, and a sample form item:

```
<?xml version="1.0" encoding="utf-8"?>
<fi:XMPForm
       xmlns:mx="http://www.adobe.com/2006/mxml"
       xmlns:fi="com.adobe.xmp.components.*" width="100%" height="100%"
       label="MySample" >
    <fi:XMPNamespaces>
       <fi:XMPNamespace prefix="dc" value="http://purl.org/dc/elements/1.1/"/>
    </fi:XMPNamespaces>
    <fi:XMPFormItem label="$$$/xmp/sdk/custompanels/MySample/Panel/Title=Document
Title:" labelTooltip="$$$/dc/title/TitleToolTip=The title of the document, or the name
given to the resource. Typically, it will be a name by which the resource is formally
known.">
       <fi:XMPTextInput xmpPath="dc:title" xmpType="Localized"/>
    </fi:XMPFormItem>
</fi:XMPForm>
```

Notice that:

- The form specifies a prefix for the Dublin Core XMP schema, using the `XMPNamespace` component.
- The sample form item contains a localized label, and a text input component connected to the Dublin Core `title` property.
- The localized label value references a translation dictionary that is already populated with an entry for this ZString, in the `target/MySample/loc` folder.

Run the project

You can run the project in the development environment for debugging, without publishing the Flash file.

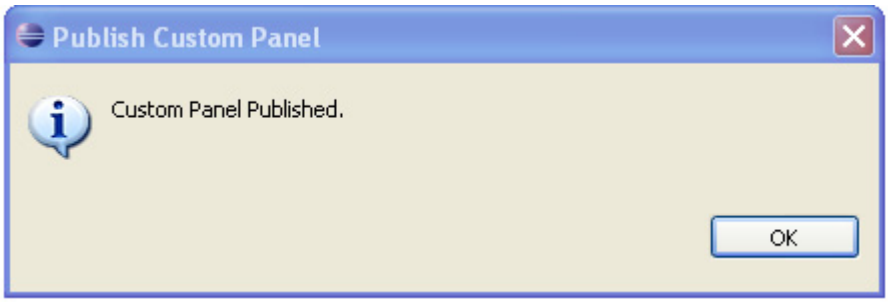
7. To run the project, right click and choose **Run As > Flex Application**. (You could also double-click the file `bin-debug/MySample.swf`.)

   A new browser window or the Flash Player should appear with the basic custom panel.

Publish the project

Now we need to publish the panel, so that it will appear in the File Info dialog of any CS5 application.

8. Select the project root, right click, and choose **XMP> Publish Custom Panel**. A dialog appears notifying you that the project has been published.

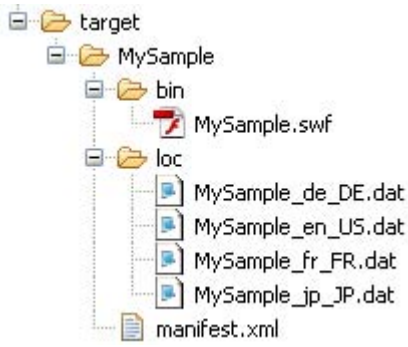


   The publication process has compiled the source MXML to create a Flash (SWF) file of the panel, and created all necessary support files in a delivery folder

9. Examine the `target/MySample...../` folder. This is the delivery folder for your panel.

   ▷ The `MySample/bin/` folder contains the file `MySample.swf`, which is the Flash file created from your source MXML.

   ▷ The publication process has also created support files needed for delivery, including the manifest that describes the panel to the File Info dialog, and localization dictionaries.



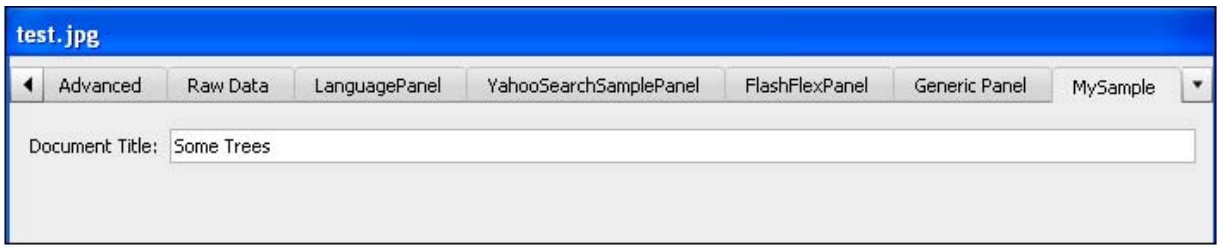
The publication operation has also written the delivery files to the application-data location for XMP on your computer:

IN WINDOWS XP: `C:\Document and Settings\<username>\Application Data\Adobe\XMP\Custom File Info Panels\3.0\panels`

IN WINDOWS 7/VISTA: `C:\Users\<username>\AppData\Roaming\Adobe\XMP\Custom File Info Panels\30\panels`

IN MAC OS: `/Users/<username>/Library/Application Support/Adobe/XMP/Custom File Info Panels/3.0/panels`

If you open any CS5 application and choose **File Info**, you should see your new panel in the dialog.



To make your panel available to the File Info dialog for your end users for any Adobe CS5 application, you must install the delivery folder containing it to the `panels` folder in the XMP application-data location for that user and platform, and also create a Flash trust file; see "Installing custom panels" on page 52. The SDK does not support creation of a custom panel installer.

DEBUGGING TIPS: If you have any trouble with the dialog (such as a custom panel not appearing), try deleting the preferences file (`FileInfoPrefs.xml`) and reinvoking the dialog; see "Panel preferences" on page 49.

You can also enable debugging messages by adding the attribute `logShowPanel=true` to the top-level `xfi:fileinfo` tag of the panel manifest; see "Panel manifests" on page 47.

Customizing a panel

We will now create a more complex panel that demonstrates some of the controls and features that you can use.

Examine the initial form item

Each form item is associated with one metadata property. It displays the current value of the property, and can be used to modify the value. An item's `xmpPath` attribute identifies the XMP property that the item reflects.

An XMP Custom Panel project that you create with the Project Wizard is automatically supplied with a form item, which you can use or modify. This `XMPFormItem` contains one text-input control, an `XMPTextInput` component. The control reflects a property in the Dublin Core (`dc`) namespace.

```
<?xml version="1.0" encoding="utf-8"?>
<fi:XMPForm
       xmlns:mx="http://www.adobe.com/2006/mxml"
       xmlns:fi="com.adobe.xmp.components.*" width="100%" height="100%"
       label="MySample" >
    <fi:XMPNamespaces>
       <fi:XMPNamespace prefix="dc" value="http://purl.org/dc/elements/1.1/"/>
    </fi:XMPNamespaces>
    <fi:XMPFormItem label="$$$/xmp/sdk/custompanels/MySample/Panel/Title=Document
Title:" labelTooltip="$$$/dc/title/TitleToolTip=The title of the document, or the name
given to the resource. Typically, it will be a name by which the resource is formally
known.">
       <fi:XMPTextInput xmpPath="dc:title" xmpType="Localized"/>
    </fi:XMPFormItem>
</fi:XMPForm>
```

The form item looks like this:

Add a rating component

We will add a rating value to show a different kind of panel component, and demonstrate how to show a metadata property in another namespace. To do this:

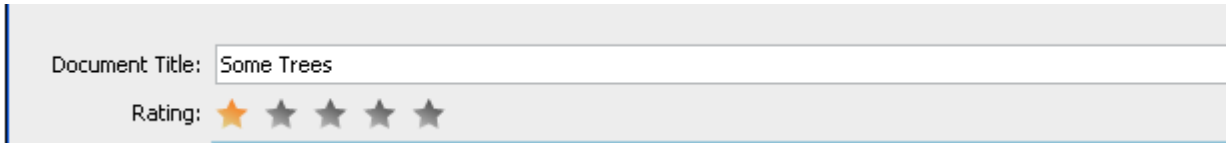
10. Add the XMP namespace for the property you want to reflect in this component.

```
<fi:XMPNamespaces>
    <fi:XMPNamespace prefix="dc" value="http://purl.org/dc/elements/1.1/"/>
    <fi:XMPNamespace prefix="xmp" value="http://ns.adobe.com/xap/1.0/"/>
</fi:XMPNamespaces>
```

11. Add a new form item to the form, containing an `XMPRating` component. Set the component's `xmpPath` attribute to the name of property that this item reflects (`rating`) and the namespace (`xmp:`) in which it is defined.

```
<fi:XMPFormItem label="$$$/xmp/sdk/custompanels/MySample/Panel/Rating=Rating:">
    <fi:XMPRating xmpPath="xmp:Rating"/>
</fi:XMPFormItem>
```

The two form items now look like this:



Localize text with ZStrings

All labels and tooltips for the XMP components support localization. Any label-text property can contain a *ZString key* (a string beginning with "$$$/*keyname*", together with a default value, such as "$$$/locKey=default value". The panel looks up the named in any provided localization files. If there is no localization file for the current locale, or if the key is not found in the file, the default value is displayed.

If you want to localize strings in your program code, use the class `ZStringManager`:

12. Import the `ZStringManager` class into the custom panel.

    After the closing tag of the first `XMPFormItem` component, add this code:

```
<mx:Script>
    <![CDATA[
        import com.adobe.xmp.utils.ZStringManager;

        private function localize(locKey: String): String
        {
            return ZStringManager.getString(locKey);
        }
    ]]>
</mx:Script>
```

    Within this `mx:Script` element you can add ActionScript code for additional functionality and handling.

The `ZStringManager` object supports the conversion of ZStrings to ActionScript properties. Notice that the code has already started to use ZStrings for localization of the display text in the custom panel. The File Info dialog uses a generic naming mechanism for localization files, which must be stored in a folder called `loc/` in the panel's root folder.

Each string-dictionary must be named using the naming convention:

```
baseName_localeCode.dat
```

The manifest declares the base name of the string dictionaries for localization:

```
localizationFile = "MySample"
```

Using this base, we might provide these string-dictionary files:

```
MySample_en_US.dat
MySample_jp_JP.dat
MySample_it_IT.dat
MySample_fr_FR.dat
```

For a complete list of locales supported by Adobe Creative Suite 5, you can refer to the built-in panels.

The content of the English localization file, `MySample_en_US.dat`, provides the English text for the three ZString used in the components at this point, and looks like this:

```
"$$$/xmp/dc/title_tooltip=The title of the document, or the name given to the resource.
Typically, it will be a name by which the resource is formally known."
"$$$/xmp/sdk/custompanels/MySample/Panel/Title=Document Title"
"$$$/xmp/sdk/custompanels/MySample/Panel/Rating=Rating:"
```

The `ZstringManager` returns a string for the current locale. The default locale is `en_US`. Use the method `ZstringManager.setLocale("localeCode")` to change the locale if required. By default, the locale is set by the host application.

