XMP Custom Panels

ADOBE SYSTEMS INCORPORATED

Corporate Headquarters
345 Park Avenue
San Jose, CA 95110-2704
(408) 536-6000
http://partners.adobe.com

September 2003

Contents

XMP Custom File Info Panels

Introduction

The custom File Info panel for XMP metadata allows you to define, create, and manage custom metadata properties using standard Adobe applications. You can do this by creating a Custom Panel Description file, as described in this document, and placing it in a common location referenced by Adobe applications that support this feature. You can also supply localization dictionary files to localize the contents of your Custom Panel Description files.

The resulting custom dialog panel is seen by users when they select the **File Info** menu option.

The File Info panel allows you to support paths to metadata properties which are not defined in default or other standard XMP schemas, but are needed for your application, company, or industry. By using XMP for that metadata, you take advantage of the potential of XMP for interchange and participation in asset management systems, while providing the ability to use standard Adobe applications to manage it.

Custom Panel Files

There are two types of files used in creating a File Info custom panel:

- Panel description files

  These files directly describe and create the custom panels. See "Custom Panel Description File Format" on page 5 for a complete description.

- Localization dictionary files

  These optional files localize the panel description file contents to other languages at run time. See "Localization Dictionary Files" on page 26 for a complete description.

Adobe applications that support this feature look for these files in the prescribed locations, and add your custom panel or panels to their **File Info** dialogs in the language of the dialog.

Custom Panel Description File Format

The panel description file is a 7-bit ASCII file in a simple form of XML, containing the user-interface definitions for a panel. See Sample Custom Panels for example of the format. One description file describes exactly one custom file panel; however, you can have as many custom panel files as you wish.

The following file creation tools are recommended:

- On Mac OS 9 use SimpleText or a free third-party text editor such as MPW Shell.

- On Mac OS X use TextEdit and make sure you save the file as type "Plain Text," not RTF.
- On Windows® use Notepad, and be sure to save the file as type "Plain Text."

Save the file with extension `.txt` or `.xmp`.

File Locations

The panel description and localization dictionary files must reside in the following locations for proper use:

Mac OS X
```
{Root Volume}/Library/Application Support/Adobe/XMP
     /Custom File Info Panels
{Home Directory}/Library/Application/Adobe/XMP
     /Custom File Info Panels
```

Windows
```
\Program Files\Common Files\Adobe\XMP
     \Custom File Info Panels
\Documents and Settings\<user>\Application Data\Adobe\XMP
     \Custom File Info Panels
```

Each time an Adobe application opens the File Info dialog, it scans all the files in these directories and appends them to the file list. If more than one of the description files has the same name, the first one found is added, and others are ignored.

Changes to the contents of the directories do not require you to reload an application; changes are automatically reflected in the dialog the next time it is opened.

Custom Panel Description File Contents

Custom Panel descriptions define *widgets*, which generally represent visual elements displayed on the screen. Custom Panel description files contain only widget definitions; they cannot contain comments.

Panel descriptions consist of nested widget definitions. Widgets that contain other widgets are called *containers*; these do not always have a visual representation. Bottom-level widget definitions are called *leaves*; these are generally UI controls, such as text boxes and pop-up menus. The nesting of widgets is called the *scoping*.

- A leaf widget has the following format:
```
widget_type (widget_variables);
```
- A container widget has the following format:
```
widget_type (widget_variables)
    {
     ...nested widgets...
    }
```

Widget types are defined to represent different kinds of containers and controls. The widget types are listed and described in "Custom Panel Widget Types" on page 7.

All widgets can have variables associated with them; a variable consists of a key-value pair (see "Widget Variable Key-Value Pairs" on page 11). The variables affect the layout of the custom panel, the behavior of the widget, and the connection between the controls and XMP properties. All of the variables are described in "Widget Variables" on page 11.

- The values of the widget variables can depend on the scope—that is, the nesting level at which they are specified. See "Scoping for Linked Variable Values" on page 12.
- Widgets contain *ZStrings*, an Adobe-defined type, to specify displayed text; see "ZString" on page 25. This allows displayed text to be translated to other languages using localization dictionaries.

Strings and ZStrings are enclosed in double quotes in the XMP standard, but must be enclosed in single quotes in the widget description language. For example, the following shows how double quotes are used in the XMP headers for the description file, but single quotes are used in the widget descriptions themselves:

```
<?xml version="1.0">
<!DOCTYPE panel SYSTEM "http://ns.adobe.com/custompanels/1.0">
<panel title="$$$/CustomPanels/Acro/PanelName=Test All Widgets"
    version="1" type="custom_panel">
    ...
    slider(name:'$$$/CustomPanels/Widgets/opt2=Slider',
           horizontal: align_fill, height : 10, default_value:22,
           min_value:0, max_value:100, num_tick_marks:6,
           xmp_path: 'SliderValue', pointing:slider_point_right);
    ...
```

Custom Panel Widget Types

Some widget types are containers, and others are leaves. Many widgets share certain variables; in addition, some types have custom variables. Both common and custom variables are described in detail in "Widget Variables" on page 11.

To have a dynamic value, a widget must be bound to an XMP property; see "Dynamic Value Variables" on page 19.

Container Widgets

Container widgets have no values or custom variables. The variables that apply to them are described in "Widget Variables" on page 11.

Widget type	Description
`group`	A simple container, with no visual representation. It exists to organize its widget children. Use nested groups to organize the alignments and layouts of the contained leaf widgets.

cluster	A container whose children are enclosed in a graphic box, with a space left for the name of the box in the upper-left corner. • Specify a **margin** of at least 15 to allow proper drawing of the graphic box. • Specify a **margin_top** of at least 25 if the cluster has a **name** value set.

Leaf Widgets

The following table summarizes the leaf widget types. The XMP value type is the expected type of an XMP property bound to such a widget. The custom variables are described in "Type-Specific Leaf Widget Variables" on page 20.

Widget type	Description	Custom variables
edit_text (dynamic-value) XMP value type: string	A text editing field control. You must provide **name**, **xmp_namespace** and **xmp_path** values for this widget to supply a dynamic value.	**password** **seven_bit_ascii** **max_chars** **restriction** **v_scroller** For details, see "edit_text variables" on page 20
edit_number (fixed-text or dynamic-value) XMP value type: string	A numeric text editing field control. Numeric values are formatted. This subclass of **edit_text** inherits all the **edit_text** custom properties and behavior. You must provide **name**, **xmp_namespace** and **xmp_path** values for this widget to supply a dynamic value.	**format** **format_addin** **min_value** **max_value** **precision** For details, see "edit_number variables" on page 21
static_text (fixed-text or dynamic-value) XMP value type: string	A single-line, non-editable text field control suitable for text display or to label another field. You must provide **name**, **xmp_namespace** and **xmp_path** values for this widget to supply a dynamic value.	**truncate** For details, see "static_text variables" on page 23

