



Adding Intelligence to Media

XMP METADATA UI DEVELOPER'S GUIDE

Copyright © 2014 Adobe Systems Incorporated. All rights reserved.

XMP Metadata UI Developer's Guide.

Adobe, the Adobe logo, Creative Cloud, Creative Suite, Flash, Flex, InDesign, InCopy, Illustrator, Photoshop, Premiere, and Prelude are either registered trademarks or trademarks of Adobe Systems Inc. in the United States and/or other countries. Microsoft and Windows are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Apple, Mac OS, and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. Java and Sun are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All other trademarks are the property of their respective owners.

The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Inc. Adobe Systems Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this document. The software described in this document is furnished under license and may only be used or copied in accordance with the terms of such license.

Adobe Systems Inc., 345 Park Avenue, San Jose, California 95110, USA.

Contents

	Preface	5
	About this document	5
	How this document is organized	5
	Conventions used in this document	6
1	Defining Customizations	7
	Terms and concepts	7
	Structure of a Metadata UI Extension package	7
	Delivering customizations	8
	Mapping property types to UI components	9
	Default mapping of simple types	9
	Default mapping of complex types	10
	Error handling	11
	Panel Conversion Tool	15
2	Sample Metadata UI Extension Package	17
	Sample panels	17
	Sample package files	17
	Viewing the sample panels	18
	Schema sample	18
	Property definitions	19
	Custom type definitions	19
	View samples	21
	All Properties view	21
	Override Defaults view	22
	Layout Elements view	23
	Default Schemas view	24
3	Metadata UI Extension Formats and Schema	26
	File formats	26
	Schema file format	26
	View file format	27
	Localization file format	28
	Manifest file elements	29
	extension tag	29
	Schema file elements	29
	xmp_definitions	30
	xmp_schema	30
	xmp_property	30
	xmp_choice	31
	xmp_type	32
	xmp_field	32

View file elements	34
views	34
view	34
property	35
choice	36
section	36
formatter	37
separator	38
note	38

Preface

The XMP Metadata UI (called the FileInfo dialog in most applications) is a component that is incorporated into many Adobe® Creative Cloud® desktop applications, including InDesign®, Photoshop®, and Illustrator®. It provides access to file metadata information through an integrated dialog or panel in the host application.

XMP is extensible, allowing developers to define new metadata properties in addition to those defined in built-in schemas. The XMP Metadata UI component is therefore also extensible. You can easily modify it to create and display custom metadata properties, or create new views for properties defined in the built-in schemas.

To customize the XMP Metadata UI, you create a Metadata UI Extension package, which is a collection of XML files in a specific structure as defined in this document. You then install the package in a global location where it can be accessed by the XMP Metadata UI, regardless of where the dialog or panel is invoked.

If you have created and added custom generic panels for the XMP FileInfo dialog in previous releases, a migration path is provided that allows you to automatically convert these panels to a suitable Metadata UI Extension file.

About this document

This document, the *XMP Metadata UI Developer's Guide*, provides guidance for developers wishing to customize the XMP Metadata UI, incorporated into Adobe desktop applications. These customizations allow users to display and modify specific XMP metadata properties that are important to their workflow.

This document provides the XML file format details and schema specifications for customization.

This document assumes that the reader is familiar with Adobe XMP and XML technology. For further information about XMP, see <http://www.adobe.com/devnet/xmp/>.

How this document is organized

This document has the following sections:

- ▶ [Chapter 1, "Defining Customizations"](#), provides an overview of the terms and concepts you need to create a Metadata UI Extension package, and provides details of how to convert FileInfo Generic Panels defined for previous releases of Adobe desktop application to the new Metadata UI Extension format.
- ▶ [Chapter 2, "Sample Metadata UI Extension Package"](#), introduces the sample customization that is part of the SDK, showing the XML sources for the Metadata UI Extension package, and the result in the XMP Metadata UI.
- ▶ [Chapter 3, "Metadata UI Extension Formats and Schema"](#), provides a reference for the formats of the XML files that you use to define customizations to the XMP Metadata UI.

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.
<i>Monospaced italic</i>	Placeholders or variables in code or paths, to be replaced by the user with appropriate values.

1 Defining Customizations

This chapter provides a definition of terms and concepts for the Metadata UI Extension SDK that you use to define customizations to the XMP Metadata UI.

Terms and concepts

A set of metadata properties in its own XMP namespace is known as an *XMP schema*. There are a number of XMP schemas defined as public standards, and by Adobe for specific products. For more information about the predefined schemas, see the [XMP Specification](#).

The XMP Metadata UI that is incorporated into many Adobe Creative Cloud desktop applications displays and allows users to edit XMP metadata for files that the application works with. The predefined schemas are globally available to the XMP Metadata UI, and by default, the UI displays properties of those schemas in selectable *views*.

In addition to displaying predefined views into built-in metadata, the UI looks for XML files that define new schemas and new views. The Metadata UI Extension SDK allows you to create these customizations and deploy them as *Metadata UI Extension packages*.

- ▶ Your Metadata UI Extension package can define new XMP schemas, regardless of how or whether the properties are displayed in the UI.
- ▶ Your Metadata UI Extension package can define new views that display different sets of properties from the built-in schemas, as well as properties from the custom XMP schemas that you define. You can combine properties from different schemas in a single view.
- ▶ Your Metadata UI Extension package can supply localization information for the display of your customized views in supported locales.

Structure of a Metadata UI Extension package

A Metadata UI Extension package contains a set of XML files that define which schemas and properties should be included, and how those properties should be displayed. In addition to the basic display details, you can provide localization information for display in different languages. Each set of customizations is contained in a separate folder, and a manifest file provides a description of what is contained.

The extension package has this file structure:

```
[package directory]/
  manifest.xml
  schema/
    schema1.xml
    schema2.xml
    ...
  view/
    view1.xml
    view2.xml
    ...
```

```
loc/
  [locFilePrefix]_en_US.dict
  [locFilePrefix]_de_DE.dict
  ...
```

- ▶ The manifest describes the contents of the package.
- ▶ The `schema` folder contains one or more XML files that specify individual XMP schemas that can be included in the XMP Metadata UI. Each file specifies the properties of one XMP schema.
- ▶ The `view` folder contains one or more XML files that specify view definitions. A view definition determines the mapping between a set of XMP properties (from predefined schemas or those defined in the schema files) and their appearance and layout in the XMP Metadata UI.
- ▶ The `loc` folder contains the localization dictionaries for supported language versions of the defined views.

The package does not require all of these files. You can supply only the schema definitions or only the view definitions, or both, and the localization mappings are optional.

For example, if you do not define your own metadata schema, you can still define a specific view into the metadata from one or more of the predefined schemas, to facilitate a particular workflow. To do this, you can create a package that contains only the view definition files that reference properties in the predefined schemas.

If you do define your own custom XMP metadata schema, your package must contain at least a schema definition file. If you want users to be able to view or edit metadata in your custom schema, you must also provide corresponding view definitions. However, if you offer your custom schema to other developers, you can provide a package that contains just the schema definition and let developers add their own view definitions.

The XMP Metadata UI displays predefined views of predefined schemas. For a listing of predefined schemas and their properties, see the [XMP Specification](#). The properties of predefined schemas are globally available, and you can create customized views for them. For example, to add a new view with all the Dublin Core properties, you could create a Metadata UI Extension package which contains a view definition that references the properties in the Dublin Core schema definition.

Delivering customizations

To deliver your extension package and make it available to all Adobe desktop applications that support the XMP Metadata UI, install it in either the shared or user-specific Metadata Extensions folder:

- ▶ User-specific locations:

WINDOWS: [user]\AppData\Roaming\Adobe\XMP\Metadata Extensions

MAC OS: [user]/Library/Application Support/Adobe/XMP/Metadata Extensions

- ▶ Shared locations:

WINDOWS: C:\Program Files (x86)\Common Files\Adobe\XMP\Metadata Extensions

MAC OS: /Library/Application Support/Adobe/XMP/Metadata Extensions

Mapping property types to UI components



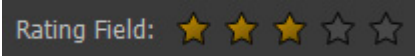



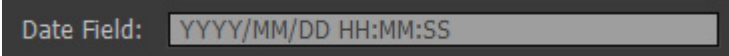
XMP defines a set of basic property types (such as text, integer, boolean) and complex types based on these basic ones. Your own schema definition can also define and use custom types based on those pre-defined types.