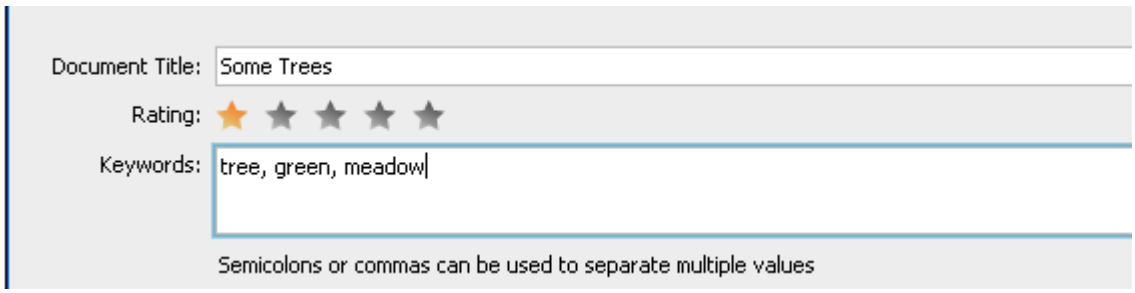
IMPORTANT: Each ZString key (that is, the portion before the "=") must be unique, so that strings are unambiguously identified.

Add a list of values

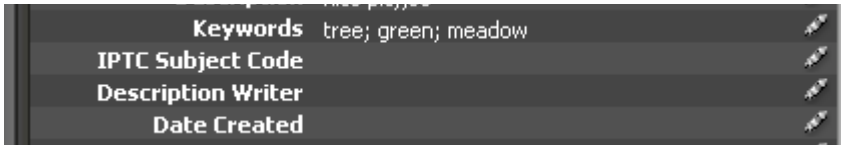
13. To demonstrate the use of the `XMPTextArea` component, use it to add a list of keywords, separated by comma or semicolon characters:

```
<fi:XMPFormItem label="$$$/xmp/sdk/custompanels/MySample/Panel/Keywords=Keywords:"

    labelTooltip="$$$/xmp/dc/subject_tooltip=An unordered array of
                  descriptive phrases or keywords that specify the topic
                  of the content of the resource.">

    <mx:VBox width="100%">
        <fi:XMPTextArea xmpPath="dc:subject" xmpArray="bag"/>
        <mx:HBox>
            <mx:Label  text="{ZStringManager.getString('$$$/xmp/sdk/custompanels/
MySample/Panel/
                  keywordHints=Semicolons or commas can be used to separate
                  multiple values')}"/>
        </mx:HBox>
    </mx:VBox>
</fi:XMPFormItem>
```

The form items now look like this:



If you invoke this panel in Adobe Bridge, for example, and save the file information, Adobe Bridge recognizes and displays these keywords:



Add behavior with event handlers

Our next form item opens a browser and navigates to the URL given in the property `url` in the `stJob` namespace. The behavior is supplied by a `click` event handler.

14. Add the `Job#` namespace:

```
<fi:XMPNamespaces>
 <fi:XMPNamespace prefix="dc" value="http://purl.org/dc/elements/1.1/"/>
 <fi:XMPNamespace prefix="xmp" value="http://ns.adobe.com/xap/1.0/"/>
 <fi:XMPNamespace prefix="stJob" value="http://ns.adobe.com/xap/1.0/sType/Job#" />
</fi:XMPNamespaces>
```

15. Add a second form to the divided box:

```
...
</fi:XMPForm>

<fi:XMPForm id="_formMid" height="33%">
    ADD NEW ITEMS HERE
</fi:XMPForm>
```

This second form within the divided box creates a new section. You will see an indicator of the divider above them; this allows a user to change the proportions of the divisions. Later, we will add an event handler to respond to that action.

16. In the new form, add this form item to define a text-input component with specialized behavior:

```
    <fi:XMPFormItem label="$$$/xmp/sdk/custompanels/MySample/Panel/URL=URL:">
        <fi:XMPTextInput xmpPath="stJob:url" id="url" width="80%"
            xmpRead="xmpReadHandler(event)"
            change="changeHandler(event)"/>
            <mx:Button id="_urlButton"
                label="{ZStringManager.getString('$$$/xmp/sdk/custompanels/MySample/
Panel/GoToURL=Go To URL...')}"
                click="goToURL(url.text)"
                enabled="false" visible="false"/>
    </fi:XMPFormItem>
```

17. Add the definition of the button's click-handler function in the `mx:Script` element:

```
private function goToURL(url: String): void
{
    if (url != null  &&  url.length > 0) {
        if (url.indexOf("://") < 0) {
            url = "http://" + url;
        }
        navigateToURL(new URLRequest(url), 'GoToURL');
    }
}
```

Notice that the `XMPTextInput` specifies two event-handler functions.

- The `xmpRead` handler, `xmpReadHandler(event)`, is called when the panel is loaded, and detects whether there is a valid value in the associated property.
- The `change` handler, `changeHandler(event)`, is called when the user updates this field. If there is no text left in the field, it disables and hides the button.

18. In the `mx:Script` element, add this code to import the supporting event functions:

```
import com.adobe.xmp.events.XMPEvent;
```

19. Add the function definition for the read handler in the `mx:Script` element:

```
private function xmpReadHandler(event:XMPEvent): void
{
    var prop:String =
        event.xmpAccess.getProperty(url.xmpPath).value;
    if (prop == null)
    {
        _urlButton.enabled = false;
        _urlButton.visible = false;
    }
    else if (prop != null)
    {
        _urlButton.enabled = url.text != null  && url.text.length > 0;
        _urlButton.visible = url.text != null  && url.text.length > 0;
    }
}
```

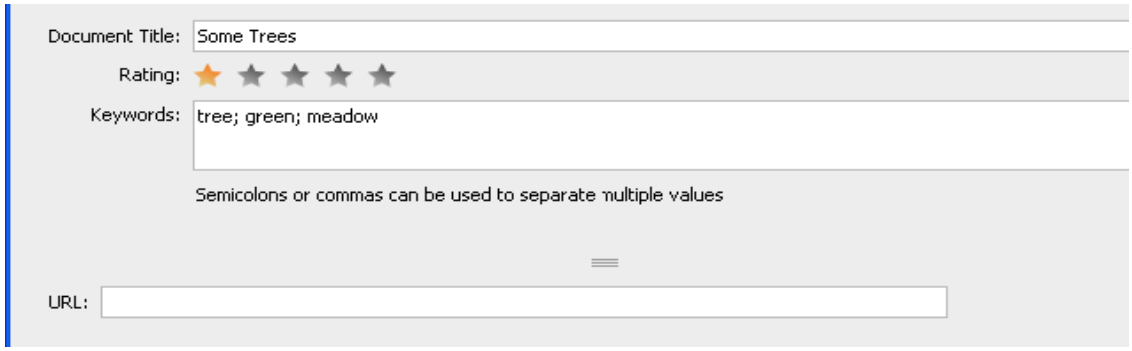
The `xmpReadHandler` function accesses the component using the assigned ID, `url`. It uses the `xmpPath` method of the component to get access to the associated metadata property. The `value` method accesses the string value contained in the property. If there is a valid entry, the button is enabled and made visible.

20. Add the function definition for the change handler in the `mx:Script` element:

```
private function changeHandler(event:Event): void
{
    _urlButton.enabled = url.text != null  && url.text.length > 0;
    _urlButton.visible = url.text != null  && url.text.length > 0;
}
```

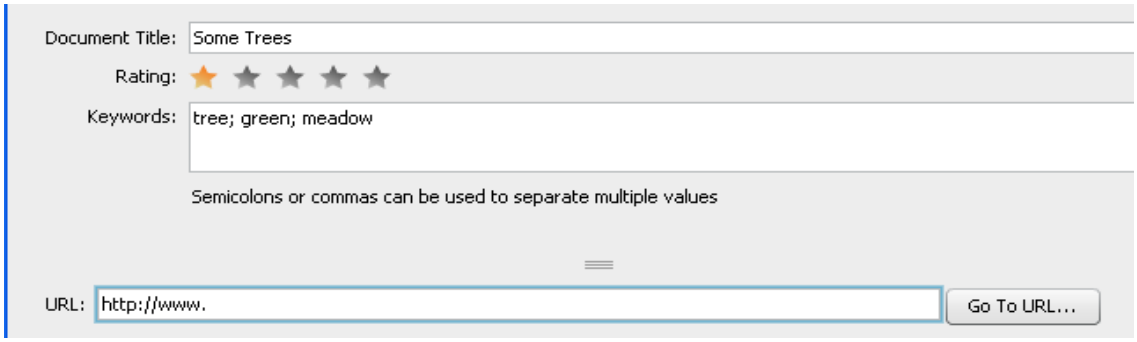
The `changeHandler` function works in the same way but it checks the panel's text field to determine if there is a valid string.

When the handler has disabled the button because there is no valid text in the text field, the result looks like this:



Notice the divider indication above the new form item. More on this later.

When valid text is entered, the handler enables and shows the button:



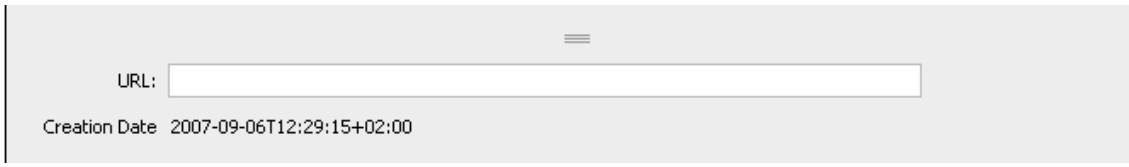
Add non-editable values

21. If you want to display a metadata value in the panel, but not allow the user to modify it, use the `XMPText` component.