Widget type	Description	Custom variables
static_text_enum (fixed-text or dynamic-value) XMP value type: string	A multi-line non-editable text field control. This subclass of **static_text** inherits all the **static_text** behavior. You must provide **name**, **xmp_namespace** and **xmp_path** values for this widget to supply a dynamic value.	**items** For details, see "static_text_enum variables" on page 23
cat_container_edit_text XMP value type: none	An editable text field bound to an XMP array of type bag or sequence. Each item in the array of XMP data is shown, with items separated by semicolons or commas (if **preserve_commas** is **false**). When the user clicks **OK** to close the dialog, the values are stored in sequence in the XMP array.	**preserve_commas** For details, see "cat_container_edit_text variables" on page 24
popup (fixed-text or dynamic-value) XMP value type: integer	A pop-up menu control. The value of the selected popup menu item is stored in XMP when the File Info dialog is closed. Setting the **font** parameter for a popup does not affect the font of the menu items, only the font of the popup menu title. You must provide **name**, **xmp_namespace** and **xmp_path** values for this widget to supply a dynamic value.	**items** **no_checks** For details, see "popup variables" on page 20

Widget type	Description	Custom variables
mru_popup (dynamic-value) XMP value type: none	Most-recently-used popup menu, bound to a specific XMP property. You must provide **name**, **xmp_namespace** and **xmp_path** values for this widget to supply the dynamic values. If this widget is in a custom panel, when the user clicks **OK** to close the dialog, the application saves the 10 most recently used values for the bound XMP property. These values are then available in this popup menu next time the dialog is opened. Choosing an item from the MRU menu changes the bound property's value; any other widgets bound that property are also updated. **NOTE:** On Mac OS, set the property **no_checks** to **false** to ensure the proper appearance.	**container_type** For details, see "mru_popup variables" on page 24
check_box (fixed-text or dynamic-value) XMP value type: boolean	A check box control. The value of a dynamic-value check box is **true** or **false**, reflecting whether it is checked. The label (**name** variable) for the check box is a localizable ZString. You must provide **name**, **xmp_namespace** and **xmp_path** values for this widget to supply a dynamic value.	**initial_value** For details, see "check_box variables" on page 23
slider (dynamic-value) XMP value type: integer	A slider control. You must provide **name**, **xmp_namespace** and **xmp_path** values for this widget to supply a dynamic value.	**min_value** **max_value** **num_tick_marks** **pointing** For details, see "slider variables" on page 23

Widget type	Description	Custom variables
`separator` XMP value type: none	A simple horizontal rule. Set the **`horizontal`** variable to **`align_fill`** to make the separator adjust itself to the width of your custom panel.	none
`picture` XMP value type: none	An icon or image.	none

Widget Variables

WIdget properties are defined by special variables that control various aspects of the appearance and layout of the panel you are creating. Some variables can be used for all widgets, some apply only to container or leaf widgets, and some are specific to a particular type of widget.

Widget Variable Key-Value Pairs

Widget variables have two parts: the *key* and the *value*:

- The key is the identifier or name of the variable. The key is composed of 7-bit ASCII alphanumeric text, and cannot have any whitespace in it.
- The value has one of the following value types:

integer	A signed 32-bit number, with no delimiting characters. Negative values are not allowed.
string	A 7-bit ASCII string, enclosed in single quotation marks.
zstring	An Adobe-defined ZString, enclosed in single quotation marks. See "ZString" on page 25 for details of the format.
enumeration	A predefined set of constants, or named values, with no delimiting characters. Enumerated values are listed in the variable descriptions below. You cannot define custom enumerations or customize existing enumerated variable values.
boolean	One of the predefined values **`true`** or **`false`**.

another variable	The key of another variable. The value of this variable is linked to that of the variable specified as the value. The variable you are linking to must be previously defined in a containing scope. See Scoping for Linked Variable Values below. The key name of the linked variable is not delimited by any special characters. Note that a variable name used as a value looks like an enumerated value—be careful that you do not skip over a predefined enumeration by declaring a variable whose key is identical to the possible enumeration value.

Scoping for Linked Variable Values

Variables whose values are linked to other variables change value based on the scope they are in. This can be a bit confusing, so here is an example:

```
outer_scope(myvar: 42, name: '$$$/Demo/Greeting=Hello!')
    {
    widget1(width: myvar);
    inner_scope(myvar: 21)
        {
        widget2(width: myvar);
        widget3(myvar: 11, width: myvar);
        widget4(width: 6);
        }
    widget5(width: myvar);
    }
```

NOTE: These are not valid widget types, but are used only for illustration.

In this example, the **outer_scope** widget has two variable, **myvar** and **name**. The initial value of **myvar** is 42.

- In the first leaf widget, **widget1**, the value of **width** is assigned the value of **myvar**, which at this point is 42, so the first widget is 42 pixels wide:

The outer container contains a nested container, **inner_scope**, which redefines the variable **myvar**, setting it to the value 21.

NOTE: The **myvar** defined by **inner_scope** is a *new* variable with the same name; it does no overwrite the **myvar** defined in **outer_scope**.

- The second leaf widget, **widget2**, is within the scope of **inner_scope**. It also assigns the value of **width** to the value of **myvar**. Within this scope, the value of **myvar** is 21, so the second widget is 21 pixels wide.
- The third leaf widget, **widget3**, defines its own **myvar**, gives it a value (11), and assigns **width** to **myvar**. The scope of this definition is the widget itself, so the third widget is 11 pixels wide. Because this is not a container widget, this scope affects only this widget, and the new **myvar**, with a value of 11, does not propogate any further.

- The fourth leaf widget, **widget4**, sets the width directly, rather than linking it to **myvar**. The fourth widget is 6 pixels wide.

At this point, **inner_scope** closes; the **myvar** defined in the **inner_scope** can no longer be referenced.

- The fifth widget, **widget5**, also assigns the value of **width** to the value of **myvar**. This widget is defined outside the scope of **inner_scope**, but is still within **outer_scope**. The **myvar** defined in **outer_scope** still has the value 42, so the fifth widget is 42 pixels wide.

The resulting widgets have the following width values, all from their different sources:

widget1: 42 pixels wide
widget2: 21 pixels wide
widget3: 11 pixels wide
widget4: 6 pixels wide
widget5: 42 pixels wide

General Widget Variables

These variables can apply to either container or leaf widgets. Some apply only to text widgets.

Variable	Description
height	Type: integer (default 0) The minimum height in pixels of the container or widget. The actual height depends on the **horizontal** and **child_horizontal** values as well. A container is always large enough to accomodate its contained widgets.
width	Type: integer (default 0) The minimum width in pixels of the container or widget. The actual width depends on the **vertical** and **child_vertical** values as well. A container is always large enough to accomodate its contained widgets.

horizontal	Type: enumeration
	`align_left (default)`
	`align_center`
	`align_right`
	`align_fill`
	The horizontal alignment of a container within the panel, or of a widget within its container, restricted to the area the container has designated for it. The **align_fill** value adjusts a widget's width to fill the allowed area.
	This value, if set, overrides the container's **child_horizontal** value.
	See also Widget Alignment Notes below.

Variable	Description

vertical	Type: enumeration
	`align_top`
	`align_center (default)`
	`align_bottom`
	`align_fill`
	The vertical alignment of a container within the panel, or of a widget within its container, restricted to the area the container has designated for it. The **align_fill** value adjusts a widget's height to fill the allowed area.
	This value, if set, overrides the container's **child_vertical** value.
	See also Widget Alignment Notes below.

