Creo® UI Editor

Java User's Guide

5.0.0.0

Contents

1

Creo UI Foundation Classes Introduction

This chapter provides an introduction to the basic concepts of the Creo UI Foundation classes.

Overview

The User Interface Foundation Classes (UIFC) provides a framework for creating, displaying and managing additions to the Creo user interface. New dialogs can be created using the Creo UI Editor, and then loaded into a Creo session. The UIFC is a platform and operating system independent toolkit, supporting trail files, mapkeys and a common appearance to the rest of the Creo user interface.

Basic Concepts

This sections provides more information on the basic concepts used in Creo UI Editor.

Dialogs

A dialog is the term used for all top-level windows by the UIFC. This includes anything from a Creo main window to an exit confirmation dialog box.

Modality

Dialogs can be defined to be either modal, also referred to as blocking or modeless. When the Activate function is called for a given dialog, modal dialogs prevent access to other individual dialogs or the whole application, whereas modeless dialogs allow the user to interact with the rest of the application as well as the modeless dialog itself.

In the activated state, modal dialogs start an event loop and process events and the function `ActivateDialog()` will only return when the dialog is exited from a callback function. Modeless dialogs on the other hand do not start an event loop so the call to activate the dialog returns immediately. Event processing for modeless dialogs is handled by the currently active event loop.

Dialog Lifecycle

The dialog lifecycle has 4 or 5 stages depending on whether it is modeless or modal. The steps to display a dialog are:

1. Create

Call the function `CreateDialog()` to create an instance of a dialog from a resource file. For example:

```
uifcComponent.CreateDialog (“MyDialogInstance”,
    “my_dialog_resource_file”);
```

Creating the dialog only loads the definition into memory; it does not show the dialog on the screen, which happens later.

2. Initialize

Once the dialog has been created, for example by loading a resource file, you can then set up run time values for the dialog or components within the dialog. For example, if the dialog relates to editing a file, you might want to set the title of the dialog to the name of the file. We recommend that you set the title and modify over components before the dialog is displayed on the screen as the values on the components can affect the overall size of the dialog and relative placement of components in the dialog. This avoids causing the dialog to resize, or cause visual disturbance, to the user due to the content in the dialog changing.

Also at this point you will need to setup the action listeners for the components in the dialog. For more information, see the section on Event processing on page 7.

3. Activate

Activate the dialog, that is, show the dialog on the screen by calling the function `ActivateDialog()`:

```
uifcComponent.ActivateDialog (“MyDialogInstance”);
```

If the dialog type is modal, this call will start a new event loop. The call will not return until the dialog exits from the event loop. Refer to the section 4. Exit on page 6.

For a modeless dialog, the activate call will display the dialog and return immediately.

4. Exit

A modal dialog stays in the `ActivateDialog()` call and has an event loop running while it is displayed. You need to exit the event loop, to dismiss the dialog. This can be done from an action listener by calling:

```
uifcComponent.ExitDialog (“MyDialogInstance”, status);
```

Specify the name of the dialog instance and an integer status value as the arguments to the call to `ExitDialog()`. The status value is used as the return status from the `ActivateDialog()` call. Note that you must exit the event loop for a modal dialog before it can be destroyed.

Note

Exiting the event loop for a dialog does not remove the dialog from the screen. It will still be displayed, and it is possible to activate the dialog again and start a new event loop.

For modeless dialogs, which do not have any associated event loop, there is no need to call the exit function.

5. Destroy

To finally remove the dialog from the screen after any event loop associated with the dialog has been exited, call:

```
uifcComponent.DestroyDialog (“my dialog instance”);
```

For modal dialogs this will normally be called immediately after the call to `ActivateDialog()`.

At this point the dialog will need to be created again before it can be reused.

Event processing

Once you have your dialog displayed on the screen, you need to be able to respond to the user interacting with it. This is done by the use of action listeners on the dialog and on the components within it.