    Add this form item to display the creation date in a non-editable text field:

```
<fi:XMPFormItem label="$$$/xmp/sdk/custompanels/MySample/Panel/
                      creationDate_text=Creation Date">
    <fi:XMPText xmpPath="xmp:CreateDate" id="xmp_CreateDate" />
</fi:XMPFormItem>
```

Here is the result:



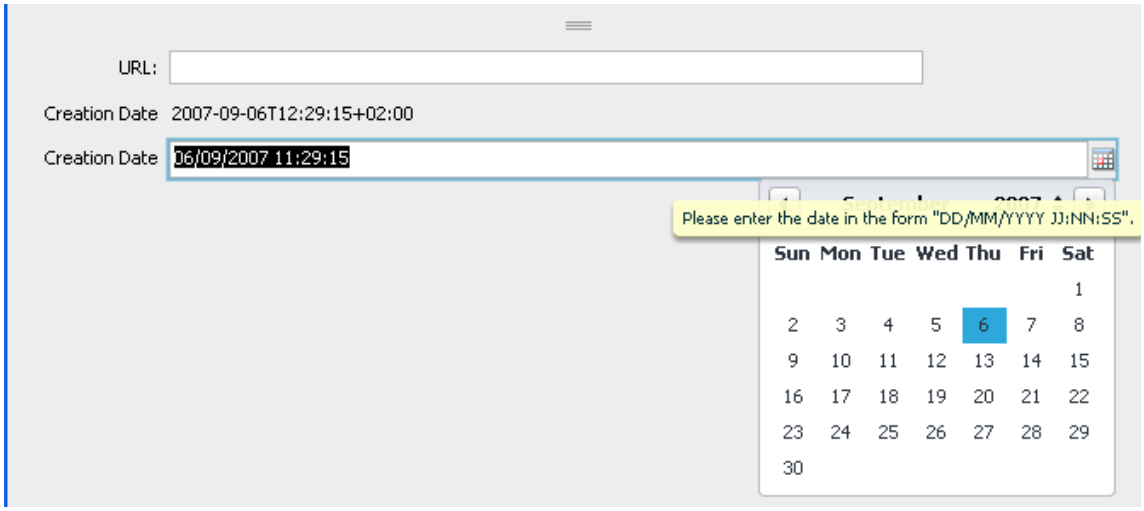
Add modifiable date values

22. A special component, `XMPDateField`, provides a pop-up calendar to make updating date fields easier.

    Add this form item to allow entry of a new creation date:

```
<fi:XMPFormItem label="$$$/xmp/sdk/custompanels/MySample/Panel/
                      creationDate_edit=Creation Date">

    <fi:XMPDateField xmpPath="xmp:CreateDate" id="xmp_create_date"
                     writeEmptyValues="false"/>
</fi:XMPFormItem>
```

Here is what it looks like:

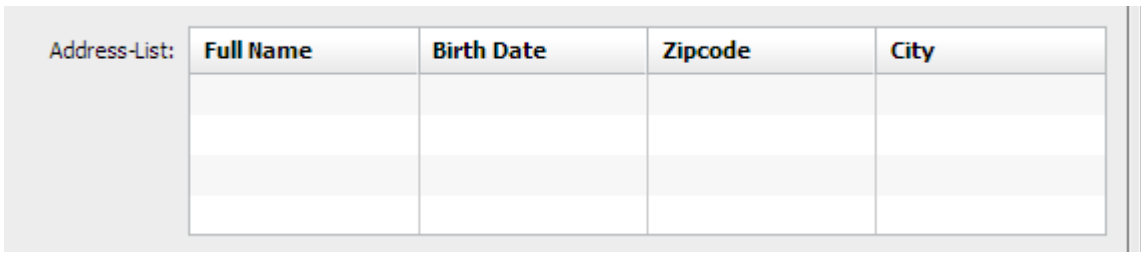


Add a table component

23. The `XMPDataGrid` component can provide a visual representation of an XMP array that contains several fields in each array item. In this example, a simple address list is embedded in the panel:

```
<!-- Use the table control for an array of structs, in this example an address list -->
<fi:XMPFormItem id="_myTable" label="Address-List:" visible="false"
       includeInLayout="false">
    <fi:XMPDataGrid xmpPath="my:AddressList" xmpArray="bag">
       <fi:columns>
          <fi:XMPDataGridColumn xmpPath="my:FullName" headerText="Full Name"
             headerTip="Please enter the full name."/>
          <fi:XMPDataGridColumn xmpPath="my:Birthdate" xmpType="Date"
                headerText="Birth Date" headerTip="Please enter the date of birth."/>
          <fi:XMPDataGridColumn xmpPath="my:ZipCode" xmpType="Integer"
                headerText="Zipcode" headerTip="Please enter the zipcode."/>
          <fi:XMPDataGridColumn xmpPath="my:City" headerText="City"
                headerTip="Please enter the city."/>
       </fi:columns>
    </fi:XMPDataGrid>
</fi:XMPFormItem>
```

Here is what is looks like:



Showing custom XMP properties

XMP is, by definition, extensible; you can create custom properties in addition to those that are predefined in public schemas. When you create a custom panel for the File Info dialog, you can simply associate components with your custom properties, and their values are displayed in exactly the same way as predefined properties.

This example assumes that you have defined the following properties in a namespace "MyNamespace", with the prefix "my", each of which contains values of a representative type:

```
my:mytext
my:mydate
my:myint
my:mybool
my:mybag
```

For the complete schema definition in XML format, see "Creating an XML schema file to support non-FileInfo applications" on page 34.

1. Edit the MXML file for your panel to add a namespace tag for the new schema:

```
<fi:XMPNamespaces>
    <fi:XMPNamespace prefix="dc" value="http://purl.org/dc/elements/1.1/"/>
    <fi:XMPNamespace prefix="my" value="http://ns.adobe.com/MyNamespace/"/>
</fi:XMPNamespaces>
```

2. In the form, add the simple `XMPSeparator` component to create a horizontal separator in the panel:

```
<fi:XMPSeparator/>
```

3. You can display these fields in the panel using the appropriate component types, as already demonstrated.

   ▷ Add `XMPTextInput` components for the text and integer values

   ▷ Add an `XMPDateField` for the date value.

4. For the multi-valued `mybag` field, add this text-input component:

```
<fi:XMPFormItem label="$$$/xmp/my/mybag=My Bag" >
    <fi:XMPTextArea xmpPath="my:mybag" xmpArray="bag"/>
</fi:XMPFormItem>
```

5. For the Boolean value, we will add a new type of component, a check box.

   ▷ In the `mx:Script` element, after the imports, add this declaration:

```
[Bindable]private var mybool:Boolean;
```

   ▷ Add this form item:

```
<fi:XMPFormItem label="$$$/xmp/sdk/custompanels/MySample/Panel/
                       Schema/mybool=My Boolean Property">
    <fi:XMPCheckBox xmpPath="my:mybool" selected="false" />
</fi:XMPFormItem>
```

6. The next form item we will add is a combo box component, whose choices map to the multi-valued `mychoice` field. Create the new form item:

```
<fi:XMPFormItem label="$$$/xmp/my/myChoice=My Choice:">
    <fi:XMPComboBox dataProvider="{myChoiceItems}" xmpPath="my:mychoice" />
</fi:XMPFormItem>
```

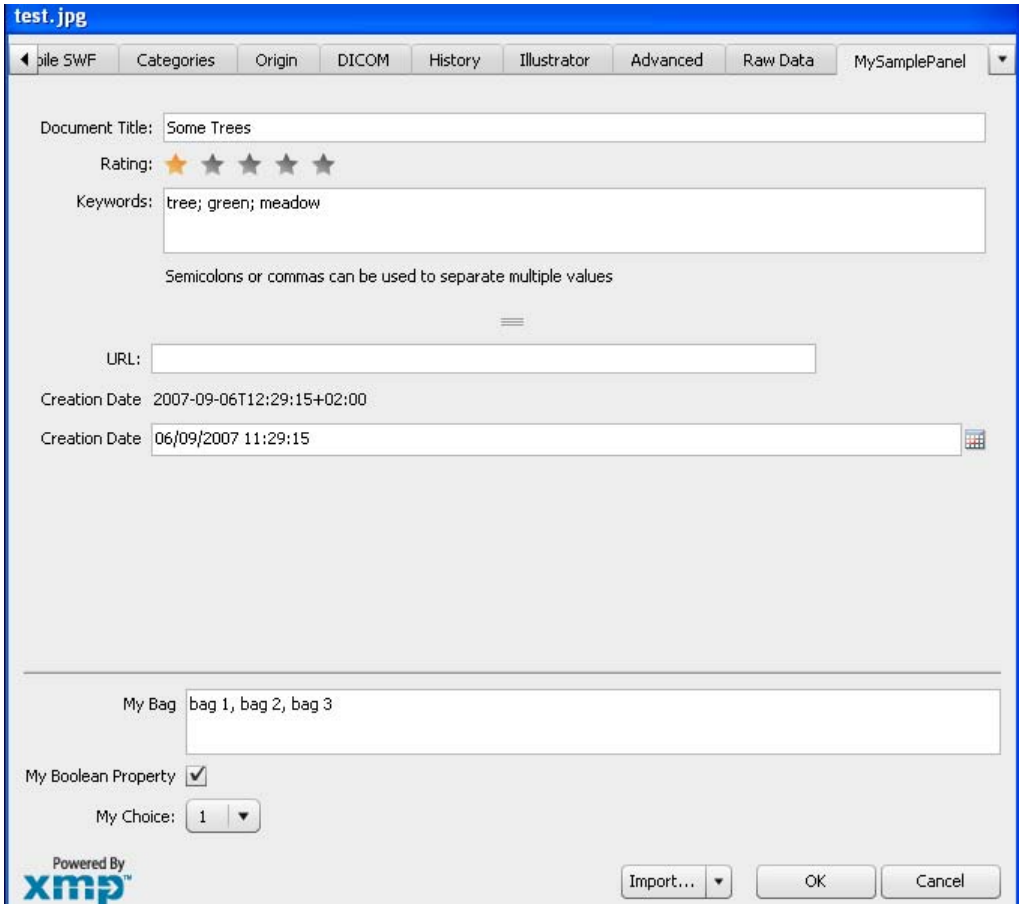
7. Notice the `dataProvider` property in the `XMPComboBox`. This expects an Array or ArrayCollection that contains key/value pair objects, each containing the properties `label` (the displayed value) and `data` (the value written to the XMP). The labels are translated to obtain the drop-down list entries. The data value is the value saved to the metadata field when a list entry is chosen.

   For this example, we will declare the `dataProvider` collection within the `mx:Script` element, and make it bindable so that the notation `{myChoiceItems}` is valid. We will use the utility function `ZstringManager.getDictionaryFrom()` to translate each of the display strings. This function generates a `dataProvider` from a single ZString entry, in the form of a semicolon-separated list, where each combobox item contains the display label and the corresponding XMP value in curly braces.