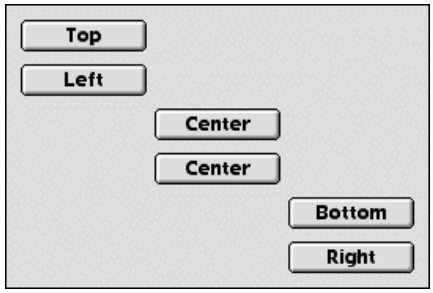
name	Type: ZString For fixed-text widgets such as **static_text** or **cluster**, this value labels the widget. **NOTE:** Applies only to fixed-text widgets. Do not use this variable to change the value or text of the widget at run time. Use **xmp_namespace** and **xmp_path** to provide dynamic values.

font	Type: enumeration
	`font_small`
	`font_small_right`
	`font_big (default)`
	`font_big_right`
	Applies only to text widgets. Specifies the style (size, weight, and right-left alignment) of the displayed text.
	See also Text Alignment Notes below.

locked	Type: boolean or the value `read_only_aware` Optional. Default is **false**. When **true**, the widget is dimmed to indicate that it is read-only. For an editable text control, it is not editable, but the user can still select and copy its contents. When **read_only_aware**, the widget is dimmed only when the application's **ReadOnly** flag is set.

Widget Alignment Notes

- A panel with that uses the various horizontal and vertical alignments might look like this:



- The **`align_fill`** value for **`horizontal`** or **`vertical`** adjusts the widget's width or height to fill the allowed area. For example, if a widget is 100 pixels wide, and its container is 150 pixels wide (and uses "**`placement: place_column`**"), setting **`horizontal: align_fill`** makes the child widget's width grow to 150 pixels.

Text Alignment Notes

- To maintain colon alignment across multiple labels of text, use font alignment, (**`font:font_small_right`** or **`font:font_small_bold_right`**), rather than setting **`horizontal:align_right`** for the widget.
- If colon alignment is not desired, try setting **`label`** to **`false`**.
- To center text, you must set **`horizontal:align_center`**, as there is no centering value for **`font`**.

Container Widget Variables

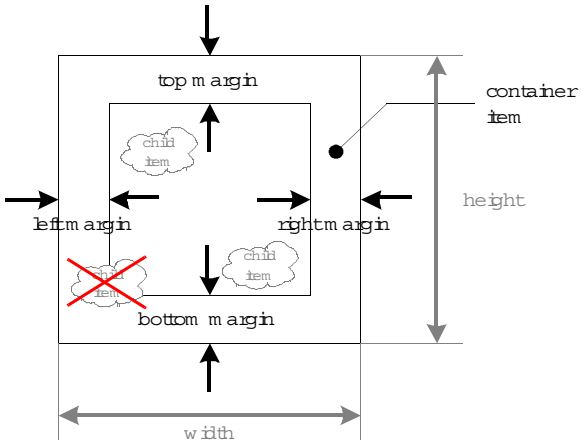
These variables apply only to container widgets.

Variable	Description

`child_horizontal`	Type: enumeration
	`align_left (default)`
	`align_center`
	`align_right`
	`align_fill`
	The horizontal alignment of all children within the container that do not have an explicit **`horizontal`** value set.
	The **`align_fill`** value adjusts the horizontal space between children so that they fill the allowed area.

Variable	Description

child_vertical	Type: enumeration
	`align_top`
	`align_center (default)`
	`align_bottom`
	`align_fill`
	The vertical alignment of all children within the container that do not have an explicit **vertical** value set.
	The **align_fill** value adjusts the vertical space between children so that they fill the allowed area.

margin	Type: integer (default 0) Margins are the padding associated with a given container. A container's children fit within the bounds of the margin padding and never encroach upon the margin:  The **margin** value applies to all the margins for a given container (top, left, right and bottom). However, if **margin_width** is present it overrides **margin** for the left and right margin values. Further, if **margin_left**, **margin_right**, **margin_top**, **margin_bottom** is defined, it overrides both **margin** and **margin_width** in the given direction.

Variable	Description

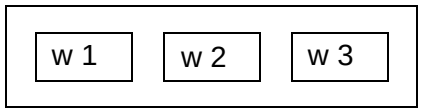
placement	Type: enumeration
	`place_column (default)`
	`place_row`
	Controls how the children of a container are laid out with respect to one another.
	• **place_row** places the children of a container along a single row.
	• **place_column** places the children in a column.
	The order of the children on placement is first-to-last in the definition, left-to-right for rows or top-to-bottom for columns.
	See also Container Size and Placement Notes below.

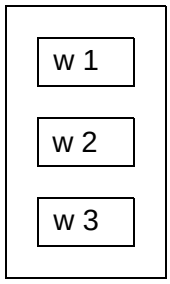
reverse	Type: The value `rtl_aware` Optional. If supplied, items are automatically realigned for left-to-right languages; that is the positions are reversed horizontally.
spacing	Type: integer (default 0) or enumeration `gSpace` (10 on Mac OS, 9 on Windows) `gMediumSpace` (10 on Mac OS, 8 on Windows) `gLargeSpace` (15 on Mac OS, 15 on Windows) The distance in pixels between children of this container. This does not affect the distance from the top of the container to the first child, nor the distance from the last child to the bottom of the container—those are set by the **margin** values.

Container Size and Placement Notes

- The size of a container is not explicitly set in the Custom File Description file, but is calculated based on the margins and the widths, heights, and spacing of the children of the container. You can set the **width** or **height** of a container to the minimum dimension for that container, but it is made larger if needed.

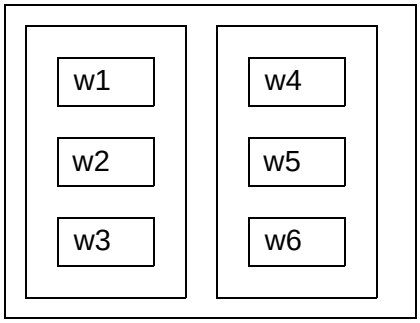
- The **`placement`** value lays out the children of a container as a row or column. For example:





To make multiple columns of widgets line up in a row with each other, you must nest the containers. For example:

```
group(placement: place_row)
 {
 group(placement: place_column)
   {
   ...column of widgets 1-3 ...
   }
 group(placement: place_column)
   {
   ...column of widgets 4-6...
   }
 }
```



Dynamic-Value Leaf Widget Variables

These variables apply only to leaf widgets whose text alignment or values are determined at run time.

Dynamic Text Variables

Variable	Description
`label`	Type: boolean When **`true`**, this text widget is considered a label of another widget, and is included in calculating colon alignment for labels. It is set to **`true`** by default for **`static_text`** widgets. When **`false`**, the widget is not included in colon-alignment calculations.

Dynamic Value Variables

For a widget's value to change at run time, it must get the value from the XML Packet for which this File Info dialog is being loaded. You specify this using both **`xmp_namespace`** and **`xmp_path`** variables.

The **`xmp_namespace`** variable is optional, and defaults to `xap_ns_xap`. For more information on the XMP namespaces, see the *XMP—Extensible Metadata Framework* document in the XMP SDK.