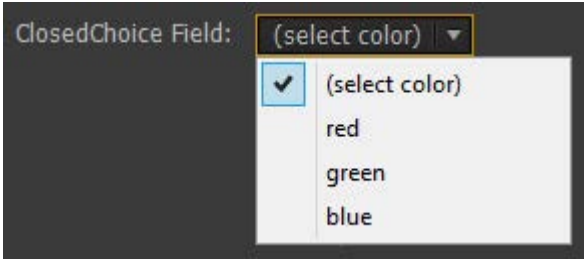
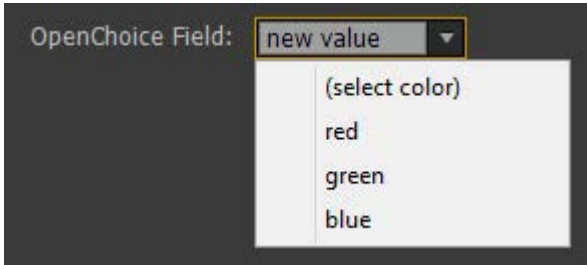


The property type determines what kind of widget can be used to display and edit the value of a property. The XMP Metadata UI has a default mapping between defined types and suitable widgets. You can use the default widget, or, in some cases, you can supply a view element to override the default.

Custom types are based on or composed of types defined in the XMP specification. The XMP Metadata UI defines widgets that are suitable for displaying such values, and by default constructs these widgets according to the type definition.

Default mapping of simple types

The widget types are used by default to display properties of particular types. Your view definitions can customize details of the presentation.

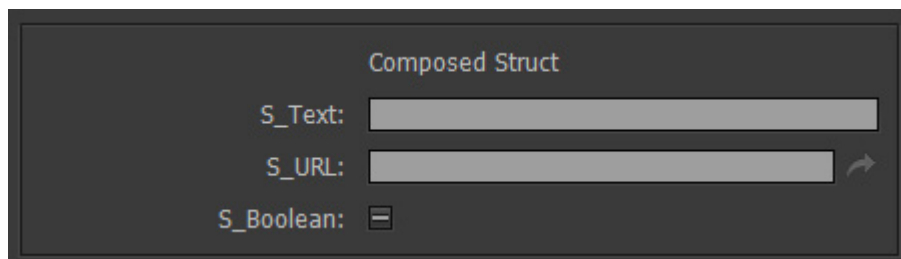
Property Type	UI Widget	Snapshot
text langalt	A single-line or multiline text control. For a <code>langalt</code> property, the <code>x-default</code> variant appears as the default value.	
integer real	A control that allows the user to enter integer or real numbers, and does not accept non-numeric input.	
rating	A control that maps integer values in the range 0 to 5 to highlighted stars.	
boolean	A simple checkbox for values true, false, and unknown (property not set). Initial state is unknown. Once a true or false value has been selected, the state cannot be reverted to unknown.	 unknown  true  false
date	A text widget with type validation for date values in a specific format. Displays a warning on invalid entry.	

Property Type	UI Widget	Snapshot
closedchoice	A simple dropdown with the list of allowed choices.	
openchoice	A dropdown with the list of specified choices, that also allows the user to enter a different value.	
url	Text widget with a button that opens the URL in the default browser.	
bag seq	A text component with individual element values rendered as a comma or semicolon separated list.	

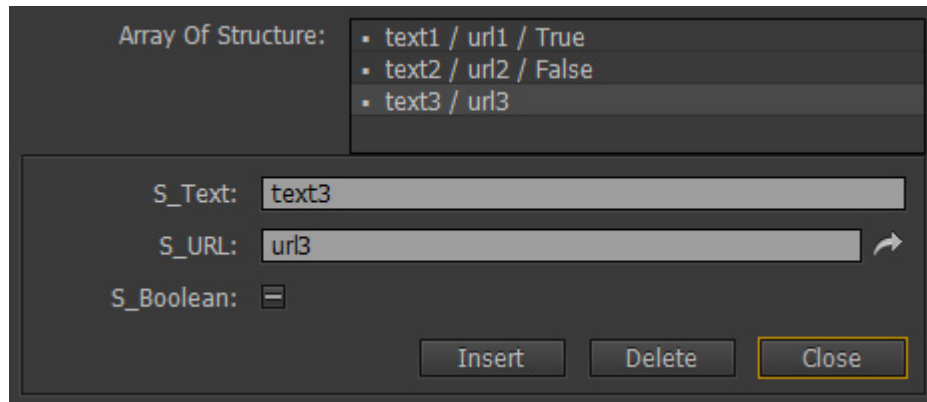
Default mapping of complex types

Complex types contains multiple values or fields that can in turn contain multiples values. By default, properties with complex values are mapped to widgets as follows. Again, your view definition can customize specific details of the mapping.

- If a custom property is defined as a structure that contains multiple fields, then the UI widgets are inherited from the base types from which the type is derived. For example:

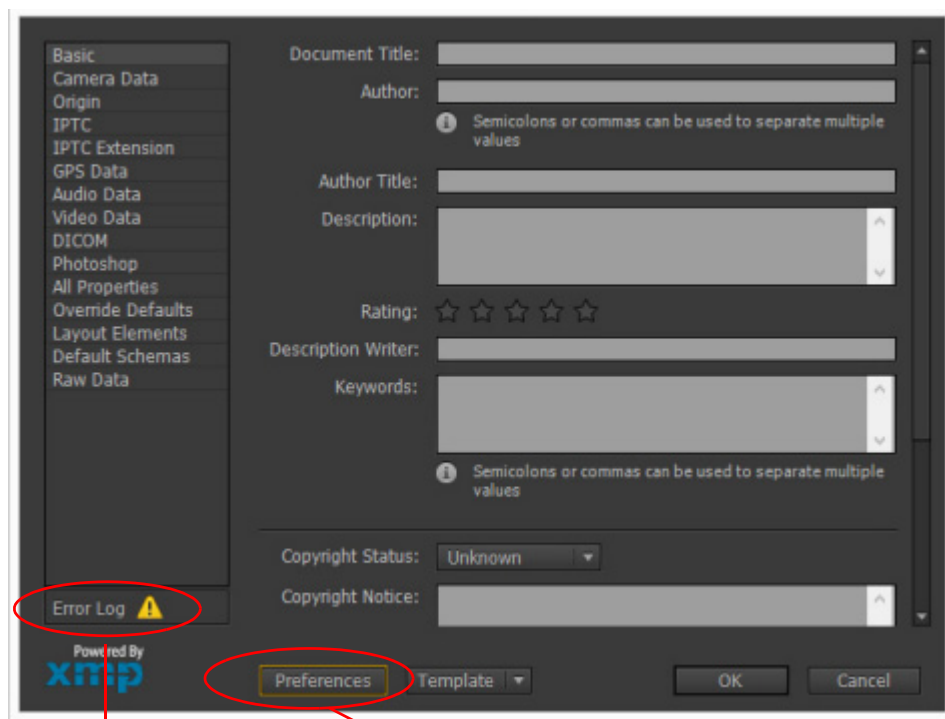


- For arrays of structure types, a specially defined control renders the multiple values, which shows a top-level summary for the array, and also allows editing of individual entries, as in this example:



Error handling

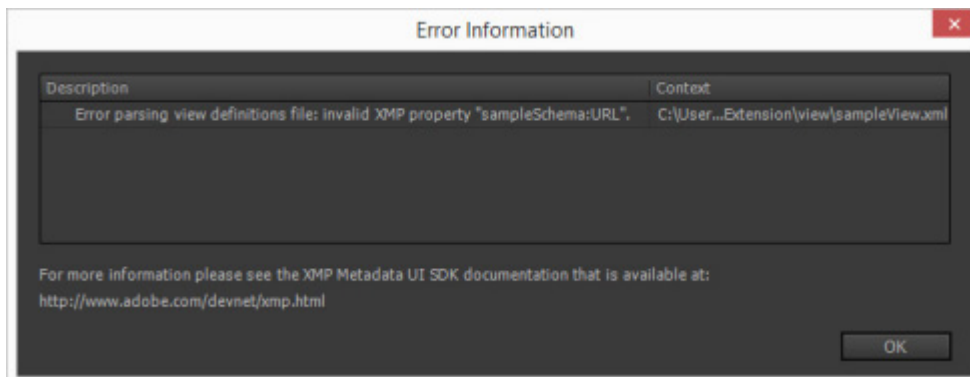
Errors that occur when attempting to load a custom panel are reported for debugging purposes. By default, debugging errors are not visible to the user. To see the error notifications and get access to the error panel, open the dialog's Preferences panel and select the 'Enable Debug Mode' option.