   After the import statement for the `ZStringManager` in the `mx:Script` element, add the following:

```
import mx.collections.ArrayCollection;
[Bindable]
private var myChoiceItems: ArrayCollection = ZStringManager.getDictionaryFrom(
"$$$/xmp/sdk/custompanels/MySample/Panel/myChoiceItems=1{1};2{2};3{3}");
```

The finished panel looks like this:



Creating an XML schema file to support non-FileInfo applications

You can use an XML representation to define your own XMP properties, and include it in your project, so that the values that are edited in your File Info panel are also made available to applications that do not use the File Info dialog.

When you create a custom schema in an XML file, you must deliver that file along with your custom panel definition in the delivery location for your platform, `.../Custom File Info Panels/3.0/custom/`. The applications that do not use the File Info dialog, such as Adobe Bridge and Premiere Pro, look in this location for the XML file, while ignoring the custom panel definitions.

You could also use the Generic Panel Tool with this schema file; the Generic Panel also looks in this location for a schema file, and automatically generates a panel from the definitions it finds here. See Chapter 3, "The Generic Panel Tool."

This sample file, `properties.xml`, creates the schema used in the previous example, with the name `MyNamespace`, and the prefix `"my"`. It defines the five properties we used to demonstrate some of the different data types.

1. Create a file named `properties.xml` in the `target/MySample` folder.

2. Add this code to define the new namespace and its properties:

```
<?xml version='1.0' encoding='UTF-8'?>
<xmp_definitions>
    <xmp_schema prefix="my" namespace="http://ns.adobe.com/MyNamespace/"
           description="The My Schema" label="$$$/xmp/my/mylabel=My" >

        <xmp_property name="mytext" category="external"
               label="$$$/xmp/my/mytext=My Text" type="text" />

        <xmp_property name="mydate" category="internal"
               label="$$$/xmp/my/mydate=My Date" type="date" />

        <xmp_property name="myint" category="external"
               label="$$$/xmp/my/myint=My Integer" type="integer" max="500"
               min="-500" />

        <xmp_property name="mybool" category="external"
               label="$$$/xmp/my/mybool=My Boolean" type="boolean" />

        <xmp_property name="mybag" category="external" type="bag"
               label="$$$/xmp/my/mybag=My Bag" element_type="text" />

        <xmp_field name='mychoice' type='closedchoice'
               label='$$$/xmp/my/mychoice=My Choice' element_type="text">
           <xmp_choice raw_value='1' label='$$$/xmp/my/mychoice/1=1'/>
           <xmp_choice raw_value='2' label='$$$/xmp/my/mychoice/2=2'/>
           <xmp_choice raw_value='3' label='$$$/xmp/my/mychoice/3=3'/>
        </xmp_field>

    </xmp_schema>
</xmp_definitions>
```

Defining panel preferences

You can define user preferences for your own panel, and persist them in XML format as part of your panel description. Your own code must collect the preference values, interpret them, and write them back.

This example adds two preferences to the My Sample panel that we have built, controlling the placement of the divider line, and the background color of the middle part of the panel. It also adds a color-picker component that allows the user to set the color.

1. First, we will add the color-picker component. In the MXML file for your panel, add this form item to the second form within the divided box:

```
<fi:XMPFormItem label="midColor">
    <mx:ColorPicker id="_colorPickerMid" change="colorChanged(event)"/>
</fi:XMPFormItem>
```

2. This definition includes an event-handling function that listens for changes in the control. In the `mx:Script` element, add this definition at the top level:

```
private function colorChanged(event:ColorPickerEvent):void{
    _formMid.setStyle("backgroundColor", event.color);
    this.xmpModification = true;
}
```

   Setting the modification flag to true causes the form's `xmpWrite` handler to be called when you close the panel—we'll define the write handler in a moment.

3. Enclose the form in a divided box, to create a section within the panel:

```
<mx:DividedBox id="_divider" width="100%" height="100%">
    <fi:XMPForm id="_formTop" height="33%">
        ...FORM ITEMS AND COMPONENTS...
    </fi:XMPForm>
</mx:DividedBox>
```

4. Within the divided box, add this event handler to listen for changes:

```
<mx:DividedBox id="_divider" width="100%" height="100%"
       dividerRelease="dividerHandler(event)">
    ...
</mx:DividedBox>
```

5. In the `mx:Script` element, add the event-handler function definition:

```
private function dividerHandler(event:DividerEvent):void{
    this.xmpModification = true;
}
```

6. In the `mx:Script` element, add this code to import the events in which we are interested:

```
import mx.events.ColorPickerEvent;
import mx.events.DividerEvent;
```

7. Add the code that checks for the existence of preferences, and uses the values to change the appearance of the panel. Because this has to happen whenever the user opens the panel, it goes into the handler for the `xmpRead` event, defined in the `mx:Script` element.

   In the function `xmpReadHandler()`, add this code:

```
    // Check the panel description to see if preferences have been defined
```

```
    if(panelDescription.prefs != null){
        var prefs:XMLList = panelDescription.prefs; // retrieve preferences
        if(prefs){
            // adjust divider using preferred height
            _formMid.height = parseInt(prefs[0].text());

            // set background color of middle section
            var bgColor:uint;
            var hexColor:String = prefs[1].text();
            // check for a valid entry
            if(hexColor.length > 0){
                // transform the stored string to a hex value
                bgColor = parseInt(hexColor.substr(1), 16);
                // set preferred color as background of middle section
                _formMid.setStyle("backgroundColor", bgColor);
                // set preferred color as current selection of the color picker
                _colorPickerMid.selectedColor = bgColor;
            }
        }
    }
```

8. Now we will add the code that saves the preference settings, in XML format. This has to happen whenever the user closes the custom panel, so we will create a handler for the `xmpWrite` event.

   In the `mx:Script` element, add this code:

```
/** Store preferences from form */
private function xmpWriteHandler(event:XMPEvent):void{
    // Store the preferences in an XML list object
    var xml_list:XMLList = new XMLList();

    // Get the current divider height
    var dividerPosition:int = _formMid.height;
    // Create an XML element and add it to the list
    xml_list += <divider>{dividerPosition}</divider>;

    // Get the current selected color
    var hexColor:String;
    // Transform the hex value to a string
    hexColor = "#" + _formMid.getStyle("backgroundColor").toString(16);
    // Create an XML element and add it to the list
    xml_list += <bgColor>{hexColor}</bgColor>;

    // Store the preferences in the panel description
    panelDescription.prefs = xml_list
}
```

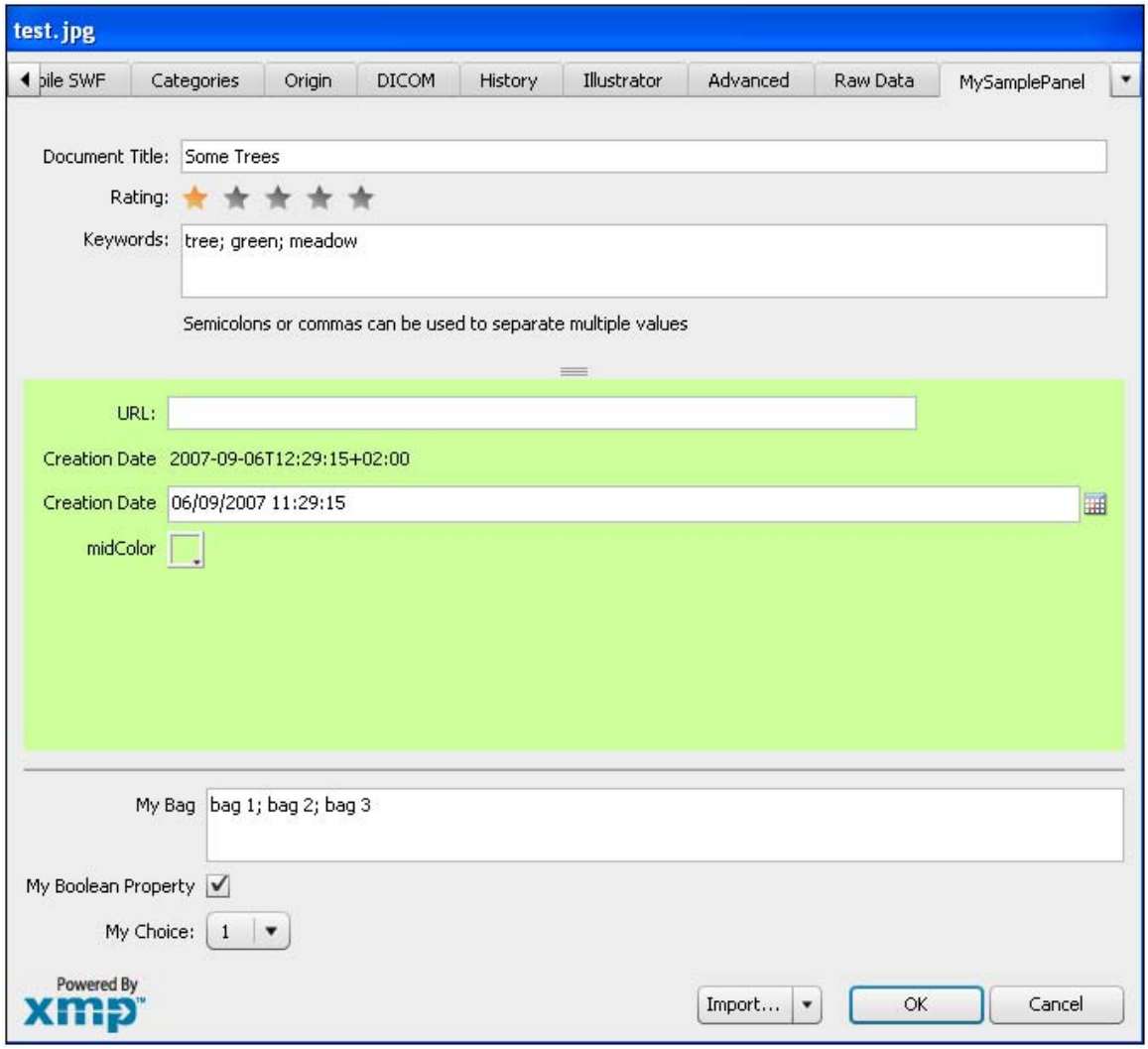
   The new preferences we have defined here, `<divider>` and `<bgColor>`, are automatically saved with other dialog preferences when you close the File Info dialog, and restored for the next session. See "Panel preferences" on page 49.