Variable	Description

`xmp_namespace`	Type: enumeration or string
	`xap_ns_xap (default)`
	`xap_ns_xap_g`
	`xap_ns_xap_g_img`
	`xap_ns_xap_dyn`
	`xap_ns_xap_dyn_a`
	`xap_ns_xap_dyn_v`
	`xap_ns_xap_t`
	`xap_ns_xap_t_pg`
	`xap_ns_xap_rights`
	`xap_ns_xap_mm`
	`xap_ns_xap_s`
	`xap_ns_xap_bj`
	`xap_ns_pdf`
	`xap_ns_dc`
	`photoshop (string)`
	`xap_ns_rdf`
	`xap_ns_user`
	`xap_ns_meta`
	`xap_ns_aw`
	`xap_ns_st_dimensions`
	`xap_ns_st_resolution`
	`xap_ns_st_track_desc`
	`xap_ns_st_font`
	`xap_ns_st_resource_ref`
	`xap_ns_st_version`
	`xap_ns_st_evebt`
	`xap_ns_st_file_disposition`
	`xap_ns_st_job`
	`xap_ns_st_right`
	The namespace containing the XMP property whose value the widget displays.

Variable	Description
xmp_path	Type: string The hard-coded path of the XMP property whose value the widget displays. You must convert all single-quote characters (**'**) in the **xmp_path** value to hat characters (^) . The File Info dialog converts them back when it retrieves the property value.

Type-Specific Leaf Widget Variables

These variables are defined only for widgets of specific types.

popup variables

items	Type: ZString Required. A semicolon-delimited list of the items to appear in this popup menu. The first item in the list is selected by default. Each item in the list is a name followed by its keyphrase in curly brackets. This keyphrase must be a 7-bit ASCII alphanumeric value, and must not be localized. The keyphrase is the value stored in the XMP data if the menu item is selected. All names and keyphrases in the list must be unique. You can add a separator by using a dash (**-**) as a menu item. The separator does not have a keyphrase. For example: `$$$/Sample/Menu/Items=First{keyOne};-;Second{keyTwo}` A menu can contain any number of items and separators.
no_checks	Type: boolean Optional. Default is **false**. This option is available only on Mac OS 9 or X. When **true**, removes the checkmark column. When **false**, the column is present and a checkmark is drawn by the currently selected menu item.

edit_text variables

max_chars	Type: integer Optional. Default is the maximum number of characters allowed for a text widget by the operating system. The maximum number of characters to be entered in the text control. If the user exceeds the maximum, the File Info dialog displays an alert message.

password	Type: boolean Optional. Default is **false**. When **true**, the text control is a password field. Characters entered by the user are displayed as bullets (•). The cleartext is stored in the XMP property.

restriction	Type: enumeration
	`optional`
	`recommended`
	`required`
	`not_restricted`
	Optional. Default is **not_restricted**.
	• When the value is **optional** or **not_restricted**, the dialog does not check the control's value before closing.
	• When the value is **restricted** or **recommended**, the user must enter information in the text control before closing the dialog. If the user clicks **OK** to close the dialog and no value is present, the dialog sends an error to the application, which can handle the error as desired.
	If you restrict an edit text field, you should prefix both the label of the field and the name of the panel with an asterisk (*). This convention is not is not enforced, but is highly recommended.

seven_bit_ascii	Type: boolean Optional. Default is **false**. When **true**, the text control value can have only 7-bit ASCII characters.
v_scroller	Type: boolean Optional. Default is **false**. When **true**, displays a vertical scroll bar when the height of the text becomes greater than the height of the widget.

edit_number variables

format	Type: enumeration
	`whole`
	`decimal`
	`fraction`
	`ration`
	Optional. Default is **whole**.
	The numeric format in which to display the value.
	Regardless of the display format, the value is stored in the original XMP data format.

format_addin	Type: ZString
	Optional. Default is no prefix or suffix.
	A prefix or suffix to display with the value. For example, the following values specify a suffix of "s" and a prefix of "f":
	`format_addin: '$$$/format/tag=^0s'`
	`format_addin: '$$$/format/tag=f^0'`

max_value	Type: numeric string Optional. Default is no maximum. A maximum value for input. This value must be specified as a string, not an integer. For example: `max_value: '10'`
min_value	Type: numeric string Optional. Default is no minimum. A minimum value for input. This value must be specified as a string, not an integer. For example: `min_value: '3'`
precision	Type: integer Optional. Default is 1. The number of points of precision for decimal numbers. This applies to both the **decimal** and **fraction** formats, as fractional values are stored as decimal. You can specify a precision without specifying minimum or maximum values. A precision of 0 is the same as using the **whole** number format. If **min_value**, **max_value**, and **precision** are specified using different precisions, the largest precision is applied. For example, when **min_value = 0.01**, **max_value = 3.0**, **precision=1**, and the user enters a value of 2, the field displays 2.00. The value is also stored with the largest precision, if any change is made. If the user does not modify the value, the original XMP value is not changed, regardless of how it is displayed.

Example

```
edit_number (format: decimal, precision: 1,
    format_addin: '$$$/format/tag=^0s',
    min_value: '0.1', max_value: '250.0');
```

- If the user enters 2, it is displayed as **2.0s**.
- If the user enters 2.25s, it is displayed as **2.3s**.
- If the user enters -1, the following error message is shown and the value is displayed as **0.1s**:

  "A number between 0.1 and 250.0 is required. Closest value inserted."

static_text variables

truncate	Type: boolean Optional. Default is **false**. When true, the text is truncated when it is too long to fit into the widget's width. This can occur with either a static string value or with a dynamic value that is read from an XMP property.

static_text_enum variables

items	Type: ZString
	Required.
	A semicolon-delimited list of the items to appear in this static text control.
	Each item is a name followed by its key phrase in curly brackets. The key phrase must be a 7-bit ASCII alphanumeric value, and must not be localized. This is the value stored in the XMP data. The key phrase value associated with each item must be unique.
	For example:
	`static_text_enum`
	`(items: '$$$/Sample/Items=First{keyOne};Second{keyTwo}',`
	`xmp_namespace: photoshop, xmp_path: 'Sample');`

check_box variables

initial_value	Type: boolean Optional. Default is **false**. When **true**, the check box control is checked when first displayed.

slider variables

min_value	Type: integer Required. The minimum value of the slider control.
num_tick_marks	Type: integer Optional. Default is 0. This option is available only on Mac OS 9 or X, for a slider that points to tick marks. Specifies the number of tick marks to display.

pointing	Type: enumeration
	`slider_point_none`
	`slider_point_up`
	`slider_point_down`
	Optional. Default is **slider_point_none**.
	This option is available only on Mac OS 9 or X, for a slider that points to tick marks. Specifies the direction of the pointer.

mru_popup variables

container_type	Type: enumeration
	`single_value`
	`alt_struct`
	`bag_struct`
	`seq_struct`
	Required.
	The type of the XMP property bound to this menu.

mru_append	Type: boolean Optional. Default is **false**. When **true**, appends the selected value to the XMP property. When **false**, replaces the XMP property with the selected value.

cat_container_edit_text variables

preserve_commas	Type: boolean Optional. Default is **true**. When **true**, preserves commas in delimited text items. When **false**, treats commas a delimiting characters.

ZString

ZStrings are an Adobe convention for defining localization strings, and are used by File Info to assist in the localization process. All strings that are displayed to the user of a custom panel must be formatted as ZStrings.

NOTE: Even if you are not interested in localizing your custom panel definition, you must provide ZStrings where they are required.