Open error panel when
debug mode is enabled

Set debug mode

When debug mode is enabled, the error notification icon is shown whenever a custom-panel load error occurs. Click the notification icon to display the error panel. The Context column shows the Metadata UI Extension package file in which the error occurred.



These error messages can be reported in the error panel:

Error Message	Meaning
Outermost Tag has to be <views/>	All the items in the property file should be put inside a <views> tag
Schema definition for " <i>namespace</i> " already exists	For the given namespace the schema has been already defined. Only the first definition will be honored.
Localization directory is missing from extension package	Localization prefix is given in the manifest file but the Loc folder is missing from the extension package
Unknown error	Some unknown parser errors occurs.
XMP property " <i>property name</i> " has an invalid type " <i>type</i> "	Some unknown type is being used with the property name
Found XMP property without a 'name' attribute	The <code>name</code> attribute is missing from the property
Invalid element type for XMP property " <i>property name</i> " is changed to default type 'text'	The <code>element_type</code> attribute of the specified property is either unknown or not supported; replaced with the default value "text"
Invalid element type for type " <i>type name</i> " is changed to default type 'text'	The <code>element_type</code> attribute of the type is not known or supported; replaced with the default value "text"
Manifest file is missing from extension package	The extension package does not contain manifest file. The package is ignored
Neither schema nor view file is present in extension package	The extension package must contain a view or schema or both.
Invalid prefix in XMP property " <i>property</i> "	The prefix defined for the property is not correct; could be a typo.
Invalid XMP property " <i>property</i> "	The property name defined is not correct. There could be a typo or the property could have failed to register

Error Message	Meaning
XMP type " <i>type name</i> " has an invalid element type " <i>element type name</i> "	The type name specified has unknown or unsupported element type attribute
Entry must extend valid primitive type	The value of the "extends" attribute is not a primitive type
Missing namespace "http://ns.adobe.com/metadata/ui/1.0/", views in the file will be skipped	The view file must include http://ns.adobe.com/metadata/ui/1.0/ as the default namespace; if not present, view is skipped
Invalid XML	The XML is not well formed
Invalid data in " <i>property name</i> "	There is a data in the XMP Packet but its format is incorrect
Namespace " <i>namespace name</i> " couldn't be registered with prefix " <i>prefix name</i> "	There is some other namespace already registered with the given prefix
Invalid field tag " <i>field name</i> "	An unknown tag is used inside the <xmp_type> tag while defining the fields for that structure
Invalid property tag " <i>tag name</i> "	An unknown tag is used in the schema file inside the <xmp_property> tag.
Invalid qualifier tag	An unknown tag is used in the schema file
Invalid definitions tag " <i>tag name</i> "	An unknown tag is used inside the schema definition file
Invalid structure in XMP property " <i>property name</i> "	Error in the definition for a custom XMP type (a structure)
View definition for " <i>view name</i> " already exists	Two view definitions are found with the same name; only first one is honored
Invalid type " <i>type name</i> "	A type used in the schema file is unknown
Type definition for " <i>type name</i> " already exists	A type with the given name is already registered; only the first one is honored
The view definition file has no <view> tag	View definitions file does not have any <view> tag where all the properties are to be defined.
<view> tag needs a name attribute	A view without a name is ignored.
XMP property " <i>property name</i> " has an invalid element type " <i>element_type name</i> "	An invalid <code>element_type</code> is used with the <code>type bag</code> or <code>seq</code> . The only allowed values are the primitive types or a custom type.
Invalid type for XMP property " <i>property name</i> " is changed to default type 'text'	Incorrect value for the <code>type</code> attribute of a property; replaced with the default value "text"
Invalid structure field in XMP property " <i>property name</i> "	Incorrect value in one of the fields for a structured property definition.

Error Message	Meaning
Error reading XMP data: invalid data in " <i>property name</i> "	Existing data in the XMP Packet does not match the data type specified for the property in the schema.
Extension package with name " <i>package name</i> " already exists	An extension package with the same name and same version number is already registered. The duplicate package is ignored.
Extension package with name " <i>package name</i> " already exists. Package name with lower version number " <i>number</i> " will be ignored.	An extension package with the same name and a higher version number is already registered. The package with the lower version number is ignored.

Panel Conversion Tool

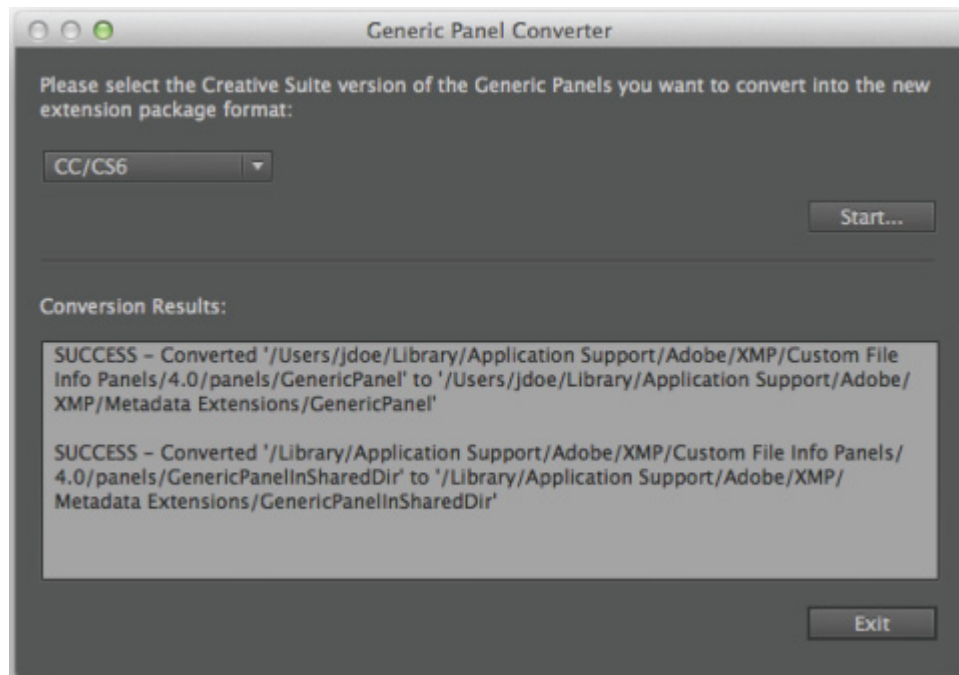
For the FileInfo dialog that was used to display metadata in Creative Suite desktop applications, the Generic Panel feature allowed developers to provide an XML specification for a custom panel. The XML format used for these specifications was similar to the one used for the new Metadata UI Extension format, but did not separate schema and type information from view information.

The Metadata UI Extension SDK includes the Panel Conversion tool, which you can use to automatically convert a Generic Panel specification that was created for the previous FileInfo dialog into the new Metadata UI Extension format.

The Panel Conversion tool is a stand-alone application (`panelconverter`). Note that in order to convert panels located in the Shared folder, you must explicitly start the tool with administrator privileges:

- ▶ In Mac OS: Before running the tool with administrator privileges, you must create the folder `/Library/Application Support/Adobe/XMP/Metadata Extensions`
- ▶ In Windows: Launch the application with the command **Run as Administrator**.

The tool opens a modal dialog, which you use to select the Creative Suite version from which to convert and start the conversion.



The dialog reports any problems encountered during the conversion, and the results. Here are some examples of conversion results:

```
SUCCESS - Converted "Users/<username>/Library/Application Support/Adobe/XMP/Custom
File Info Panels/4.0/panels/SamplePanel" to "Users/<username>/Library/Application
Support/Adobe/XMP/Metadata Extensions/SamplePanel".
```

```
INFO - Cannot convert "Users/<username>/Library/Application
Support/Adobe/XMP/Custom File Info Panels/4.0/panels/SamplePanel". Extension
Package folder "Users/<username>/Library/Application Support/Adobe/XMP/Metadata
Extensions/SamplePanel" already exists.
```

INFO - No Generic Panels found in folder "/Library/Application Support/Adobe/XMP/Custom File Info Panels/4.0/panels/".