Create an action listener by deriving a new class from the appropriate Object TOOLKIT Listener base class for the given component type. The following example shows you how to define an action listener for a PushButton component:

```
class MyButtonListener extends uifcDefaultPushButtonListener
{
    MyButtonListener() {};
    ~MyButtonListener() {};

    OnActivate (PushButton component);
};

PushButton ok_button = uifcPushButton.PushButtonFind("MyDialogInstance",
                                                      "ok_button");

MyButtonListner ok_listner = new MyButtonListner();
Ok_button.AddActionListener (ok_listner);
```

In this example, the `OnActivate()` method is overridden, which informs you when the user has activated the PushButton, that is, when the user has clicked on the component, or pressed the spacebar when the keyboard focus was on the component, or if the activate action was programmatically pushed from code. As you are defining your own class for the notification, you are free to include your own methods and data in the class, which allow for more versatility in associating your own data with the component.

Note

Certain action types are not recorded to trail files unless there is a method that had been derived for them from the base class. If you were to override all the notification methods in the Listener class, for example to have a general purpose class for all your components, then you may cause additional unwanted actions to be recorded into the output trail file. It is recommended that you only override those notifications that you need for a given individual component.

Text Display

Text displayed to the user in components can be either simple Unicode strings or a subset of HTML tags to control text attributes like the use of bold, italics, font size, and so on.

Images

The supported image formats are PNG, Jpeg, GIF, BMP, and ICO. If using an image to define a cursor it is recommended that you use an ICO file, to allow the definition of the cursor hotspot.

Component Positioning

The primary way to position components is via a Grid structure. Grids allow automatic relative placement of components and resizing of a component if the component is a child of a Dialog or Layout. Alternatively, components can be positioned and resized manually when the component is a child of a DrawingArea, NakedWindow, or PGLWindow. Component classes such as the Sash, Tab, or Table component define their own placement schemes for their child components.

Grid

The Layout and Dialog components both use a grid based positioning scheme for their child components. This consists of a recursive rectangular grid of cells similar to an HTML table or a spreadsheet. Each cell in the grid can either be empty or can contain a component or a nested sub-grid.

A grid cell has offset values in pixels for the top, bottom, left and right sides, which give the spacing between a component in the cell and the cell edges. You can also define attachments for the cell content, so that a component can have it's left, right, top or bottom edges fixed to the corresponding cell edge taking into account any offset defined for the edge, in any combination.

If you attach a component to only the left or right side, or the top or bottom of a grid cell, then the component will stick to that edge if the grid cell changes size or position. Attaching a component to both the left and right sides or both the top and bottom edges will cause the component to stretch to be the size of the grid cell, less any offsets in that direction.

A row or column in a grid can be defined as being either resizable or non-resizable. This controls the distribution of any size changes made to the Dialog or Layout component, so that the change in size in the horizontal or vertical direction is divided up between the row and columns that are marked as being resizable.

Button Sizes

By default, a toggle style PushButton in a dialog that is not in a menu and has no attachments will have the same width, based on the widest toggle style PushButton component in the dialog. You can explicitly control this behavior by using the `UseStandardWidth` attribute. When set to `TRUE`, the component will have the standard width behavior regardless of any attachments.

Note

When determining the widest component, the 'natural' size of the component is used, that is the size of the component before it is potentially stretched by any attachments.

You can also set CascadeButton and CheckButton components to have the standard width behavior by setting the `UseStandardWidth` attribute to `TRUE`.

Internationalization

Where possible you should define your dialogs using resource files rather than creating the dialogs and components in code. The strings defined in the resource file that are displayed on the user interface can be automatically extracted and used to create a translation file. Separate translation files can then be created for each supported language so that at run time the appropriately localized text is taken from the translation file.

Note

By default, the resource files contain English text strings, if any translations are missing, then displayed test will fallback to the English text in the resource file.

Textual input component such as the TextArea, InputPanel, and so on support input methods and right to left input.