The ZString Format

The format of a ZString is:

```
$$$/context_path/property=value
```

$$$	The ZString marker is always required to identify a ZString and distinguish it from any other 8-bit ASCII string.
contextpath	The context path is a series of 7-bit ASCII character strings separated by the slash (/) character. You can use any strings you wish, except that no white space is allowed. The context (rather than the property name and value) distinguishes one ZString from another. For example, **$$$/a/b/OK=OK** is distinct from **$$$/x/y/OK=OK**, although both represent the text of an **OK** button. The context path also shows the context or usage of the ZString in a human-readable format. It is important to use a clear and descriptive context path to identify a property, as comments are not allowed in custom panel files. When localization is performed, the context path helps the localizers know the context of the value (that is, the string) which must be translated. It is very important that the context and path of each ZString be unique, even among different custom panels. If the context and path are not unique, you might end up with the wrong translation.

property=value	The last string in the context path is the property name, and is separated from the value by an equal sign (=). The string following the separator (=) is the UI string to use for this ZString. Everything up to the value is stripped off during the localization process. The value is the string displayed to the user, and this is the string that is translated according using any localization dictionary files that you provide. See "Localization Dictionary Files" on page 26. The value can contain special characters indicated by escape sequences, as shown in ZString Escape Sequences. It cannot contain a # character, which is always interpretted as an escape character. The value can contain an ampersand (&) character to indicate an accelerator key on Windows. Mac OS and Unix systems ignore this character.

ZString Examples

```
$$$/Adobe/Photoshop/Dialogs/XYZDialog/Buttons/OK=OK
$$$/Adobe/ImageReady/Dialogs/ABCDialog/Buttons/OK=OK
```

Notice that the context provides much more usage information than a simple property name and value:

```
$$$/OK=OK
```

Localization Dictionary Files

Localization is the process of translating and otherwise manipulating an interface so that it looks as if it had been originally designed for a particular language. The File Info dialog for which you are writing your custom panel gives you the ability to localize the strings in your panel. The language of the dialog (and panel) is chosen by the application displaying the dialog.

To localize you custom panel, you must provide a localization dictionary file for each language, in the same directory as your custom panel description file. The File Info dialog looks for all localization files appropriate to the application language. It uses the set of localization dictionaries to find translations for static ZString values in the panel. It performs the ZString translation when it loads each custom panel.

NOTE: Widget values set by means of XMP metadata retrieval cannot be localized. Even if the value is a valid ZString, it is not localized; you will see the raw ZString for the value. To localize XMP metadata, use **`alt-by-lang`** containers in your XML; see the XMP SDK documentation for more information.

If File Info finds any localization dictionary files in the directory that contains the custom panel definition, it loads the dictionary whose name ends with the *locale-string* of the language it is looking for, followed by `.dat`.

For example, for your custom panel deliverables you might distribute the following files:

- **`MyCustomPanelFile.xmp`** (the custom panel definition file)
- **`MyZStringDict_en-us.dat`** (United States English localization dictionary file)
- **`MyZStringDict_fr-fr.dat`** (French localization dictionary file)
- **`MyZStringDict_en-uk.dat`** (United Kingdom English localization dictionary file)
- **`MyZStringDict_ja-jp.dat`** (Japanese localization dictionary file)

These localization dictionary files have no internal correlation to the related custom panel definition, and the part of the file name before the language code can be anything you choose. You can provide any number of dictionaries for each language.

The File Info dialog loads only the dictionary file or files that correspond to the language of the application. For American English, for example, it loads any dictionary ending in "**`en-us.dat`**"; for French, any one ending in "**`fr-fr.dat`**", and so on.

If there are no localization dictionary files for a custom panel, or if none is found to match the application language, the File Info dialog displays the UI string found in the original ZString.

Localization Dictionary File Format

A localization dictionary file is a collection of ZStrings whose final string values are in the destination language (see "ZString" on page 25). The file must be a UTF-16 (double-byte) file with a UTF-16 byte-order marker (BOM) as the first character.

The only things allowed on a line (after the first character) are ZStrings; no newline characters or comments are allowed. The ZStings in this file must all be enclosed with double quotes.

The following text editor are recommended for creating these files:

- On Mac OS 9 you must use a third-party application which supports Unicode.
- On Mac OS X simply use TextEdit and make sure you save the file type as "Unicode" (versus UTF-8 or RTF).
- On Windows use Notepad, and be sure to save the file as type "Unicode."

NOTE: TextEdit and Notepad both have menu items that say Unicode and UTF-8 , rather than UTF-16 and UTF-8. You must use UTF-16, which corresponds to the Unicode menu item.

The first two bytes of the file must be a byte order marker (BOM):

- **`0xFEFF`**: MSB-first, or big-endian byte order
- **`0xFFFE`**: LSB-first, or little-endian byte order

The rest of the file is UTF-16 formatted text with the byte order in accordance to the byte order marker.

NOTE: For programs like TextEdit on the Macintosh and Notepad on Windows, you do not have to enter the BOM (byte order marker) manually; saving the file as a UTF-16 Unicode file format implicitly adds it. The BOM is discussed here to assist those who do not have a Unicode-aware text editing utility available to them.

Matching ZStrings for Localization

The File Info dialog searches for a ZString translation in all localization dictionaries that have been loaded for the application's language, not just in the dictionary for your custom panel. More than one localization dictionary can contain a particular ZString, or a dictionary can contain the same ZString more than once. If the File Info dialog finds more than one translation for a ZString in any combination of dictionaries, the last one found is used for the displayed value.

ZStrings in your panel can have the same property name and value, as long as they are distinguished by the context. However, another panel might use the same property names, and even define the same contexts. It is particularly important to distinguish contexts sufficiently, as the same property name might be used by another custom panel, and found in another localization dictionary. If the context is not unique, there is no guarantee that the dictionary associated with your custom panel is the last loaded, and that your translation is therefore displayed.

Troubleshooting a Custom Panel

If your custom panel does not appear or does not behave as expected, here are a few issues to consider:

- Check that the localization and custom panel description files are in the proper directory locations.
- Check the file formats. Make sure the panel description file is 7-bit ASCII (single-byte) and the ZString dictionary files are double-byte with a Unicode BOM as the first two bytes of the file.
- Check whether two or more widgets are editing the same XMP property.
- Check the **`xmp_path`** and **`xmp_namespace`** property values. Try removing numbers and other non-alphanumeric characters from your namespaces and paths to get the values to be retained. Some characters invalidate these paths.
- Check the syntax of the custom panel description. No automatic syntax checking is provided.

Additional Documentation

For more information about the XMP format and syntax, see the following documents, which are available as part of the XMP SDK:

- *XMP Overview*: an overview of how XMP works and how to use it.
- *XMP—Extensible Metadata Framework*: a complete reference for the XMP specification.

Sample Custom Panels

Overview of Samples

These sample Custom Panel description files illustrate how to achieve various effects.

- Example 1: All Widgets
- Example 2: Document Description Metadata

Check the SDK installation for these and other samples.