9. Add this new write-event handler to the form. At the top of the main `XMPForm` tag, add this line:

```
xmpWrite="xmpWriteHandler(event)"
```

We have now added a color-picker control, and event handlers that respond to picking a color and to moving the divider. The color handler changes the appearance of the panel immediately, but both handlers mark the panel as modified. This allows the write-handler to store the new color and divider position in persistent preferences, in order to save the values when you close the panel. The read-handler restores the values when you re-open the panel.

This is what the panel looks like when you have picked a background color for the middle section of the divided box:



This appearance is saved and restored when you close and re-open the panel.

3 The Generic Panel Tool

The Generic Panel, a tool provided with the XMP FileInfo SDK, generates a panel automatically from an XML file that defines custom XMP properties. You can use this tool to quickly and easily provide access to your own custom XMP schema. Install this ready-made panel as it is, and simply provide your own schema in XML format.

The XML file that supplies the XMP schema information for the Generic Panel is referenced from the panel manifest. A sample XML file is provided to show the syntax and structure; you can modify it to describe your own XMP schema. This same XML file that you use to generate a panel for the File Info dialog can also be read and displayed by the built-in metadata palettes used in Adobe Bridge, Adobe Premiere, and other Adobe audio-video applications.

The Generic Panel can display only simple XMP properties and comma-separated array lists, with limited layout capabilities. For more complex workflows, it is recommended that you create a custom panel using Flex components.

Using the Generic Panel

To view the generated Generic Panel, copy the ready-made panel files from the SDK `samples/panels/Generic` folder into the user-panels delivery folder. See "Installing custom panels" on page 52.

The files include the following:

- The compiled Flash file that implements the panel itself, `bin/generic.swf`.
- The sample `properties.xml` file that contains the XMP schema to be displayed. To customize the panel, edit this file to contain your custom properties, in XML format. See "XML schema file format" on page 42.
- The panel description file, `manifest.xml`. To customize the panel, you will edit this file. See "Customizing the manifest" on page 39.
- Sample translation dictionaries for English and German in the `/loc` subfolder. When you customize the panel, you can edit these to translate the strings used in your own schema, and add any other languages you need; see "Customizing the localization files" on page 39.

If you plan to use more than one instance of the Generic Panel, you must modify the `manifest.xml` file to customize it for each specific instance. You must at least change the panel name; see "Customizing the manifest" on page 39. You must also ensure that any ZStrings you use in each panel have unique keys.

If you make any changes to an instance of the Generic Panel, you must update the modification date of the `manifest.xml` file, so that the File Info dialog will recognize the changes when it opens. Alternatively, you could delete the preferences file (although user preferences would be lost); see "Panel preferences" on page 49.

Customizing the manifest

To customize your panel, remove the read-only attribute of the `manifest.xml` file, and edit the properties highlighted in this example:

```
<xfi:fileinfo xmlns:xfi="http://ns.adobe.com/xmp/fileinfo/">
    <xfi:panels>
        <xfi:panel
            name = "MyPanel"
            label = "My Panel"
            description = "This is my custom panel."
            type = "custom"
            version = "1.0"
            panelLibrary = "generic"
            panelClass = "examples.panels.generic.GenericPanel"
            propertyDescriptionFile = "properties.xml"
            localizationFile = "MyPanel"
            modifyDate = "2008-12-13Z"
            visible="true">
        </xfi:panel>
    </xfi:panels>
</xfi:fileinfo>
```

- The `label` value appears in the tab for this panel, and the `description` is the tool tip. Both strings can be specified as ZStrings for localization.
- The `name` value is a unique identifying name for the panel. It must not contain spaces.
- You can use more than one copy of the Generic Panel; you must give each one a different name. Do NOT change the name of the SWF file, or the values of `panelLibary` and `panelClass`.
- The `propertyDescriptionFile` value is the name of the XML file that defines your custom properties. This one references the provided sample file.
- The `localizationFile` value is the base file name for translation dictionaries that you supply. See "Customizing the localization files" on page 39.
- The `modifyDate` value is the current date in ISO 8601 format.

Customizing the localization files

The localization dictionaries must be located in the `loc/` subfolder. The tool comes with sample files, which you must replace with your own translations for the ZStrings used in your panel. If you don't use ZStrings, you can delete the folder.

The naming convention for dictionary files appends a locale code to the base name specified by the manifest's `localizationFile` value. In the sample manifest, this is "Generic", and the sample dictionary files (English and German) are named `Generic_en_US.dat` and `Generic_de_DE.dat`.

You can add dictionary files for other languages as needed. To see which languages are supported by the Creative Suite applications, you can look at the dictionaries provided for the File Info dialog itself, in your platform's application data location, `.../Custom File Info Panels/Panels/3.0/bin/loc`.

If you customize the manifest as shown in the example, providing the base name "MyPanel", the panel looks for dictionary files in the `loc/` subfolder named `MyPanel_en_US.dat` and so on. You must supply these files for all languages that you support. If no appropriate dictionary is found for a locale, the panel is displayed in English. For details, see "Localize text with ZStrings" on page 26.

You can rename and edit the sample dictionaries in any text editor. You must include entries for all ZString keys used in your panel.

IMPORTANT: You must save the dictionary files with UTF-16LE (little-endian) encoding.

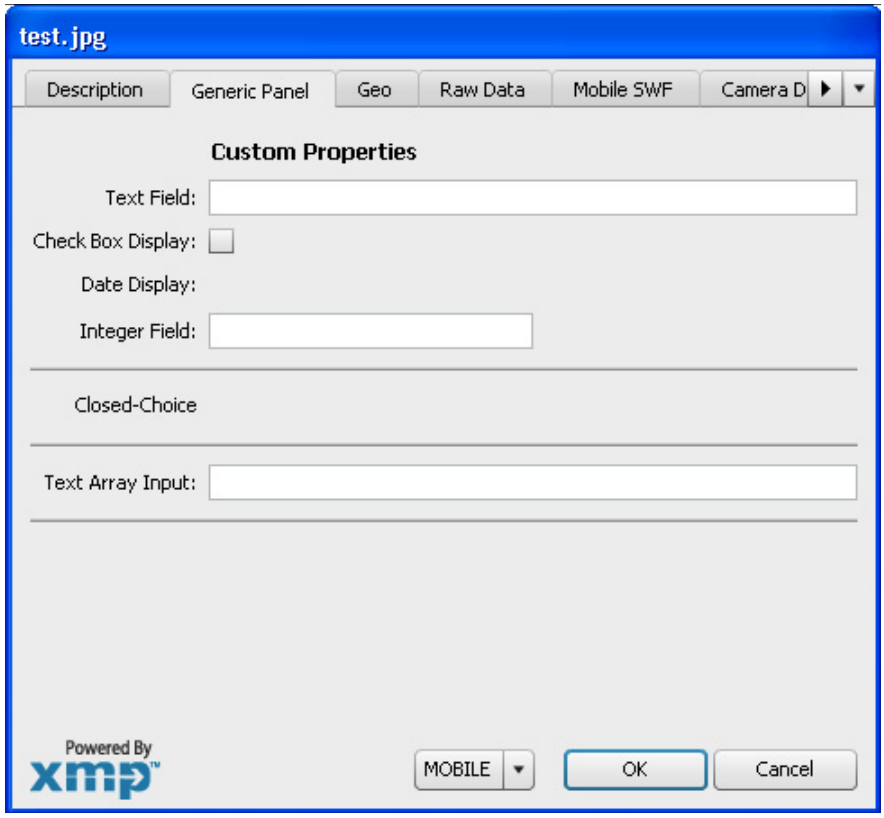
Customizing the property description

Here is an example of a `properties.xml` file that demonstrates the different types of properties that can be shown by the Generic Panel, and the panel that results from interpreting this file. For complete details of the XML syntax, see "XML schema file format" on page 42.

Example properties.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<xmp_definitions xmlns:ui="http://ns.adobe.com/xmp/fileinfo/ui/">

    <xmp_schema prefix="my" namespace="http://ns.adobe.com/MyNamespace/"
        label="$$$/Custom/Schema/Label=Custom Properties"
        description="$$$/Custom/Schema/Description=This is my example panel.">
        <!-- simple properties -->
        <xmp_property name="mytext" category="external"
            label="$$$/Custom/Property/TextInputLabel=Text Field:" type="text" />
        <xmp_property name="mybool" category="external"
            label="$$$/Custom/Property/BooleanDisplay_Label=Check Box Display:"
            type="boolean" />
        <xmp_property name="mydate" category="internal"
            label="$$$/Custom/Property/DateDisplay_Label=Date Display:" type="date" />
        <xmp_property name="myint" category="external"
            label="$$$/Custom/Property/IntegerInputLabel=Integer Field:" type="integer"
            ui:format="{0} cm" ui:width="50%" />
        <ui:separator/>
        <!--simple properties (choices) -->
        <xmp_property name='mychoice' category='internal' type='closedchoice'
            label='$$$$/Custom/Property/ClosedChoiceDisplay_Label=Closed-Choice
                Display:'>
            <xmp_choice raw_value='1' label='$$$/xmp/my/mychoice/1=1'/>
            <xmp_choice raw_value='2' label='$$$/xmp/my/mychoice/2=2'/>
            <xmp_choice raw_value='3' label='$$$/xmp/my/mychoice/3=3'/>
        </xmp_property>
        <ui:separator/>
        <!-- arrays -->
        <xmp_property name="mybag" category="external" type="bag"
            label="$$$/Custom/Property/TextArrayInput_Label=Text Array Input:"
            element_type="text" />
        <ui:separator/>
    </xmp_schema>
</xmp_definitions>
```

Resulting panel



XML schema file format

The schema file format that the Generic Panel uses is generally the same as that used for Premiere and other DVA-application metadata panels, but is somewhat more restrictive. It supports top-level properties and arrays, but not complex XMP types.

The outer tag for the `properties.xml` file is `<xmp_definitions>`. It contains one or more `<xmp_schema>` tags. Generally there is one schema tag for each XMP namespace, but a file can contain more than one schema tag for the same namespace. Each schema tag and its contained properties are rendered into the Generic Panel as UI controls, in order of appearance.