FAILURE - Cannot convert "/Library/Application Support/Adobe/XMP/Custom File Info Panels/4.0/panels/AllUserSamplePanel". Extension Package folder "/Library/Application Support/Adobe/XMP/Metadata Extensions" cannot be created. Please make sure that you have write permissions.

FAILURE - The folder 'Metadata Extensions' cannot be created. Please make sure that the folder "/Library/Application Support/Adobe/XMP/Metadata Extensions" exists.

2 Sample Metadata UI Extension Package

This chapter provides an overview of the sample Metadata UI Extension package that is provided with the SDK, and shows the resulting customizations in the XMP Metadata UI.

To try the sample package, simply copy the files to one of the deployment locations on your system and open the FileInfo dialog in an Adobe desktop application that supports it, such as InDesign, Photoshop, or Illustrator. See [‘Viewing the sample panels’ on page 18](#).

Sample panels

The sample defines a custom XMP schema that contains properties of all types, then references that custom schema to define four new panel views that illustrate presentation techniques:

- ▶ All Properties View: Renders all the properties described in the sample schema and demonstrates different types of widgets used for different data types.
- ▶ Override Defaults View: Illustrates how to override various aspects of default presentations.
- ▶ Layout Elements View: Illustrates layout elements, such as sections and separators, that can be used to organize the view elements.
- ▶ Default Schemas View: Illustrates how to create new views to access a set of properties defined in built-in schemas, and mix properties of built-in schemas and custom schemas in a single view.

The sample Metadata UI Extension package also illustrates localization, providing translation files for English and German.

Sample package files

The sample Metadata UI Extension package includes these components:

<code><pkg_root>/sample/manifest.xml</code>	The manifest that describes the contents of the package and specifies the prefix used for localization.
<code><pkg_root>/sample/schema/sampleSchema.xml</code>	A custom XMP schema that defines properties of all types.
<code><pkg_root>/sample/view/sampleViews.xml</code>	A set of panel definitions that reference the properties in the custom XMP schema in order to illustrate various presentation techniques.
<code><pkg_root>/sample/loc/sample_en_US.dict sample_de_DE.dict</code>	Translation files for ZStrings used in the views, for English and German locales.

Viewing the sample panels

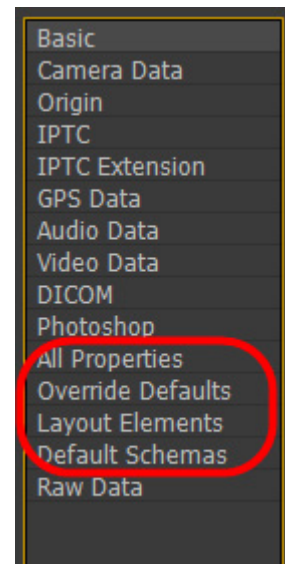
To see the results of these definitions in an Adobe application, install the Metadata UI Extension package in one of the user-specific or general shared resource folders:

WINDOWS: [user] \AppData\Roaming\Adobe\XMP\Metadata Extensions
C:\Program Files (x86)\Common Files\Adobe\XMP\Metadata Extensions

MAC OS: [user]/Library/Application Support/Adobe/XMP/Metadata Extensions
/Library/Application Support/Adobe/XMP/Metadata Extensions

Open a desktop application such as InDesign, Photoshop, or Illustrator, and invoke the metadata dialog (**File > FileInfo**) to see the custom panels.

The list of available panels on the left includes the four panels defined in the sample package. When you select one of these panels, the defined view appear at the right of the dialog.



Schema sample

The example schema definition is named `sampleSchema`. It provides examples of each of the allowed property types. The simple, or primitive types are those shown in [‘Default mapping of simple types’ on page 9](#).

The schema also demonstrates the use of composed types. It creates a new type by extending the primitive type `openchoice`. It also shows how to group different primitive types into a structure.

The sample shows the basic structure of a schema definition file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Top-level container for individual schema definitions in this file-->
<xmp_definitions>

  <!-- Top-level for an individual schema definition, with namespace and prefix-->
  <xmp_schema prefix='sampleSchema'
    namespace='http://my.sampleSchema.namespace/'
    label='Sample Schema for Demonstration'
    description='This example demonstrates how a new schema can be created'>

    <!-- Property definitions-->
  </xmp_schema>
```

```

    <!-- Custom type definitions -->
    <xmp_type>
      <!-- Type definition-->
    <xmp_type>
  </xmp_definitions>

```

Property definitions

The property definitions in the sample schema illustrate how to define properties of all the simple types. For example:

- Simple types are defined with localizable strings:

```

<xmp_property name='Boolean' category='external' type='boolean'
  label="$$$/sampleSchema/Property/BooleanInputLabel=Boolean Field1"
  description='$$$/sampleSchema/Property/BooleanInputDescription=
    Defining boolean data type'/>

```

- Choice types must contain the selectable choices:

```

<xmp_property name='OpenChoice' category='external' type='openchoice'
  element_type='text'
  label="$$$/sampleSchema/Property/OpenChoiceInputLabel=OpenChoice Field"
  description='$$$/sampleSchema/Property/OpenChoiceInputDescription=
    Defining openchoice data type'>
  <!-- Define choices-->
  <xmp_choice raw_value=""
    label="$$$/sampleSchema/Property/Choice_NoColor=(select color)"/>
  <xmp_choice raw_value="red"
    label="$$$/sampleSchema/Property/Choice_Red=red"/>
  <xmp_choice raw_value="green"
    label="$$$/sampleSchema/Property/Choice_Green=green"/>
  <xmp_choice raw_value="blue"
    label="$$$/sampleSchema/Property/Choice_Blue=blue"/>
</xmp_property>

```

Custom type definitions

There are two kind of custom types; those that extend a primitive type, and those that define a structure, with fields of different types.

- Extending a primitive type

The sample schema defines two custom types that extend the choice primitives by specifying a particular set of choices:

```

<xmp_type name='openchoice_type' extends='openchoice' element_type='text'>
  <xmp_choice raw_value='default'
    label='$$$/sampleSchema/Property/C_Choice_Default=Default'/>
  <xmp_choice raw_value='choice1'
    label='$$$/sampleSchema/Property/C_Choice_Choice1=Choice 1'/>
  <xmp_choice raw_value='choice2'
    label='$$$/sampleSchema/Property/C_Choice_Choice2=Choice 2'/>
  <xmp_choice raw_value='choice3'
    label='$$$/sampleSchema/Property/C_Choice_Choice3=Choice 3'/>
</xmp_type>

```

The property definition that uses this type does not need to specify the choices, because they are part of the type:

```
<xmp_property name='ExtendsOpenChoice' category='external' type='openchoice_type'
  label="Extends OpenChoice"
  description="Data Type Extending openchoice"/>
```

► Defining a structure type

The `<xmp_type>` element that defines a structure must contain `<xmp_field>` elements for each field in the structure. Each field specifies a property type for that field:

```
<xmp_type name='struct_type'
  description='Description=Defining Structure' label='Struct Type'>
  <xmp_field name='s_text'
    prefix='stSample'
    namespace='http://my.sampleSchema.namespace/struct#'
    type='text'
    label='S_Text' description='Text Field in Struct'/>
  <xmp_field name='s_url'... type='url'.../>
  <xmp_field name='s_boolean'... type='boolean'.../>
</xmp_type>
```

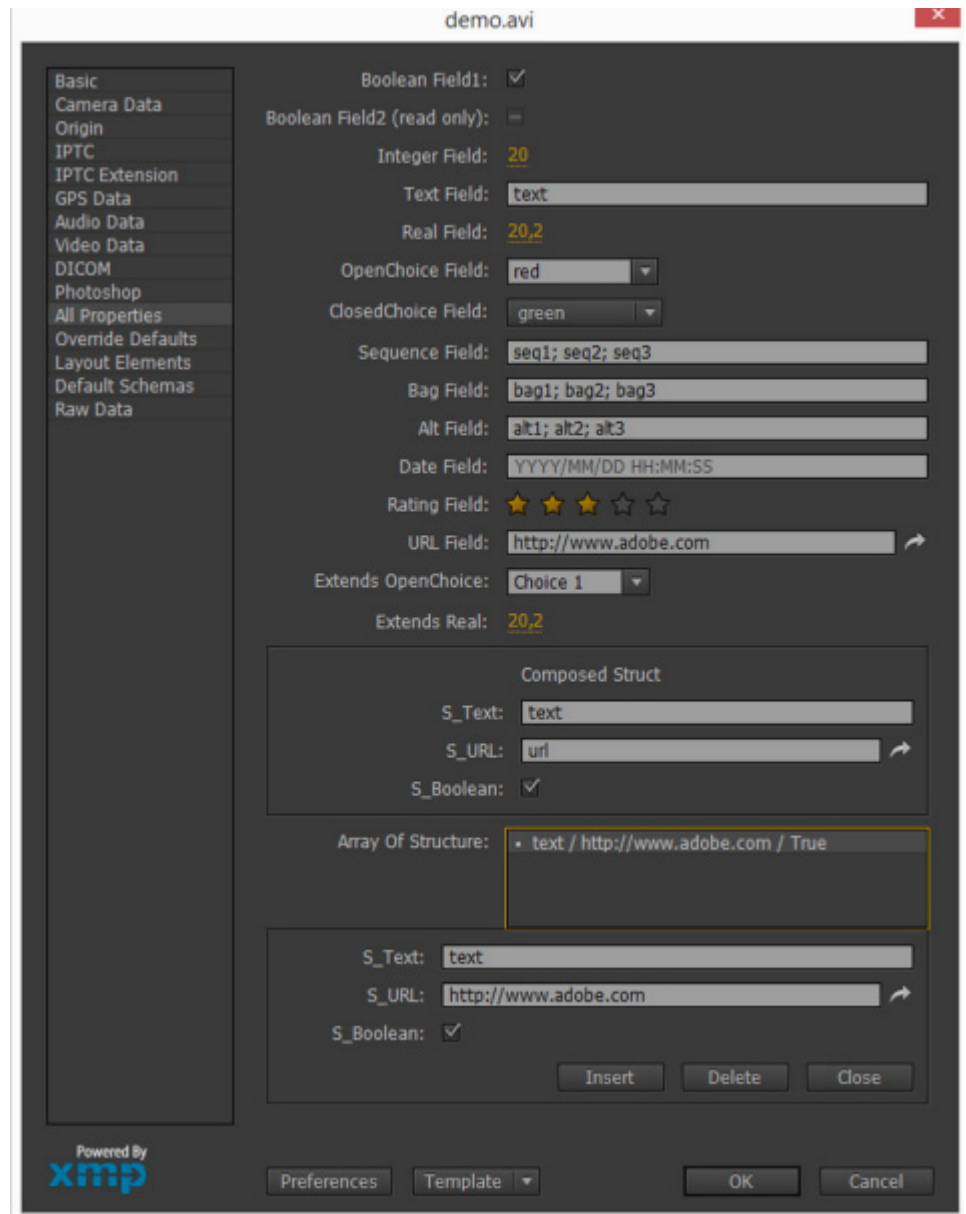
In this sample, the fields all have primitive types, but you can also create more complex types by specifying structure types for fields within a structure.

View samples

The single view-definition XML file defines four different views, all of which reference properties that are defined in the sample XMP schema. One of the views mixes these custom property presentations with presentations of selected properties from globally-available built-in XMP schemas.

All Properties view

This panel demonstrates the different types of widgets used by default for all of the different property types.



This view element defines a panel that renders all the properties defined in the sample schema:

```
<view xmlns:sampleSchema="http://my.sampleSchema.namespace/"
      name="sample.allPropsView"
      label="$$$sampleSchema/Property/AllPropsViewLabel=All Properties"
      description="$$$sampleSchema/Property/AllPropsViewDescription=
        This view displays all the properties defined in the sampleSchema">
```

```

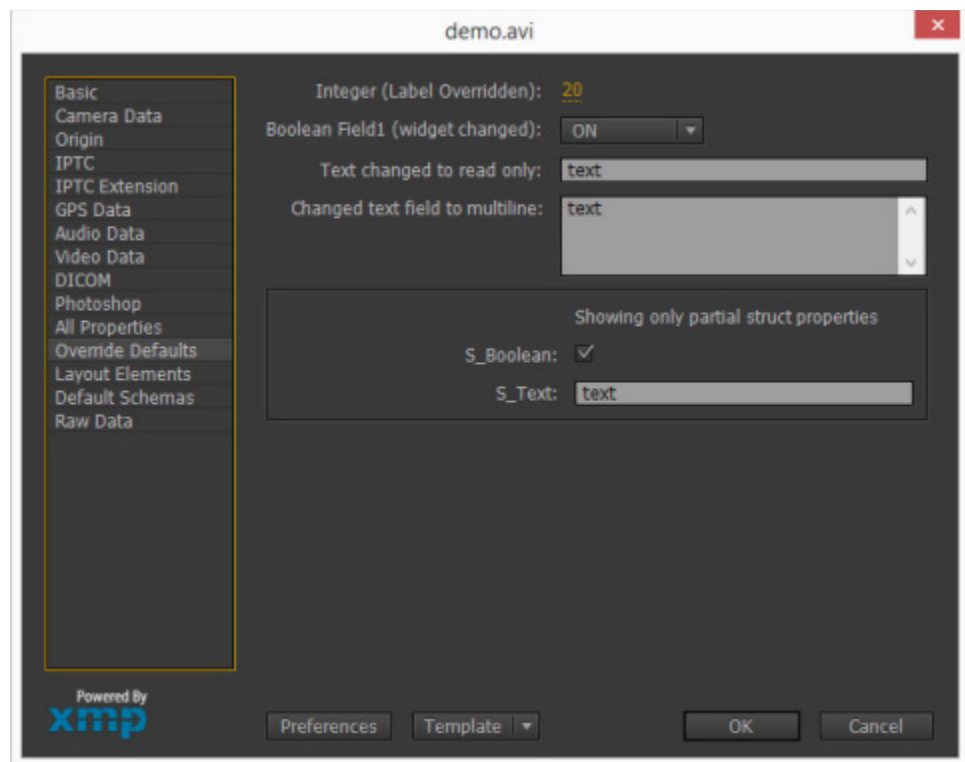
<property name="sampleSchema:Boolean" />
<property name="sampleSchema:Boolean_readOnly" />
...
<property name="sampleSchema:ExtendsOpenChoice" />
<property name="sampleSchema:ExtendsClosedChoice" />

<separator/>
<property name="sampleSchema:ComposedStruct" />
<separator/>
<property name="sampleSchema:ArrayOfStruct" />
</view>

```

Override Defaults view

This panel demonstrates the various ways in which you can customize the rendering of different types of properties.



This `view` element defines a panel that overrides these aspects of the default presentations for properties that are defined in the schema file:

- Override label and description (tool tip):

```

<property name="sampleSchema:Integer"
  label ="Integer (Label Overridden)"
  description="Overriding the tool tip" />

```

- Override default widget by changing the default checkbox widget for a boolean property to a dropdown, with appropriate choices:

```
<property name="sampleSchema:Boolean"
  label="Boolean Field1 (widget changed)"
  widget="dropdown">/
  <choice label="Unknown" value="" />
  <choice label="ON" value="True" />
  <choice label="OFF" value="False" />
</property>
```

- Render a writable property as read-only:

```
<property name="sampleSchema:ComposedStruct"
  label="Struct changed to read only"
  readOnly="true" />
```

- Make a text field multi-line:

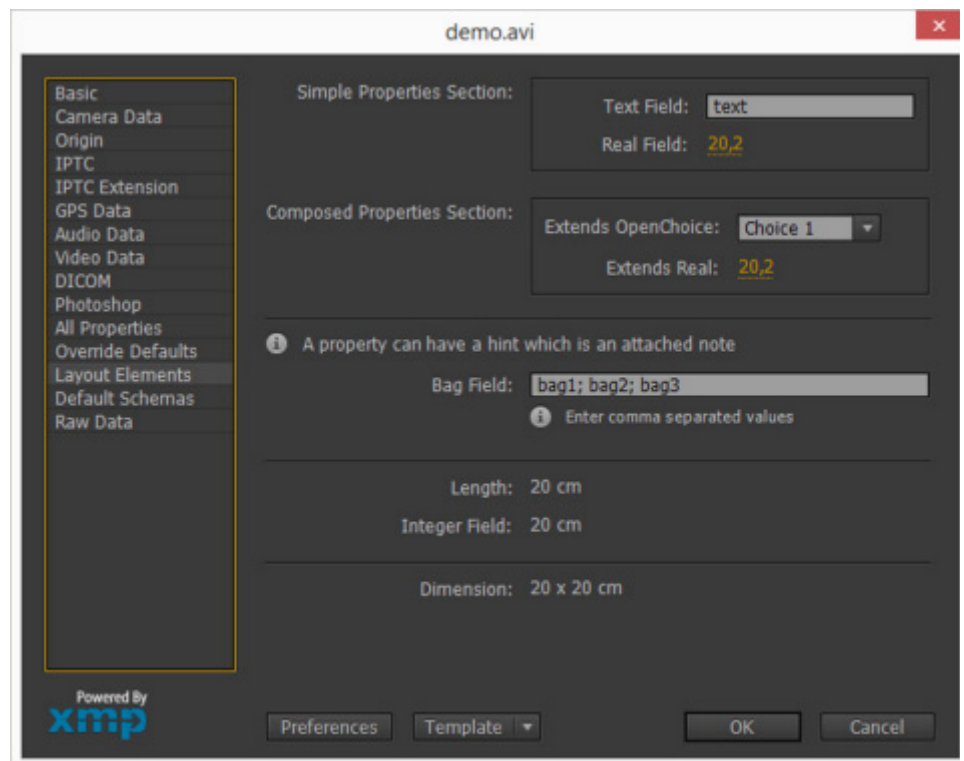
```
<property name="sampleSchema:Text"
  label="Changed text field to multiline"
  multiLines="3" />
```

- Display only some fields of a structure:

```
<property name="sampleSchema:ComposedStruct"
  label="Showing only partial struct properties">
  <property name="sampleSchema:ComposedStruct/sampleSchema:s_boolean" />
  <property name="sampleSchema:ComposedStruct/sampleSchema:s_text" />
</property>
```

Layout Elements view

This panel illustrates more layout options for visually grouping components of a panel



This `view` element defines a panel that uses sections, separators, and notes, as well as a formatter to combine various values into a text display.

► Creating sections:

```
<section label="Simple Properties Section" type="labelled">
  <property name="sampleSchema:Text" />
  <property name="sampleSchema:Real" />
</section>
```

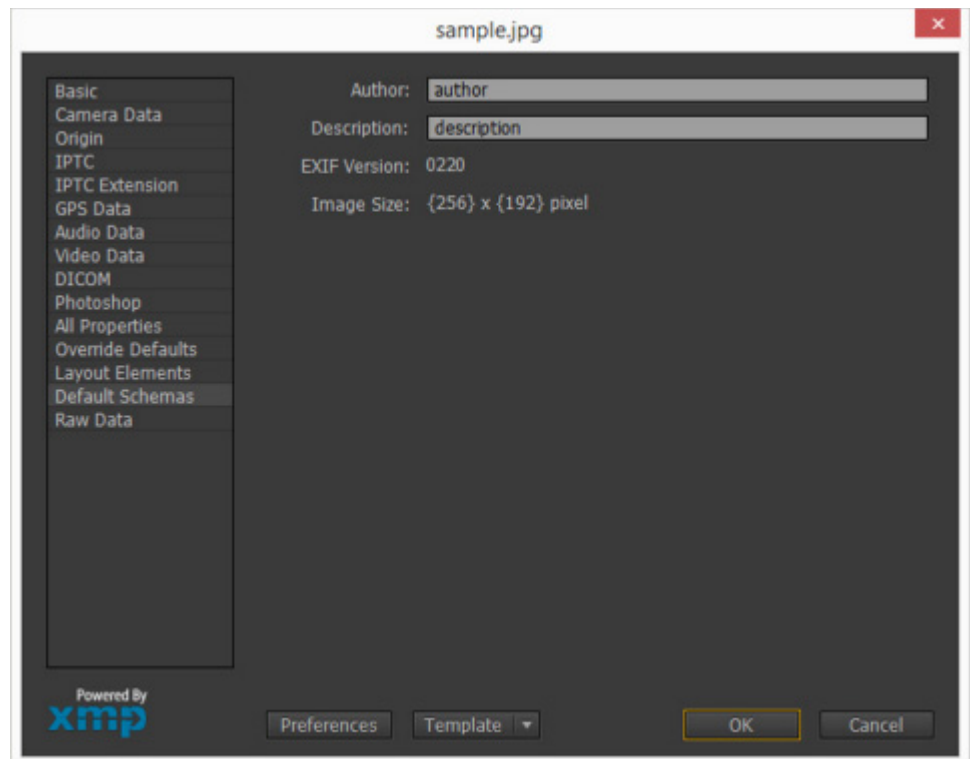
► Adding notes: you can use a separate `note` tag for a section, or a `hint` attribute in the tag for the annotated property.

► Formatting (can only be used for read-only or internal properties):

```
<formatter label="Dimension"
  description="Clubbing two properties together with formatter"
  format="@0 x @1 cm">
  <property name="sampleSchema:Integer" readOnly="true"/>
  <property name="sampleSchema:Integer" readOnly="true"/>
</formatter>
```

Default Schemas view

This panel shows how to create new views of properties defined in built-in schemas.



This `view` element defines a panel that mixes selected properties of Dublin Core and EXIF built-in schemas in a single view, and also demonstrates the use of the `<formatter>` element.

```
<view xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:exif="http://ns.adobe.com/exif/1.0/"
  name="sample.usingDefaultSchema"
  ...>
```



```
<property name="dc:creator" />
<property name="dc:description" />

<property name="exif:ExifVersion" />

<formatter label="Image Size"
  description="Shows the X and Y size of the image"
  format="{@0} x {@1} pixel">
  <property name="exif:PixelXDimension"/>
  <property name="exif:PixelYDimension"/>
</formatter>

</view>
```

3 Metadata UI Extension Formats and Schema

This chapter provides a complete reference for the Metadata UI Extension package file formats and XML schema.

About careful XML coding

XML files have strict syntax requirements. When you create or edit an extension installation file, make sure that you use correct XML syntax:

- ▶ Every attribute value must be enclosed in a single pair of single or double quotation marks. For example, `version = "1.0.0"`.
- ▶ A tag with no contents must end with `/>`. Do not include any spaces between the slash and the closing angle bracket.
- ▶ Each attribute name must be preceded by a space (or other form of white space). If you use more than one attribute in a tag, you must put a space between each attribute's value and the next attribute's name.
- ▶ XML uses the ampersand (&) as an escape character. To include an ampersand within a tag, use the code `&`. Similarly, use `<` and `>` for the `<` and `>` characters.
- ▶ It is recommended that you encode XML with UTF-8 and explicitly declare the encoding.
 - ▷ In both Mac OS and Windows, include the UTF-8 encoding declaration at the head of the XML file:
`<?xml version="1.0" encoding="UTF-8"?>`
 - ▷ In Windows, also include the BOM at the head of the XML file. The easiest way to is to open the XML file with Notepad and select File > Save As, then set Encoding as UTF-8.

File formats

These file formats are defined for the Metadata UI Extension XML schema:

- ▶ Schema file format, used for files in the `Pkg_Root/schema/` folder of a Metadata UI Extension package. For details of the XML schema used in these files, see [‘Schema file elements’ on page 29](#).
- ▶ View file format, used for files in the `Pkg_Root/view/` folder of a Metadata UI Extension package. For details of the XML schema used in these files, see [‘View file elements’ on page 34](#).
- ▶ The Adobe ZString format is used for localization files in the `Pkg_Root/loc/` folder of a Metadata UI Extension package. The filename format you use must be referenced in the manifest file for the package. See [‘Localization file format’](#) and [‘Manifest file elements’ on page 29](#).

Schema file format

An XML file in the `schema` folder of a Metadata UI Extension package describes the properties contained in an XMP schema. This XML representation can be used to define new custom properties that can then be included in any projects, so that the values can be edited using the XMP Metadata UI.

All properties are unique within the defined namespace. The schema file contains only descriptive information about the properties in the namespace. It does not contain any layout or presentation details (although there are default presentations associated with each property type). You can reference these properties in a view description file using the prefix defined for the namespace and the corresponding property name, so that they appear in custom panels in the XMP Metadata UI.

A schema file can also define custom property types that extend the primitive types defined directly by the XMP specification. Custom types can be simple, single-value types that are derived from primitive types, or you can define complex types, or *structures*, that contain multiple values of different types. The *fields* of a structure contain some combination of simple or complex types.

The outer tag for the schema definition file is [xmp_definitions](#). It contains one or more [xmp_schema](#) tags, and [xmp_type](#) tags if needed. By convention, there is one schema tag for each XMP namespace, but a file can contain more than one schema tag, so that you can define different namespaces in a single file.

This simple snippet shows the general structure of the file:

```
<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_type>
    <!-- TYPE DEFINITIONS HERE -->
  </xmp_type>
</xmp_definitions>
```

View file format

An XML file in the `view` folder of a Metadata UI Extension package describes whether and how to render schema properties as widgets in a UI panel. It contains only the presentation information for the panel and properties; it does not contain actual property definitions. A property that is referenced in a view (using the fully qualified property name) is linked to its full definition in the schema definition file

The following example snippet defines a single view that maps a single property to the default widget for its type. This assumes that there is a schema file in the same package that defines the "http://ns.adobe.com/myNamespace/" schema, which contains a property definition with the name `boolProp`. The property is referenced using the defined prefix "my". The strings are in ZString format, which allows them to be localized using supplied localization files in the same package.