Sample Custom Panel Description Files

Example 1: All Widgets

This example contains one of each kind of widget, illustrating their variable assignments. The code can be found in the file `CustomPanel_allWidgets.txt`.

```
<?xml version="1.0">
<!DOCTYPE panel SYSTEM "http://ns.adobe.com/custompanels/1.0">
<panel title="$$$/CustomPanels/Acro/PanelName=Test All Widgets"
     version="1" type="custom_panel">
group(placement: place_column, spacing: gLargeSpace,
      horizontal: align_fill, vertical: align_top)
{
     cluster(name:
            '$$$/CustomPanels/Widgets/widgetName=check box, slider, progress bar',
           placement: place_column, spacing: gSpace, margin_height:10,
           horizontal: align_center, vertical: align_top, width : 500, height : 100,
           child_vertical: align_top)
     {
        check_box(name:'$$$/CustomPanels/Widgets/opt1=Check box',
                  initial_value:true, margin_width : 10);
        slider(name:'$$$/CustomPanels/Widgets/opt2=Slider',
               horizontal: align_fill, height : 10, default_value:22, min_value:0,
               max_value:100, num_tick_marks:6, xmp_path: 'SliderValue',
               pointing:slider_point_right);
     }
     cluster(name: '$$$/CustomPanels/Widgets/widgetName=static text,
            edit text, cat_container text',placement: place_column, spacing: gSpace,
            margin_height:10, horizontal: align_center, width : 400, height : 100,
            child_vertical: align_bottom)
     {
```

```
        static_text(name: '$$$/CustomPanels/Widgets/static_text=Static text:',
                    font: font_small_bold_right, vertical: align_top);
        edit_text(horizontal: align_fill, font: font_small, xmp_path: 'Title',
                  vertical: align_top);
        cat_container_edit_text(horizontal: align_fill, height: 54,
                                xmp_path: 'Keywords', v_scroller: true);
    }

    cluster (name:
     '$$$/CustomPanels/Widgets/widgetName=popup,most-recently-used popup,separator',
        placement: place_column, spacing: gSpace, horizontal: align_center,
        margin_height:10, width : 300, height : 100, child_vertical: align_center)
    {
        mru_popup(xmp_path: 'Authors', container_type: seq_struct, no_check: true,
                  vertical: align_top, mru_append: true);
        separator(horizontal: align_fill);
        popup(items: '$$$/CustomPanels/Widgets/Language/PopupItems= Popup 1{1};
                      Popup 2{2}; Popup 3{3}',
              xmp_path: 'foo');
    }
}
</panel>
```

Example 2: Document Description Metadata

This example defines a panel that shows the pre-defined Description namespace. The panel contains a set of controls bound to all of the document metadata properties in that namespace. The code can be found in the file `Description.txt`.

```
<?xml version="1.0">
<!DOCTYPE panel SYSTEM "http://ns.adobe.com/custompanels/1.0">
<panel title="$$$/AWS/FileInfoLib/Panels/Description/PanelName=Description"
    version="1" type="custom_panel">

group(placement: place_column, spacing: gSpace, horizontal: align_fill,
        vertical: align_top)
{
    group(placement: place_row, spacing: gSpace, horizontal: align_fill,
          reverse: rtl_aware)
    {
        static_text(name:
                  '$$$/AWS/FileInfoLib/Panels/Description/Title=&Document Title:',
                  vertical: align_center, font: font_big_right);
        edit_text(horizontal: align_fill, xmp_path: 'Title',
                  container_type: alt_struct);
        mru_popup(xmp_path: 'Title', container_type: alt_struct,
                  no_check: true, vertical: align_top);
    }
    group(placement: place_row, spacing: gSpace, horizontal: align_fill,
          reverse: rtl_aware)
```

```
{
   static_text(name: '$$$/AWS/FileInfoLib/Panels/Description/Author=&Author:',
                vertical: align_center, font: font_big_right);
   cat_container_edit_text(horizontal: align_fill, xmp_path: 'Authors',
                           container_type: seq_struct, preserve_commas: true);
   mru_popup(xmp_path: 'Authors', container_type: seq_struct,
             no_check: true, vertical: align_top, mru_append: true);
}
group(placement: place_row, spacing: gSpace, horizontal: align_fill,
      reverse: rtl_aware)
{
   static_text(name:
             '$$$/AWS/FileInfoLib/Panels/Description/Description=De&scription:',
             vertical: align_top, font: font_big_right);
   edit_text(horizontal: align_fill, height: 54,
             xmp_path: 'Description', container_type: alt_struct,
             v_scroller: true);
   mru_popup(xmp_path: 'Description', container_type: alt_struct,
             no_check: true, vertical: align_top);
}
group(placement: place_row, spacing: gSpace, horizontal: align_fill,
      reverse: rtl_aware)
{
   static_text(name:
          '$$$/AWS/FileInfoLib/Panels/Description/DescriptionWriter=
                D&escription Writer:',
          vertical: align_center, font: font_big_right);
   edit_text(horizontal: align_fill, xmp_namespace: photoshop,
             xmp_path: 'CaptionWriter');
   mru_popup(xmp_namespace: photoshop, xmp_path: 'CaptionWriter',
             no_check: true, vertical: align_top);
}
group(placement: place_row, spacing: gSpace, horizontal: align_fill,
      reverse: rtl_aware)
{
   static_text(name:
             '$$$/AWS/FileInfoLib/Panels/Description/Keywords=Ke&ywords:',
             vertical: align_top, vertical: align_top, font: font_big_right);
   group(placement: place_column, spacing: gSpace, horizontal: align_fill)
   {
      cat_container_edit_text(horizontal: align_fill, height: 54,
                              xmp_path: 'Keywords', container_type: bag_struct,
                              v_scroller: true);
      group(placement: place_row, spacing: gSpace, horizontal: align_fill)
      {
         icon(builtin_icon:builtin_icon_alert, width: 20, height: 20);
         static_text(name:
            '$$$/AWS/FileInfoLib/Panels/Description/KeywordsHint=
               Commas can be used to separate keywords',
            vertical: align_center, horizontal: align_fill);
      }
```

```
    }
    mru_popup(xmp_path: 'Keywords', container_type: bag_struct,
              no_check: true, vertical: align_top, mru_append: true);
}

separator(horizontal: align_fill);

group(placement: place_row, spacing: gSpace, horizontal: align_fill,
      reverse: rtl_aware)
{
    static_text(name:
       '$$$/AWS/FileInfoLib/Panels/Description/CopyrightState=
       &Copyright Status:', vertical: align_center, font: font_big_right);
       popup(items:
              '$$$/AWS/FileInfoLib/Panels/Description/CopyrightPopupItems=
                 Unknown{};Copyrighted{True};Public Domain{False}',
              xmp_namespace: xap_ns_xap_rights, xmp_path: 'Marked');
}
group(placement: place_row, spacing: gSpace, horizontal: align_fill,
      reverse: rtl_aware)
{
    static_text(name:
       '$$$/AWS/FileInfoLib/Panels/Description/CopyrightNotice=
       C&opyright Notice:', vertical: align_top, font: font_big_right);
    edit_text(horizontal: align_fill, height: 54,
              xmp_namespace: xap_ns_xap_rights, xmp_path: 'Copyright',
              container_type: alt_struct, v_scroller: true);
    mru_popup(xmp_namespace: xap_ns_xap_rights,
              xmp_path: 'Copyright', container_type: alt_struct,
              no_check: true, vertical: align_top);
}

group(placement: place_column, spacing: gSpace, horizontal: align_fill,
      vertical: align_top)
{
    group(placement: place_row, spacing: gSpace, horizontal: align_fill,
          reverse: rtl_aware)
    {
       static_text(name:
          '$$$/AWS/FileInfoLib/Panels/Description/CopyrightInfoURL=
          Copyright Info URL:', vertical: align_top, font: font_big_right);
       group(placement: place_column, spacing: gSpace,
              horizontal: align_fill)
          {
          edit_text(horizontal: align_fill,
                    xmp_namespace: xap_ns_xap_rights,
                    xmp_path: 'WebStatement');
          button(name: '$$$/AWS/FileInfoLib/Panels/Description/GoToURL=
                  Go To URL...', vertical: align_bottom,
                 horizontal: align_right, label: false);
          }
```

```
        mru_popup(xmp_namespace: xap_ns_xap_rights,
                  xmp_path: 'WebStatement', no_check: true,
                  vertical: align_top, visible: false);
    }
}

separator(horizontal: align_fill);

group(placement: place_row, spacing: gSpace, horizontal: align_fill)
{
    group(placement: place_row, spacing: gSpace, horizontal: align_fill,
        reverse: rtl_aware)
    {
        group(placement: place_column, spacing: 5)
        {
            static_text(name:
                '$$$/AWS/FileInfoLib/Panels/Description/DateCreated=Created:',
                label: false, horizontal: align_right, vertical: align_top,
                font: font_big_right);
            static_text(name:
                '$$$/AWS/FileInfoLib/Panels/Description/DateModified=Modified:',
                label: false, horizontal: align_right, vertical: align_bottom,
                font: font_big_right);
        }
        group(placement: place_column, spacing: 5, horizontal: align_fill)
        {
            static_text(name:
                          '$$$/AWS/FileInfoLib/Panels/IntentionallyBlank=',
                          xmp_path: 'CreateDate', horizontal: align_fill,
                          vertical: align_top, truncate: true);
            static_text(name:
                          '$$$/AWS/FileInfoLib/Panels/IntentionallyBlank=',
                          xmp_path: 'ModifyDate', horizontal: align_fill,
                          vertical: align_bottom, truncate: true);
        }
    }
    group(placement: place_row, spacing: gSpace, horizontal: align_fill,
          reverse: rtl_aware)
    {
        group(placement: place_column, spacing: 5)
        {
            static_text(name:
                '$$$/AWS/FileInfoLib/Panels/Description/CreatorApplication=
                 Application:', label: false, horizontal: align_right,
                vertical: align_top, font: font_big_right);
            static_text(name:
                '$$$/AWS/FileInfoLib/Panels/Description/Format=Format:',
                label: false, horizontal: align_right, vertical: align_bottom,
                font: font_big_right);
        }
        group(placement: place_column, spacing: 5, horizontal: align_fill)
```

```
            {
               static_text(name: '$$$/AWS/FileInfoLib/Panels/IntentionallyBlank=',
                              xmp_path: 'CreatorTool', horizontal: align_fill,
                              vertical: align_top, truncate: true); <<truncate not
documented>>
               static_text(name: '$$$/AWS/FileInfoLib/Panels/IntentionallyBlank=',
                         xmp_path: 'Format', horizontal: align_fill,
                         vertical: align_bottom, truncate: true);
            }
         }
      }
}
</panel>
```