```
<?xml version="1.0" encoding="UTF-8"?>
<xmp_definitions xmlns:ui="http://ns.adobe.com/xmp/fileinfo/ui/">

    <xmp_schema prefix="my" namespace="http://ns.adobe.com/MyNamespace/"
        label="$$$/Custom/Schema/Label=Custom Properties"
        description="$$$/Custom/Schema/Description=This is my example panel.">
         <!-- PROPERTIES HERE -->
    </xmp_schema>

</xmp_definitions>
```

Within the schema tag, a list of `<xmp_property>` tags defines the actual properties to be displayed in the panel. Each property is rendered as a UI control within an `XMPFormItem` in the form for its parent schema, containing XMP components appropriate to the property type.

<xmp_definitions>

The top-level tag of an XMP schema definition. Contains

Attributes

`xmlns:ui`	The namespace for the optional UI attributes that you can specify for properties; see "<xmp_property>" on page 43. Value is `"http://ns.adobe.com/xmp/fileinfo/ui/"`. Note that the properties in this namespace are for the exclusive use of the Generic Panel, and do not belong to the custom schema format.

Subtags

`<xmp_schema>`	Specifies one custom namespace. Any number of schemas are allowed.

<xmp_schema>

Specifies a namespace to be rendered by a form in the Generic Panel.

Attributes

`label`	The localizable display name for this schema, used in the heading for the associated form.
`description`	An optional, localizable description, used as a tooltip for the associated form.
`prefix`	The namespace prefix used within this tag.
`namespace`	The namespace URI.

Subtags

<xmp_property>	Specifies a property in the namespace. Any number of properties are allowed.
<ui:separator>	Draws a line across the panel.

<xmp_property>

Specifies an XMP property to be rendered by a form item in the form for a namespace.

Attributes

label	The localizable display name for this schema, used in the label for the associated form item.
description	An optional, localizable description, use as a tooltip for the associated form item.
name	The name of the XMP property, unique within the namespace.
category	The read-write status of the property. One of external for editable, or internal for read-only.
type	The property type, which determines the types of components used in the associated form item. See "Mapping property types to components" on page 45 for details. ▶ For a simple property, one of: text integer real boolean date closedchoice openchoice ▶ For a language alternates array: langalt ▶ For any other array with simple-type elements, one of: bag: An unordered array. seq: An ordered array. alt: An array containing alternative values. For these types, you must also specify the element_type.
element_type	For any of the array types, the array element type. One of the simple data types: text integer real boolean date

UI attributes

The namespace `xmlns:ui="http://ns.adobe.com/xmp/fileinfo/ui/"` defines these attributes that can be used to customize the layout of the form item for a specified property in the Generic Panel. You must register this namespace in the <xmp_definitions> tag.

`ui:multiLine`	Boolean. Whether to render a text field as single-line (false, the default) or multi-line. Can be used for the types `text`, `langalt`, and all array types.
`ui:width`	The width of components, as a percentage of the panel width. Default is 100.
`ui:height`	The height in pixels for multi-line fields. Default is `NaN` (not a number).
`ui:mru`	Boolean. Whether to enable the "Most Recently Used" functionality for text input fields. Default is true.
`ui:format`	A format string that specifies a prefix or postfix for the property value. The value replaces `{0}` in this string; for example, `"{0} inch"`. Default is null, no additional formatting is done.

Subtags

<xmp_choice>	Choices for the property types `openchoice` and `closedchoice`.

Examples

A text property:

```
<xmp_property name="mytext" category="external"
   label="$$$/Custom/Property/TextInputLabel=Text Field:" type="text" />
```

A language-alternates property:

```
<xmp_property name="LocalizedText" category="external"
   label="$$$/Custom/Property/LocalizedTextInput_Label=Localized Text Input:"
   type="langalt"/>
```

An array property with unordered text elements:

```
<xmp_property name="mybag" category="external" type="bag"
   label="$$$/Custom/Property/TextArrayInput_Label=Text Array Input:"
   element_type="text" />
```

A numeric property with additional formatting attributes:

```
<xmp_property name="myint" category="external"
   label="$$$/Custom/Property/IntegerInputLabel=Integer Field:" type="integer"
   ui:format="{0} cm" ui:width="50%" />
```

Mapping property types to components

The type of component used to display a property depends on the property type. For many types, it also depends on whether the property is editable (that is, `category="external"`) and whether is it multi-line (that is, has the attribute `ui:multiLine="true"`).

`text`
`integer`
`real`
	For numeric and string values, the component type depends on whether the property is editable and multi- or single-line. For the types `text`, `integer`, and `real`, the following components are used:
	▶ Editable, single-line: `XMPTextInput`
	▶ Editable, multi-line: `XMPTextArea`
	▶ Read-only, single-line: `XMPLabel`
	▶ Read-only, multi-line: `XMPText`

`boolean`	Rendered as an `XMPCheckBox`. The value can be True or False.
`date`	Rendered as an `XMPDateField`. The value is a date in ISO8601 format, which can include time and optional time zone information; for example, `2008-07-17T18:53Z`.
`closedchoice`	Rendered as a non-editable `XMPComboBox`, with a list of options specified by <xmp_choice> tags.
`openchoice`	Rendered as an editable `XMPComboBox`, with a list of options specified by <xmp_choice> tags.
`langalt`	The `x-default` value is the one mapped to the component. The type of component is chosen using the editable and multi-line status, as for `text`.

`bag`
`seq`
`alt`
	Array-type properties are represented by a text-type component, with individual element values rendered as a comma- or semicolon-separated list. The type of component is chosen using the editable and multi-line status, as for `text`.
	▶ Editable, single-line: `XMPTextInput`
	▶ Editable, multi-line: `XMPTextArea`
	▶ Read-only, single-line: `XMPLabel`
	▶ Read-only, multi-line: `XMPText`
	For example, an editable, single-line array of Boolean values would be shown in an `XMPTextInput` component, using a string such as "True; True; False".
	It is recommended that array values not be made editable, except for simple text values such as keywords.

<xmp_choice>

A property of type `openchoice` or `closedchoice` can contain any number of choice tags. Each tag specifies one choice, which appears in the menu of the combo box associated with the property.

Attributes

`label`	The localizable display name for this choice, used in the menu for the associated combo box.
`raw_value`	The XMP value placed in the component when the user chooses the associated item.

Example

```
<xmp_property name="Color" label="Color:" type="openchoice">
    <xmp_choice raw_value="red" label="Red"/>
    <xmp_choice raw_value="green" label="Green"/>
    <xmp_choice raw_value="blue" label="Blue"/>
</xmp_property>
```

4 File Info Panel Concepts

This chapter provides a conceptual overview of the XMP FileInfo SDK.

- "Contents of a custom panel description" on page 47 describes the files that make up your custom panel description.
- "XMP Flex components" on page 50 describes the special-purpose Flex components that the SDK provides for building the panel contents.
- "Trust files for custom panels" on page 53 describes how to tell the Flash security system that your panel's Flash files are trusted.

Contents of a custom panel description

To define a custom File Info panel for an Adobe CS5 application, your plug-in must supply these files:

- An MXML source file containing the panel definition, using `XMPForm` and other Flex components supplied by the XMP FileInfo SDK. See "XMP Flex components" on page 50. (This source file does not need to be included in the delivery folder; it is compiled into a SWF file.)
- An XML panel manifest, named `manifest.xml`. See "Panel manifests" on page 47.
- A Flash (SWF) file created by compiling the MXML source file.
- Translation dictionaries for any localized strings (ZStrings) used in the panel. The use of ZStrings is optional, but if they are used, you must include the dictionaries in a subfolder named `loc`, and the files must be named using a base name specified in the manifest and a locale code, in this format:

  basename_localeCode.dat

  For example:

  panelRoot/loc/myPanel_en_US.dat
  panelRoot/loc/myPanel_de_DE.dat
  panelRoot/loc/myPanel_ja_JP.dat

All of the delivery files must be included in a single folder, whose name is specified in the manifest. (The name of the panel itself, which must be unique among panels, does not need to be the same as the folder name.) The folder must be installed in the Adobe XMP application-data location for the user, as defined for the platform. See "Installing custom panels" on page 52.

Panel manifests

For each custom panel plug-in, a file named `manifest.xml` provides the host application with a description of the File Info panel supplied by the plug-in. When the plug-in is loaded, the information is stored as a `PanelDescription` object for each panel, accessible programmatically through the `XMPForm.panelDescription` property. See the API documentation for details.

This XML file contains an `<xfi:panel>` tag for each panel being defined, in the following format:

```
<xfi:fileinfo xmlns:xfi="http://ns.adobe.com/xmp/fileinfo/">
```

```
    <xfi:panels>

        <xfi:panel ...>
        </xfi:panel>

    </xfi:panels>
</xfi:fileinfo>
```

You can enable debugging messages by adding the attribute `logShowPanel=true` to the top-level `xfi:fileinfo` tag of the panel manifest.

Each panel tag contains these attributes:

`name`	A unique identifying string for the panel. Must not contain spaces.
`label`	The display title for the panel, localizable with use of ZString notation and translation dictionaries.
`description`	A descriptive help string for the panel, which appears as a tooltip for when the cursor hovers over the tab. Localizable with use of ZString notation and translation dictionaries.
`type`	The panel type, "custom", which distinguishes a custom panel from Adobe "built-in" panels.
`panelLibrary`	The name of the compiled Flash file for the panel (without the `.swf` extension). For example, `myPanel`.
`panelClass`	The name of the MXML/ActionScript class containing the source panel definition. For example, `my.namespace.MyPanel`
`localizationFile`	The base name of translation dictionaries for Zstrings used in the panel. Must not contain spaces or special characters. All dictionary files must be in a `loc` subfolder. See "Contents of a custom panel description" on page 47.
`visible`	When true, the panel is shown in the File Info dialog, when false it is hidden.
`version`	A version number for this panel. This allows the File Info dialog to update cached information if you provide a new version of the panel.
`modifyDate`	The date when this panel definition was last updated, for use with version control. In ISO 8601 format; for example, 2010-04-23Z.

For example:

```
<xfi:fileinfo xmlns:xfi="http://ns.adobe.com/xmp/fileinfo/">
    <xfi:panels>
        <xfi:panel
            name = "CustomPanelSample"
            label = "Custom Panel Sample"
            description = "This panel contains the Custom Panel Sample."
            type = "custom"
            panelLibrary = "CustomPanelSample"
            panelClass = "examples.panels.CustomPanelSample"
            localizationFile = "CustomPanelSample"
            visible="true"
            version = "1.0"
            modifyDate = "2008-04-15Z">
        </xfi:panel>
```

```
    </xfi:panels>
</xfi:fileinfo>
```

Panel preferences

You can define arbitrary user preferences for your panel. Your components can access those preferences and use them in any way you wish. To do this, create an XML tag with the preference key and value, and store it in the `PanelDescription.prefs` property in an `XMLList`. For example, these are the preferences created in the example ("Defining panel preferences" on page 35) to set the appearance of the panel:

```
/** Store preferences from form */
private function xmpWriteHandler(event:XMPEvent):void{
    // Store the preferences in an XML list object
    var xml_list:XMLList = new XMLList();
    // Get the current divider height
    var dividerPosition:int = _formMid.height;
    // Create an XML element and add it to the list
    xml_list += <divider>{dividerPosition}</divider>;
    // Get the current selected color
    var hexColor:String;
    // Transform the hex value to a string
    hexColor = "#" + _formMid.getStyle("backgroundColor").toString(16);
    // Create an XML element and add it to the list
    xml_list += <bgColor>{hexColor}</bgColor>;
    // Store the preferences in the panel description
    panelDescription.prefs = xml_list;
}
```

Preferences that you define for your panel are automatically persisted in an XML file when the File Info dialog is closed. When the dialog is opened, it retrieves the preferences and stores them as XML in the `PanelDescription.prefs` property. See the API documentation for details of the `PanelDescription` object.

Preference file

All user preferences for the File Info dialog are automatically saved whenever the dialog is closed, to a location in the user's application-data folder for the dialog:

IN WINDOWS XP: `C:\Documents and Settings\<username>\Local Settings\Application Data\Adobe\XMP\File Info\3.0\work\FileInfoPrefs.xml`

IN WINDOWS 7/VISTA: `C:\Users\<username>\AppData\Local\Adobe\XMP\File Info\3.0\work\FileInfoPrefs.xml`

IN MAC OS: `/Users/<username>/Library/Preferences/Adobe/XMP/File Info/3.0/work/FileInfoPrefs.xml`

This file contains the top-level dialog preferences as attributes of the top-level tag:

```
<xfi:fileinfo
    modifyDate="2008-07-01T16:52:30.098"
    importOptions="ar"
    currentLocale="en_US"
    topmostPanel="description"
        ...
```

This tag then contains copies of the `<panel>` tag from the manifest for each panel:

```
    <xfi:panel name="description" ... /> <-- first panel
    <xfi:panel name="IPTC" ... /> <-- second panel
    ...
```

The panels appear in the dialog in the same order as their tags appear in this preferences file, with the first one at the left of the dialog. You can change the order of panels by editing this file (as well as by dragging and dropping panels interactively in the dialog).

In this context, the `<panel>` element can contain a `<prefs>` element, which stores any user preferences you have defined for your own panel. For example, this is how the preferences created in the example are stored:

```
<xfi:panel name="BasicControlSamplePanel" ...>
    <prefs>
        <divider>104</divider>
        <bgColor>#33ff99</bgColor>
    </prefs>
</xfi:panel>
```

NOTE: The `<prefs>` element does *not* appear in the panel manifest; it is used only within the preferences file. To create preferences, you must use the `PanelDescription` object.

You can remove the preferences file to restore defaults to the entire File Info dialog; it is automatically recreated. If there is any problem with the dialog (such as a custom panel not appearing), it is a good idea to remove the preferences file and reinvoke the dialog.

XMP Flex components

The XMP FileInfo SDK includes the component library `FileInfoFoundation`, which extends the basic Flex component set with components and methods that are specialized for use within the File Info dialog. All components named with the "XMP" prefix automatically perform data exchange with the XMP Packet of the file being examined.

- The base component, `XMPForm`, is derived from the standard Flex `Form` component, with added handling for XMP metadata. It can contain UI controls that connect directly to XMP fields defined in the file, through the `xmpPath` property.
- An `XMPFormItem` extends the standard Flex `FormItem`, a small container that contains a label and space for a choice or input control, which aligns the labels properly within the containing form. The XMP extension adds the ability to localize label string using the ZString format, and the `labelTooltip` property which provides access to an XMP property description.
- Display controls such as `XMPLabel` and `XMPImage` allow you to describe properties and display existing values.
- Choice and entry controls such as `XMPTextInput` and `XMPComboBox` allow you to gather new values from the user with which to update XMP properties.

XMP component behavior

The File Info dialog reads the XMP Packet for the file when it is first displayed, and updates any custom components at that time. It does not write back to the dialog is closed with the OK button. Any changes that the user makes in the dialog are written to a working packet.

- When the user accepts the changes by dismissing the dialog with the OK button, the working packet is written to the file.
- If the user cancels the dialog, the file's original XMP Packet is not modified.

Typically, the only thing you need to do is provide the `xmpPath` and `xmpType` values for the XMP property to be associated with a control. The XMP components have built-in default responses to events defined for the `XMPForm`, which include `XMPRead`, `XMPWrite`, and `XMPModification`. These occur when the dialog is opened or closed, but also when the user switches to a new panel. By default, the handlers read from and write to the working packet, keeping those values synchronized with the control values.

The data flow of XMP values goes like this:

1. When the File Info dialog is displayed, the XMP Packet is copied into a working packet.
2. Each time a panel is displayed, either by opening the dialog or by tabbing through the panels, XMP values are read from the working packet.
3. When a different panel is selected or the dialog is closed, the working packet is updated with current values from the currently-shown panel's controls.
4. When the dialog is dismissed, the file's XMP Packet is either replaced by the working packet (if OK was clicked), or the original is restored (if Cancel was clicked).

TIP: *This mechanism allows different panels to display the same property -- that is, be associated with the same XMP path-- and still stay synchronized when the user switches panels. However, you must not connect more than one control with the same XMP path on the same panel.*

It is possible to override the event handlers for specific controls in order to achieve more complex behavior. For example, a panel could use values entered for the GPS coordinate fields to update a map, or update the coordinate values from the position of a location marker in the map.

Most-recently-used (MRU) lists

The component library includes special text components that give the user a choice of most-recently-used (MRU) values in a pop-up menu. There is an editable-text version (`XMPTextInputMRU`) that allows the user to enter values, which are automatically added to the list, and a non-editable version (`XMPTextAreaMRU`). These components have the same appearance as the `XMPTextInput` and `XMPTextArea`, but different behavior.

For example, to provide an MRU menu for the non-editable unordered-array property, `mybag`, use the component `XMPTextAreaMRU`:

```
<fi:XMPFormItem label="$$$/xmp/my/mybag=My Bag"
    labelTooltip="$$$/xmp/my/myproperty_tooltip=Show the use of a Custom Schema"
    id="_mybag" visible="false">
    <mx:VBox width="100%">
        <fi:XMPTextAreaMRU xmpPath="my:mybag" xmpArray="bag"/>
    </mx:VBox>
</fi:XMPFormItem>
```

In this case, the MRU menu is automatically populated with the array elements.

To provide an MRU menu that collects entered values for the editable simple-text property, `mytext`, use the component `XMPTextInputMRU`:

```
<fi:XMPFormItem id="_mytext" label="$$$/xmp/my/mytext=My Text"
    labelTooltip="$$$/xmp/my/myproperty_tooltip=Show the use of a Custom Schema"
    visible="false" includeInLayout="false" >
    <fi:XMPTextInputMRU xmpPath="my:mytext"/>
</fi:XMPFormItem>
```

In addition to providing the pop-up menu and populating it appropriately, the MRU component saves the collected values in XML format, so that they can be restored in subsequent sessions. The values are saved in the File Info dialog's application-data location for the individual user. For example, in Windows XP:

```
C:\Documents and Settings\<username>\Local Settings\Application Data\Adobe\XMP\File
Info\3.0\work\FileInfoMRU.xml
```

All MRU values are stored in the single XML file, `FileInfoMRU.xml`, in the user space. This is an editable XML file, which you can modify to provide new choices programmatically. For the two example components, the XML file might look like this:

```
<xfi:fileinfo xmlns:xfi="http://ns.adobe.com/xmp/fileinfo/">
    <xfi:mruLists>
        <xfi:mruList key="my:mytext" maxEntries="20">
            <item>Some Trees</item>
        </xfi:mruList>
        <xfi:mruList key="my:mybag" maxEntries="20">
            <item>bag 1</item>
            <item>bag 2</item>
            <item>bag 3</item>
        </xfi:mruList>
    </xfi:mruLists>
</xfi:fileinfo>
```

The MRU choices provide completion in the editable-text component. Completions are selected alphabetically, ignoring case.

An MRU list defaults to 20 entries. The most recent values are kept. If a list is already full, the oldest value is dropped. You can set the maximum number of entries programmatically; see the API Reference documentation. If you set the number of entries to zero, you disable the MRU menu for the component.

The API allows you to predefine MRU lists using the `dataProvider` object. You can also specify a *closed list* where no entries will be added.

Installing custom panels

Each custom panel definition must reside in a single panel-root folder, whose name is specified in the contained manifest. Delivery files must include:

- An XML panel manifest, named `manifest.xml`. See "Panel manifests" on page 47.
- A Flash (SWF) file created by compiling the MXML source file.
- Translation dictionaries for any localized strings (ZStrings) used in the panel, in a subfolder named `loc`. (The use of ZStrings is optional; if they are not used, the `loc` folder is not needed.)

For delivery, the folder must be installed in the Adobe application-data location for XMP, as defined for the platform. There is no installer; the developer is responsible for installing the correctly packaged set of panel files in the proper location.

- For single-user plug-ins, this is in a user-specific location:

  IN WINDOWS XP: `C:\Documents and Settings\<username>\Application Data\Adobe\XMP\Custom File Info Panels\3.0\panels\<PANEL_ROOT>`

  IN WINDOWS 7/VISTA: `C:\Users\<username>\AppData\Roaming\Adobe\XMP\Custom File Info Panels\3.0\panels\<PANEL_ROOT>`

IN MAC OS: `/Users/<username>/Library/Application Support/Adobe/XMP/Custom File Info Panels/3.0/panels/<PANEL_ROOT>`

- If your panel is part of a plug-in that is installed with administrative privilege, the panel plug-in should be installed in the main XMP application-data location:

  IN WINDOWS XP: `C:\Program Files\Common Files\Adobe\XMP\Custom File Info Panels\3.0\panels\<PANEL_ROOT>`

  IN WINDOWS 7/VISTA: `C:\Program Files (x86)\Common Files\Adobe\XMP\Custom File Info Panels\3.0\panels\<PANEL_ROOT>`

  IN MAC OS: `/Library/Application Support/Adobe/XMP/Custom File Info Panels/3.0/panels/<PANEL_ROOT>`

Trust files for custom panels

A custom panel is compiled as a Flash file, and Flash security requires that the user directory contains a configuration file called a *trust file*. This is a simple text file with a `.cfg` file extension, that identifies Flash files that have been determined to be safe. The trust file contains the location of one or more local Flash (SWF) files. If you specify a path, all SWF files beyond that path are trusted.

If you cannot display a panel in the File Info dialog, create a trust file in the user's `FlashPlayerTrust` folder that includes the XMP application-data folder where the panel is installed.

For example, to run one of the custom panel samples as a trusted file:

1. Create a text file that contains the path and file name of the sample panel, in the Adobe application-data location (with appropriate substitutions):

   IN WINDOWS XP: `C:\Documents and Settings\<username>\Application Data\Adobe\XMP\Custom File Info Panels\3.0\panels\<PANEL_ROOT>\bin\<PANEL_DEF>.swf`

   IN WINDOWS 7/VISTA: `C:\Users\<username>\AppData\Roaming\Adobe\XMP\Custom File Info Panels\3.0\panels\<PANEL_ROOT>\bin\<PANEL_DEF>.swf`

   IN MAC OS: `/Users/<username>/Library/Application Support/Adobe/XMP/Custom File Info Panels/3.0/panels/<PANEL_ROOT>/bin/<PANEL_DEF>.swf`

   This example specifies a particular panel Flash file. To trust everything in the user's panel folder, you can specify just the parent folder (using platform syntax):

   `<USER_DATA_DIR>/Adobe/XMP/Custom File Info Panels/3.0/panels/`

2. Save this text file with the extension `.cfg` in a folder named `FlashPlayerTrust` (which you may need to create), in one of the User Flash Player Trust folder. This is at the following locations, depending on the platform and users for which the SWF should be trusted.

   IN WINDOWS XP: `C:\Documents and Settings\<username>\Application Data\Macromedia\Flash Player\#Security\FlashPlayerTrust`

   IN WINDOWS VISTA: `C:\Users\<username>\AppData\Roaming\Macromedia\Flash Player\#Security\FlashPlayerTrust`

   IN MAC OS: `/Users/<username>/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust`

For more information on trust files see:

http://livedocs.adobe.com/flex/3/html/05B_Security_03.html

5 Using the Flex SDK

The Adobe Flex 3 Software Development Kit (SDK) is available free of charge; see http://www.adobe.com/devnet/flex/. This SDK includes the Flex framework (component class library) and Flex compiler, enabling you to develop and deploy Flex applications using any text editor or IDE of your choice.

This section describes how to build a simple File Info panel using the free Flex SDK and Apache Ant, an open-source software build tool, available from http://ant.apache.org/.

Building a panel with Flex SDK

This exercise requires these files, which are included in the XMP FileInfo SDK:

`FileInfoFoundation.swc`	The Flex XMP component library.
`build.xml`	The Ant build file for compiling your panels.

Creating the panel definition

To create the source for the custom panel, follow these steps:

1. Create a new folder called `TestPanel` in your chosen location.
2. Within this folder create new subfolders called `target` and `src`.
3. In the `target` folder, create a subfolder called `TestPanel`. Within this subfolder, create the file `manifest.xml`.
4. In the `src` folder, create a new file called `TestPanel.mxml`.
5. Open `TestPanel.mxml` in a text editor and add the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<fi:XMPForm
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:fi="com.adobe.xmp.components.*" width="100%" height="100%"
    label="Test Panel">
    <fi:XMPNamespaces>
        <fi:XMPNamespace prefix="dc" value="http://purl.org/dc/elements/1.1/"/>
    </fi:XMPNamespaces>
    <fi:XMPFormItem label="Document Title:" labelTooltip="Title of the document.">
        <fi:XMPTextInput xmpPath="dc:title" xmpType="Localized"/>
    </fi:XMPFormItem>
</fi:XMPForm>
```

Compiling the panel

6. Copy the files `build.xml` and `FileInfoFoundation.swc` from the XMP FileInfo SDK into the `TestPanel` folder.

7. Open the file `build.xml` in a text editor and update the property values so that they match your environment settings:

```
<property name="FLEX_HOME" value="INSERT FLEX SDK LOCATION HERE" />
<property name="panel-dir" value="INSERT NAME OF PANEL HERE (TestPanel)"/>
<property name="build.target" value="USER DIRECTORY/Application Data/
   Adobe/XMP/Custom File Info Panels/3.0/panels"/>
<property name="panel-classes" value="INCLUDE ALL PANELS (for this example
TestPanel)"
```

8. Open the file `manifest.xml` in a text editor and add the following code:

```
<xfi:fileinfo xmlns:xfi="http://ns.adobe.com/xmp/fileinfo/">
    <xfi:panels>
        <xfi:panel
            name = "TestPanel"
            label = "Test Panel"
            type = "custom"
            panelLibrary = "TestPanel"
            panelClass = "TestPanel"
            visible = "true" >
        </xfi:panel>
    </xfi:panels>
</xfi:fileinfo>
```

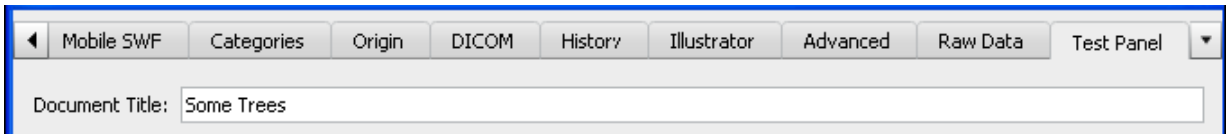
9. Open a command shell and navigate to the main `TestPanel` folder.

10. To build the panel type in the following command and press ENTER.

```
ant build.xml
```

    If the build succeeds, the files are generated directly into the user's delivery folder; see "Installing custom panels" on page 52.

11. To see the result, open a CS5 application, select an item, and open the File Info dialog. Your new panel "Test Panel" appears in the dialog.



6 Porting Guide

This chapter details changes between the CS4 (SDK 2.0) and CS5 (SDK 3.0) release of the XMP File Info SDK and the File Info dialog, for users and developers of custom File Info panels. It is assumed that the reader of this document is familiar with the development of custom panels, as described in this document.

In general, CS4 panels work in the CS5 File Info dialog, but not the reverse. Differences fall into the following categories:

- Flex versions and compatibility
- Installation locations and porting
- Changes to XMP components
- Changes to the FileInfoLibrary API

Flex versions and compatibility

The CS4 File Info dialog and panels use the Flash Engine 9 and the Flex SDK 3.0.0; the CS5 File Info dialog and the panels use the Flash Engine 10 and the Flex SDK 3.3.0. Flash 10 is fully backward compatible with Flash 9; Flex SDK 3.3.0 is an update to Flex SDK 3.0.0 and is also fully compatible.

You must compile your CS5 custom panels with Flex SDK 3.3.x, *not* Flex SDK 4.x, which is a an entirely new version.

Some components may behave slightly differently, due to bug fixes and updates in the newer Flex SDK. In general, CS4 custom panels that contain only standard XMP components should work as expected in CS5. If CS4 panels use more advanced features of the Flex SDK, they will probably work, but cannot be guaranteed to be completely compatible. You should test them in CS5 before delivering them to customers.

Installation locations and porting

The installation locations of the File Info dialog and panels, as shown throughout this document, are distinct in versions 2.0 and 3.0 of the SDK, so that both versions can co-exist. The locations for CS4 are under the `2.0/`folder, while all CS5 locations are under the `3.0/` folder. For example, in Windows XP, panels are in these locations:

CS4: `C:\Documents and Settings\<username>\Application Data\Adobe\XMP\Custom File Info Panels\2.0\panels\`

CS5: `C:\Documents and Settings\<username>\Application Data\Adobe\XMP\Custom File Info Panels\3.0\panels\`

The same rule applies to all platforms. The version folder also contains the `work/` folder, which contains the preferences file (`FileInfoPrefs.xml`) and auto-completion lists (`FileInfoMRU.xml`); and the `custom/` folder, which contains custom schema files.

Updating CS4 custom panels

The CS5 version of the FileInfo dialog does not load custom panels from the CS4 location; if you want to use them, you must copy them to the CS5 location. If you want them to still work in CS4, be sure to copy them, rather than move them, to the new location.

Test the copied panels in any CS5 application that supports the File Info dialog, such as Photoshop. In some cases, you might need to recompile the custom panel against Flex SDK 3.3.

If you need to port a custom panel that you have not created, contact the panel vendor.

Reusing CS4 auto-completion lists

The auto-completion file format has been extended, but is compatible. To reuse an auto-completion list that was created in CS4, make sure that no FileInfo dialog is open, then copy the file `FileInfoMRU.xml` from the `.../2.0/work/` folder to the `.../3.0/work/` folder.

Reusing custom schema files

Custom schema files provide custom schemas to the Metadata panel in Adobe Bridge and in audio-video application such as Premiere, which do not support the File Info dialog. The Generic Panel also uses these files to create a simple custom panel automatically. The file format defined in "XML schema file format" on page 42 is compatible between CS4 and CS5.

To reuse custom schema files created for CS4 in CS5, copy them from the `../2.0/custom/` folder to the `.../3.0/custom/` folder.

Changes to XMP components

- Multiple-file editing in the `XMPComboBox` has been improved. If the selected assets have differing values for the XMP property represented by the combo box, it displays the string "(Multiple Values)".
- Two new components have been added, `XMPDataGrid` and `XMPDataGridColumn`. These present and allow you to modify an array of structs in the XMP data model. The `XMPDataGrid` component is rendered as a table where each line represents a complex array item. Each cell represents a struct field of one array item. The item editor for the cell behaves like an `XMPTextInput` component, and also allows the `xmpType` and `xmpArray` attributes.

Changes to the FileInfoLibrary API

The FileInfoLibrary API is used internally by XMP components to read and write XMP values, and can also be used directly by custom panels. This API is documented as part of the *XMP Toolkit Programmer's Guide*.

In CS5, these functions are new:

- `getArrayItem(schemaNS, arrayName, index)`
- `appendArrayItem(schemaNS, arrayName, itemValue, itemOptions,arrayOptions)`
- `countArrayItems(schemaNS, arrayName)`
- `resetXMP()`

The metadata is reset to its original state, before modification. This has the same effect as cancelling the dialog and re-opening it.

These functions have been changed:

- `setProperty(schemaNS, propName, propValue, options)`

  Can be used to create arrays and structs. Not used in CS4, because it could not take the value "null".

- `separateArrayItems(...)`

  If passed a simple property and an array property with the same name, this function now overwrites the simple property and replaces it with the array.

For complete details of these functions and the rest of the API, see the *XMP Toolkit Programmer's Guide*.