```
<!-- This specific namespace is required for the top-level container-->
<views xmlns="http://ns.adobe.com/metadata/ui/1.0/">
  <view xmlns:my="http://ns.adobe.com/myNamespace/"
    name="com.example.my.view"
    description="$$$/CustomView/my_description=An Example"
    label="$$$/CustomView/my_label=My View">
    <property name='my:boolProp' />
  </view>
</views>
```

The single [views](#) top-level element can contain multiple [view](#) elements, each of which describe one panel. The [property](#) elements within a `view` tag define which properties are to be displayed in the panel, and can also override the default ways in which they are mapped to widgets. The view can also define other display details, such as [separator](#) and [note](#) elements in the panel.

Localization file format

The Metadata UI Extension SDK supports ZString localization of all display strings. If you supply display strings in ZString format, you must also supply ZString translation files for all supported locales in the `loc/` folder of the package, and specify the prefix that you use for these file names in the `manifest.xml` file that describes the package.

For example, suppose you supply these files in the package:

```
loc/  
    myapp_en_US.dict  
    myapp_de_DE.dict
```

The `manifest.xml` file's single `extension` element which specifies the name and version of the package, must also supply the filename prefix for the localization files:

```
<extension xmlns="http://ns.adobe.com/metadata/extension/1.0/"  
    name="com.mycompany.metadata.myapp"  
    version="1.0"  
    locFilePrefix="myapp">  
</extension>
```

The use of ZStrings and localization files is completely optional. If you do not use this mechanism, your package does not need to contain a `loc/` folder.

Using ZStrings for localization

The ZString localization format and usage is standard for Adobe applications, and is described in product SDK documentation. All labels and tooltips for the XMP components support localization.

Any label-text property can contain a ZString key, which is a string in this format:

```
"$$$/locKey=default value"
```

The panel looks up the localized value for the string before displaying it. It searches for a localization file for the current locale, and if it finds one, searches for the key in that file. If a localized string is found, it is displayed. 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.

Each ZString key (the string on the left of the "=") must be unique, so that strings are unambiguously identified.

Manifest file elements

The single element of the top-level `manifest.xml` file for a package describes the package. The name and SDK version information is required; the localization information is required if you supply localization files.

extension tag

The single element of the top-level `manifest.xml` file for an extension package describes that package. The version information is used in resolving conflicts between extensions.

ALLOWED SUBTAGS: None

ATTRIBUTES: `xmlns`, `name`, `version`, `locFilePrefixs`

<code>xmlns</code>	Required. The namespace of the XMP Metadata UI. Must be the value "http://ns.adobe.com/metadata/extension/1.0/"
<code>name</code>	Required. The unique name of this package. Because this must be globally unique, it is recommended that you use some version of your domain name; for example, "com.myCompany.metadata.myProduct".
<code>version</code>	Required. The version of this extension package, a number with no more than one decimal place, such a "1" or "1.1". If two packages have the same <code>name</code> value, the one with the highest version number is loaded. A package with the same name and version as one already registered is not loaded.
<code>locFilePrefix</code>	Optional. The prefix used for localization files in this package.

Schema file elements

These elements are used in XML files that define XMP schemas and their properties.

Tag	Description
<code>xmp_definitions</code>	Top-level container
<code>xmp_schema</code>	Defines a schema definition, contains property definitions.
<code>xmp_property</code>	Defines a metadata property within the schema.
<code>xmp_choice</code>	Defines the allowed choices for a choice-type property.
<code>xmp_type</code>	Defines a custom complex data type that can be used in these schemas.
<code>xmp_field</code>	Defines a field of a custom type structure.

xmp_definitions

Top-level container for a schema file, contains schema definitions, and optionally custom data types.

ALLOWED SUBTAGS: [xmp_schema](#), [xmp_type](#)

ATTRIBUTES: None

xmp_schema

Container for a schema definition.

CONTAINED IN: [xmp_definitions](#)

ALLOWED SUBTAGS: [xmp_property](#)

ATTRIBUTES: label, namespace, prefix, description

label	The localizable display name for this schema.
namespace	The full namespace URI for this schema.
prefix	The prefix used for this namespace within this tag
description	Optional. A localizable descriptive string for this schema.

xmp_property

Defines a property with the parent schema.

CONTAINED IN: [xmp_schema](#)

ALLOWED SUBTAGS: [xmp_choice](#)

ATTRIBUTES: label, name, description, category, type, element_type, min, max

label	The localizable display name for this property.
name	The unique name of the property within this schema.
description	Optional. A localizable descriptive string for this property.
category	The read-write status of the property. One of: external (editable) internal (read-only)

<code>type</code>	<p>The data type for this property, which determines the type of component used to render the item in the UI.</p> <ul style="list-style-type: none"> ► For a simple property, one of the simple data types defined by XMP: <ul style="list-style-type: none"> <code>text</code> <code>integer</code> <code>real</code> <code>boolean</code> <code>date</code> <code>closedchoice</code> <code>openchoice</code> ► For an array (with elements of the type defined by the <code>element_type</code> value), one of: <ul style="list-style-type: none"> <code>bag</code>: An unordered array <code>seq</code>: An ordered array <code>alt</code>: An array containing alternative values ► For a language alternates array: <ul style="list-style-type: none"> <code>langalt</code>
<code>element_type</code>	<p>The data type of elements of this array or choice property. Can be a custom type defined in this schema definition, or one of these primitive types:</p> <ul style="list-style-type: none"> <code>text</code> <code>integer</code> <code>real</code> <code>boolean</code> <code>date</code> <p>Required if <code>type</code> is any of the array or choice types.</p>
<code>min, max</code>	<p>The minimum and maximum threshold for properties of type <code>real</code> or <code>integer</code>. The property cannot store a value outside this inclusive range.</p>

xmp_choice

A property of type `openchoice` or `closedchoice` can contain any number of `xmp_choice` tags. Each tag specifies one choice, which appears in the drop-down list associated with the property in the UI.

CONTAINED IN: [xmp_property](#)

ALLOWED SUBTAGS: None

ATTRIBUTES: `label, raw_value`

<code>label</code>	The localizable display name for this choice.
<code>raw_value</code>	The XMP value placed in the component when the user chooses the associated item.

xmp_type

Describes a custom data type that either inherits from and modifies a primitive type, or defines a complex structure. If the type extends one of the choice types, this element must contain the `xmp_choice` elements for the allowed choices. If it defines a structure, this element must contain the `xmp_field` elements that define the contents of the structure.

CONTAINED IN: [xmp_definitions](#)

ALLOWED SUBTAGS: [xmp_field](#), [xmp_choice](#)

ATTRIBUTES: name, label, description, extends

name	The unique name of this type, used to refer to it within the schema file. This must not conflict with any type names defined by the XMP specification.
label	The localizable display name for this custom type.
description	Optional. A localizable descriptive string for this custom type.
extends	Optional. If this custom type is derived from an XMP-defined property type, that type. One of: <div><div>text</div><div>integer</div><div>real</div><div>boolean</div><div>openchoice</div><div>closedchoice</div><div>bag</div><div>seq</div><div>alt</div></div>

xmp_field

Defines a field of a custom property type structure. If the field uses one of the choice types, this element must contain the `xmp_choice` elements for the allowed choices.

CONTAINED IN: [xmp_type](#)

ALLOWED SUBTAGS: [xmp_choice](#)

ATTRIBUTES: name, label, description, type, namespace, prefix

name	The unique name of this field, used to refer to it within the schema file.
label	The localizable display name for this field.
description	Optional. A localizable descriptive string for this field, used for tooltip.

type	<p>The data type for this field, which determines the type of component used to render the item in the UI.</p> <ul style="list-style-type: none">► For a simple property, one of the simple data types defined by XMP: text integer real boolean date closedchoice openchoice► For an array with simple-type elements (of the type defined by the <code>element_type</code> value), one of: bag : An unordered array seq : An ordered array alt : An array containing alternative values► For a language alternates array: langalt
namespace	The full namespace URI for namespace to which the parent type belongs
prefix	The prefix used for this namespace.

View file elements

These elements are used in XML files that define which XMP properties should be displayed in a custom panel in the XMP Metadata UI, along with details of how property values are rendered and the appearance of the custom panel.

Tag	Description
views	Top-level container for a view file.
view	Defines a single custom panel.
property	Specifies a property to be displayed in a panel, with details of the rendering and layout.
choice	Specifies a choice to be displayed in a dropdown list.
formatter	A text formatter for read-only properties in the panel.
separator	Draws a line across the panel.
section	Defines a group of properties within the panel that is displayed in a box with a title.
note	Adds an informational message to the panel.

views

Top-level container for a view file, contains one or more view definitions, each of which describes one panel in the XMP Metadata UI.

ALLOWED SUBTAGS: [view](#)

ATTRIBUTES: xmlns

xmlns	Required. The value "http://ns.adobe.com/metadata/ui/1.0/", which identifies this file as a view layout definition.
-------	---

view

A view layout definition for one panel in the XMP Metadata UI.

CONTAINED IN: [views](#)

ALLOWED SUBTAGS: [property](#), [formatter](#), [separator](#), [section](#), [note](#)

ATTRIBUTES: label, name, prefix, description

label	The localizable display name for this view, used as the panel title.
name	The unique name of this view. Must be unique across all extension packages. It is recommend that this start with a package name; for example, "com.mycompany.area.myPanel1".

<code>xmlns:<prefix></code>	Maps a namespace to the given prefix, which you can then use to refer to properties defined in the corresponding schema. For example, <code>"xmlns:dc="http://purl.org/dc/elements/1.1/"</code> allows you to refer to Dublin Core properties. You must specify all of the namespaces whose properties are referenced in this view.
<code>description</code>	Optional. A localizable descriptive string for this panel, used as a tooltip for the panel.

property

Specifies a property to be displayed in the panel. For properties with structured types, nested property elements can specify which fields within the structure to display, and their order.

CONTAINED IN: [view](#), [property](#), [section](#), [formatter](#)

ALLOWED SUBTAGS: [property](#), [choice](#)

ATTRIBUTES: `name`, `label`, `description`, `widget`, `readOnly`, `hint`, `multiLines`, `format`

<code>name</code>	A qualified name of an XMP property to display in the panel according to this specification. Use the namespace prefix with the unique property name; for example, <code>"dc:creator"</code> .
<code>label</code>	Optional. A localizable display name for this property that overrides the label value in the schema definition.
<code>description</code>	Optional. A localizable descriptive string for this property that overrides the description value in the schema definition. Displayed in the tooltip for this property's widget.
<code>widget</code>	Optional. The name of a widget that should explicitly be used to render this item, overriding the default widget according to the property type. One of: <ul style="list-style-type: none"> ► <code>dropdown</code>: Uses a dropdown list for the property. If specified, you must also supply choice tags within this element that define each choice to be displayed in the list. The widget is closed-choice by default, unless you supply the attribute <code>openChoice=true</code>. ► <code>rating</code>: Uses the rating widget for this property; use only with a property of type <code>real</code>. When you use this widget, the user can only enter integer values 0-5, with 0 corresponding to no stars highlighted.
<code>openChoice</code>	If a dropdown widget is specified, <code>true</code> to make it open-choice. Default is <code>false</code> .
<code>size</code>	If a rating widget is specified, the size of the stars in the widget. One of <code>"large"</code> , <code>"medium"</code> , or <code>"small"</code> (the default).
<code>readOnly</code>	Optional. One of <code>"true"</code> or <code>"false"</code> . True to prevent a read-write property from being editable in the panel. (You cannot make a property editable if it is defined to be read-only in the schema.)

hint	Optional. Adds an informational message that is displayed below this property's widget.
multiLines	Optional. The number of lines to allow for a property whose type is <code>text</code> .
format	Optional. For read-only properties, a format string in which the parameter <code>@0</code> is replaced by the property value for display.

EXAMPLES: `<property name="sampleSchema:Text" label="Change text field to multi-line and prevent edit" multiLines="3" readOnly="true" />`

`<property name="sampleSchema:Real" label="Enforce usage for rating value 0-5, with large star icons" widget="rating" size="large" />`

choice

Specifies a choice to be displayed in the dropdown list. Optional for a property that has the type `openchoice` or `closedchoice`, to override the set of choices specified in the property definition. Required when the parent `property` tag specifies `widget=dropdown` for a property of another simple type.

CONTAINED IN: [property](#)

ALLOWED SUBTAGS: None

ATTRIBUTES: `label`, `value`

label	The localizable display name for this choice.
value	The XMP value placed in the component when the user chooses the associated item.

section

Defines a group of properties that can be displayed in a box with a title.

CONTAINED IN: [view](#)

ALLOWED SUBTAGS: [property](#), [formatter](#), [note](#)

ATTRIBUTES: `type`, `label`, `description`

type	Optional. How to show the section. One of: <ul style="list-style-type: none"> ▶ <code>header</code>: The section has a title, but no box. This is the default type. ▶ <code>labelled</code>: The section has a title, and is surrounded by a box. ▶ <code>simple</code>: The section is surrounded by a box but has no title.
------	--

label	Optional. The localizable title of the section. Ignored if the type is <code>simple</code> .
description	Optional. A localizable descriptive string used as the tooltip for the title. Ignored if the type is <code>simple</code> .

EXAMPLE: `<section label="Simple Properties Section" type="labelled">
 <property name="sampleSchema:Text" />
 <property name="sampleSchema:Real" />
</section>`

formatter

A text formatter for a grouped set of read-only properties. Formats the values of the nested properties into a text display, and optionally gives the widget a title.

You can supply either label/description or labelRef/descriptionRef. If neither is supplied, no title is shown.

CONTAINED IN: [view, section](#)

ALLOWED SUBTAGS: [property](#)

ATTRIBUTES: label, description, labelRef, descriptionRef, format, separator

label	Optional. The localizable title. You can supply either <code>label</code> or <code>labelRef</code> ; if neither is supplied, no title is shown.
description	Optional. A localizable descriptive string used for the tooltip of the title. You can supply either <code>description</code> or <code>descriptionRef</code> ; if neither is supplied, no tooltip is shown.
labelRef	Optional. An XMP property reference from which to take the label for the title. Supersedes local <code>label</code> value if both are supplied.
descriptionRef	Optional. An XMP property reference from which to take the description for the tooltip. Supersedes local <code>description</code> value if both are supplied.
format	Optional. The format string. In this string, parameters <code>@0</code> to <code>@9</code> are replaced by property values from the contained property tags, in the order in which they appear.
separator	Optional. A string to use between the concatenated values of the contained property tags. Ignored if a <code>format</code> string is supplied.

EXAMPLE: `<formatter label="Dimension"
 descriptionRef="sampleSchema:Integer"
 format="@0 x @1 cm">
 <property name="sampleSchema:Integer" readOnly="true"/>
 <property name="sampleSchema:Integer" readOnly="true"/>
</formatter>`

separator

Draws a line across the panel.

CONTAINED IN: [view](#)

ALLOWED SUBTAGS: None

ATTRIBUTES: None

note

A note that provides information about the panel, displayed across the full width of the panel. (Use the `hint` attribute of the [property](#) tag to annotate widgets.)

CONTAINED IN: [view](#), [section](#)

ALLOWED SUBTAGS: None

ATTRIBUTES: `icon`, `message`, `fontSize`, `align`

<code>icon</code>	Optional. Controls the icon to be displayed with the note. Currently only one icon is defined, "info", which displays the system-defined info icon.
<code>message</code>	The message to display. Automatically formatted as single-line or multi-line as needed.
<code>fontSize</code>	Optional. The font size to use, one of: <ul style="list-style-type: none">▶ <code>small</code>▶ <code>normal</code> (default)▶ <code>large</code>

EXAMPLE:

```
<note fontSize="large" icon="info"
      message="This panel contains only custom properties"/>
```