ZString Escape Sequences

Strings used in ZString values must use escape sequences to indicate certain characters. The following tables summarize the required escape sequences.

Backslash Escape Sequences

To include the backslash character itself in a string, use the sequence "\\".

Sequence	Glyph	Unicode	Unicode name	Alternate Unicode names
\	""	U+0022	QUOTATION MARK	APL quote
\n	[LF]	U+000A	LINE FEED	
\r	[CR]	U+000D	CARRIAGE RETURN	
\t	[HT]	U+0009	HORIZONTAL TABULATION	
\b	[BS]	U+0008	BACK SPACE	

Hat Escape Sequences

To include the hat character itself in a string, use the sequence "^^".

Sequence	Glyph	Unicode	Unicode name	Alternate Unicode names
^Q	"	U+0022	QUOTATION MARK	APL quote
^[	“	U+201C	LEFT DOUBLE QUOTATION MARK	DOUBLE TURNED COMMA QUOTATION MARK
^]	”	U+201D	RIGHT DOUBLE QUOTATION MARK	DOUBLE COMMA QUOTATOIN MARK
^{	‘	U+2018	LEFT SINGLE QUOTATION MARK	SINGLE TURNED COMMA QUOTATION MARK
^}	’	U+2019	RIGHT SINGLE QUOTATION MARK	SINGLE COMMA QUOTATION MARK
^C	©	U+00A9	COPYRIGHT SIGN	

Sequence	Glyph	Unicode	Unicode name	Alternate Unicode names
^R	®	U+00AE	REGISTERED SIGN	REGISTERED TRADE MARK SIGN
^T	™	U+2122	TRADEMARK SIGN	
^D	°	U+00B0	DEGREE SIGN	
^B	•	U+2022	BULLET	black small circle
^#	?	U+2318	PLACE OF INTEREST SIGN	COMMAND KEY
^!	¬	U+00AC	NOT SIGN	
^I	?	U+2206	INCREMENT	Laplace operator forward difference
^S	?	U+2211	N-ARY SUMMATION	summation sign

ISO-based Escape Sequences

Sequence	Glyph	Unicode	Description
#{endl}	*1	*1	Platform dependent end-of-line This is translated into CR+LF on Windows, CR on MacOS Carbon and LF on Unix.
#{tab}	[HT]	U+0009	horizontal tabulation
#{cr}	[CR]	U+000D	carriage return
#{lf}	[LF]	U+000A	line feed
#{nbsp}		U+00A0	no-break space = non-breaking space
#{iexcl}	¡	U+00A1	inverted exclamation mark
#{cent}	¢	U+00A2	cent sign
#{pound}	£	U+00A3	pound sign
#{curren}	¤	U+00A4	currency sign
#{yen}	¥	U+00A5	yen sign = yuan sign
#{brvbar}	¦	U+00A6	broken bar = broken vertical bar
#{sect}	§	U+00A7	section sign
#{uml}	¨	U+00A8	diaeresis = spacing diaeresis
#{copy}	©	U+00A9	copyright sign
#{ordf}	ª	U+00AA	feminine ordinal indicator

Sequence	Glyph	Unicode	Description
#{laquo}	«	U+00AB	left-pointing double angle quotation mark
#{not}	¬	U+00AC	not sign
#{shy}	-	U+00AD	soft hyphen = discretionary hyphen
#{reg}	®	U+00AE	registered sign = registered trade mark sign
#{macr}	¯	U+00AF	macron = spacing macron = overline
#{deg}	°	U+00B0	degree sign
#{plusmn}	±	U+00B1	plus-minus sign = plus-or-minus sign
#{sup2}	²	U+00B2	superscript two = superscript digit two
#{sup3}	³	U+00B3	superscript three = superscript digit three
#{acute}	´	U+00B4	acute accent = spacing acute
#{micro}	µ	U+00B5	micro sign
#{para}	¶	U+00B6	pilcrow sign = paragraph sign
#{middot}	·	U+00B7	middle dot = Georgian comma
#{cedil}	¸	U+00B8	cedilla = spacing cedilla
#{sup1}	¹	U+00B9	superscript one = superscript digit one
#{ordm}	º	U+00BA	masculine ordinal indicator
#{raquo}	»	U+00BB	right-pointing double angle quotation mark
#{frac14}	¼	U+00BC	vulgar fraction one quarter
#{frac12}	½	U+00BD	vulgar fraction one half
#{frac34}	¾	U+00BE	vulgar fraction three quarters
#{iquest}	¿	U+00BF	inverted question mark
#{Agrave}	À	U+00C0	latin capital letter A with grave = latin capital letter A grave
#{Aacute}	Á	U+00C1	latin capital letter A with acute
#{Acirc}	Â	U+00C2	latin capital letter A with circumflex
#{Atilde}	Ã	U+00C3	latin capital letter A with tilde
#{Auml}	Ä	U+00C4	latin capital letter A with diaeresis
#{Aring}	Å	U+00C5	latin capital letter A with ring above = latin capital letter A ring
#{AElig}	Æ	U+00C6	latin capital letter AE = latin capital ligature AE

Sequence	Glyph	Unicode	Description
#{Ccedil}	Ç	U+00C7	latin capital letter C with cedilla
#{Egrave}	È	U+00C8	latin capital letter E with grave
#{Eacute}	É	U+00C9	latin capital letter E with acute
#{Ecirc}	Ê	U+00CA	latin capital letter E with circumflex
#{Euml}	Ë	U+00CB	latin capital letter E with diaeresis
#{Igrave}	Ì	U+00CC	latin capital letter I with grave
#{Iacute}	Í	U+00CD	latin capital letter I with acute
#{Icirc}	Î	U+00CE	latin capital letter I with circumflex
#{Iuml}	Ï	U+00CF	latin capital letter I with diaeresis
#{ETH}	Ð	U+00D0	latin capital letter ETH
#{Ntilde}	Ñ	U+00D1	latin capital letter N with tilde
#{Ograve}	Ò	U+00D2	latin capital letter O with grave
#{Oacute}	Ó	U+00D3	latin capital letter O with acute
#{Ocirc}	Ô	U+00D4	latin capital letter O with circumflex
#{Otilde}	Õ	U+00D5	latin capital letter O with tilde
#{Ouml}	Ö	U+00D6	latin capital letter O with diaeresis
#{times}	×	U+00D7	multiplication sign
#{Oslash}	Ø	U+00D8	latin capital letter O with stroke = latin capital letter O slash
#{Ugrave}	Ù	U+00D9	latin capital letter U with grave
#{Uacute}	Ú	U+00DA	latin capital letter U with acute
#{Ucirc}	Û	U+00DB	latin capital letter U with circumflex
#{Uuml}	Ü	U+00DC	latin capital letter U with diaeresis
#{Yacute}	Ý	U+00DD	latin capital letter Y with acute
#{THORN}	Þ	U+00DE	latin capital letter THORN
#{szlig}	ß	U+00DF	latin small letter sharp s = ess-zed
#{agrave}	à	U+00E0	latin small letter a with grave = latin small letter a grave
#{aacute}	á	U+00E1	latin small letter a with acute
#{acirc}	â	U+00E2	latin small letter a with circumflex

Sequence	Glyph	Unicode	Description
#{atilde}	ã	U+00E3	latin small letter a with tilde
#{auml}	ä	U+00E4	latin small letter a with diaeresis
#{aring}	å	U+00E5	latin small letter a with ring above = latin small letter a ring
#{aelig}	æ	U+00E6	latin small letter ae = latin small ligature ae
#{ccedil}	ç	U+00E7	latin small letter c with cedilla
#{egrave}	è	U+00E8	latin small letter e with grave
#{eacute}	é	U+00E9	latin small letter e with acute
#{ecirc}	ê	U+00EA	latin small letter e with circumflex
#{euml}	ë	U+00EB	latin small letter e with diaeresis
#{igrave}	ì	U+00EC	latin small letter i with grave
#{iacute}	í	U+00ED	latin small letter i with acute
#{icirc}	î	U+00EE	latin small letter i with circumflex
#{iuml}	ï	U+00EF	latin small letter i with diaeresis
#{eth}	ð	U+00F0	latin small letter eth
#{ntilde}	ñ	U+00F1	latin small letter n with tilde
#{ograve}	ò	U+00F2	latin small letter o with grave
#{oacute}	ó	U+00F3	latin small letter o with acute
#{ocirc}	ô	U+00F4	latin small letter o with circumflex
#{otilde}	õ	U+00F5	latin small letter o with tilde
#{ouml}	ö	U+00F6	latin small letter o with diaeresis
#{divide}	÷	U+00F7	division sign
#{oslash}	ø	U+00F8	latin small letter o with stroke, = latin small letter o slash
#{ugrave}	ù	U+00F9	latin small letter u with grave
#{uacute}	ú	U+00FA	latin small letter u with acute
#{ucirc}	û	U+00FB	latin small letter u with circumflex
#{uuml}	ü	U+00FC	latin small letter u with diaeresis
#{yacute}	ý	U+00FD	latin small letter y with acute
#{thorn}	þ	U+00FE	latin small letter thorn

Sequence	Glyph	Unicode	Description
#{yuml}	ÿ	U+00FF	latin small letter y with diaeresis
#{quot}		"U+0022	quotation mark = APL quote
#{amp}	&	U+0026	ampersand
#{lt}	<	U+003C	less-than sign
#{gt}	>	U+003E	greater-than sign
#{OElig}	Œ	U+0152	latin capital ligature OE
#{oelig}	œ	U+0153	latin small ligature oe
#{Scaron}	Š	U+0160	latin capital letter S with caron
#{scaron}	š	U+0161	latin small letter s with caron
#{Yuml}	Ÿ	U+0178	latin capital letter Y with diaeresis
#{circ}	ˆ	U+02C6	modifier letter circumflex accent
#{tilde}	˜	U+02DC	small tilde
#{ensp}		U+2002	en space
#{emsp}		U+2003	em space
#{thinsp}	?	U+2009	thin space
#{zwnj}	?	U+200C	zero width non-joiner
#{zwj}	?	U+200D	zero width joiner
#{lrm}	?	U+200E	left-to-right mark
#{rlm}	?	U+200F	right-to-left mark
#{ndash}	–	U+2013	en dash
#{mdash}	—	U+2014	em dash
#{lsquo}	‘	U+2018	left single quotation mark
#{rsquo}	’	U+2019	right single quotation mark
#{sbquo}	,	U+201A	single low-9 quotation mark
#{ldquo}	“	U+201C	left double quotation mark
#{rdquo}	”	U+201D	right double quotation mark
#{bdquo}	„	U+201E	double low-9 quotation mark
#{dagger}	†	U+2020	dagger
#{Dagger}	‡	U+2021	double dagger

Sequence	Glyph	Unicode	Description
#{permil}	‰	U+2030	per mille sign
#{lsaquo}	‹	U+2039	single left-pointing angle quotation mark
#{rsaquo}	›	U+203A	single right-pointing angle quotation mark
#{euro}	€	U+20AC	euro sign