Trail Files and Mapkeys

Actions such as the user clicking on a PushButton or selecting an item in a List component are automatically written into the output trail file for the session. For simple actions such as activating a PushButton, only the action type and the dialog and component names are recorded in the trail files. For more complex actions, such as selecting an item in a List, Table, OptionMenu, or RadioGroup, along with the action type and component name, the names of the items that were selected are also recorded in the trail file.

Having meaningful names for components and particularly in the case of the names of items in the component, will be helpful while examining trail files, for example, `ok_button` rather than `PushButton3`.

In the case of items in a List, where the content might change from session to session, such as a list of file names, you should base the names on a scheme that will be as far as possible invariant between times that the dialog is displayed. For example, if you use a numeric index for the item names, then this reduces the readability of the resulting trail file entries and will most likely prevent any mapkeys that use the component from working in another situation other than when the set of items in the component are exactly identical. Further more, if at some later point in time you add more items into the component, then a simple index will mean that the names written in an earlier trail file or mapkey will no longer map to the correct items. If however you used an invariant name, the mapping will be unaffected and trail files and mapkeys will still work.

Accelerators and Mnemonics

Accelerators and mnemonics are two different ways of controlling components via the keyboard.

Mnemonics

Mnemonics are shown as an underlined character in the label text of a component, using `Alt` + the underlined character will activate the component. The mnemonic is defined by putting an ampersand character in the label text of the component immediately in front of the character to be used, for example `&File`. To display a literal ampersand character you need to use two ampersands, for example `This && that`.

In the case of a PushButton or CheckButton component the mnemonic behaves as if the user clicked on the component. In the case of a Label or Layout component this will move the keyboard focus onto the component defined by the Focus attribute. In the case of a MenuBar component it will open the menu with the matching mnemonic and similarly for a CascadeButton it will open it's menu.

Mnemonics are only available to the user to use if they are shown in the currently active dialog, that is the dialog with the keyboard focus. The component with the mnemonic also needs to be visible. If duplicate mnemonics are used in the dialog for PushButton or CheckButton, then rather than immediately activating the component, the keyboard focus is cycled between the components with the same matching mnemonic, to allow the user to chose and activate using the spacebar.

Note

A best practice is to avoid having duplicate mnemonics as far as possible.

It is good practice to add mnemonics to all the components in a menu, as this allows the user to directly active a button in the menu by typing the sequence of key presses, rather than having to navigate through the menu using the arrow keys.

Note

When a menu pane is open, pressing the mnemonic character key will activate the mnemonic, the Alt key is not required. Also when the menu is open the scope of any mnemonics available to the user is limited to just those in the menu itself.

Accelerators

Unlike mnemonics, accelerators can be used on components that are not immediately visible in the dialog, that is, an accelerator can activate components that are in a menu such as a popup menu or a menu associated with a CascadeButton or MenuBar without having to open the menu.

Define an accelerator for a component using the `AcceleratorCode` attribute in Creo UI Editor. The accelerator consists of a character key and one or more modifier keys, such as, Ctrl, Alt, or Shift where one of them should be the Ctrl key. When a component with an accelerator is shown in a menu, the accelerator definition is automatically shown in a column on the right-hand side.

For component classes that support the `AcceleratorCode` attribute, the accelerator will call the Activate action on the component. The Dialog class is an exception to this, where the accelerator will call the Close action on the dialog, that is, using the accelerator will be similar to clicking the Close button on the dialog. It is a good practice to define an accelerator using the ‘Escape’ key on dialogs that contain transient content or short tasks, for example, prompts, queries or perhaps something like renaming an object. This allows the user to quickly get out of the dialog and should behave as though the task was cancelled.

Components

Refer to the *Creo UI Editor C++ User’s Guide* for more information on the user interface components.

2

User Interface Basics

This chapter describes the user interface for the Creo UI Editor in detail.

About the Creo UI Editor Main Window

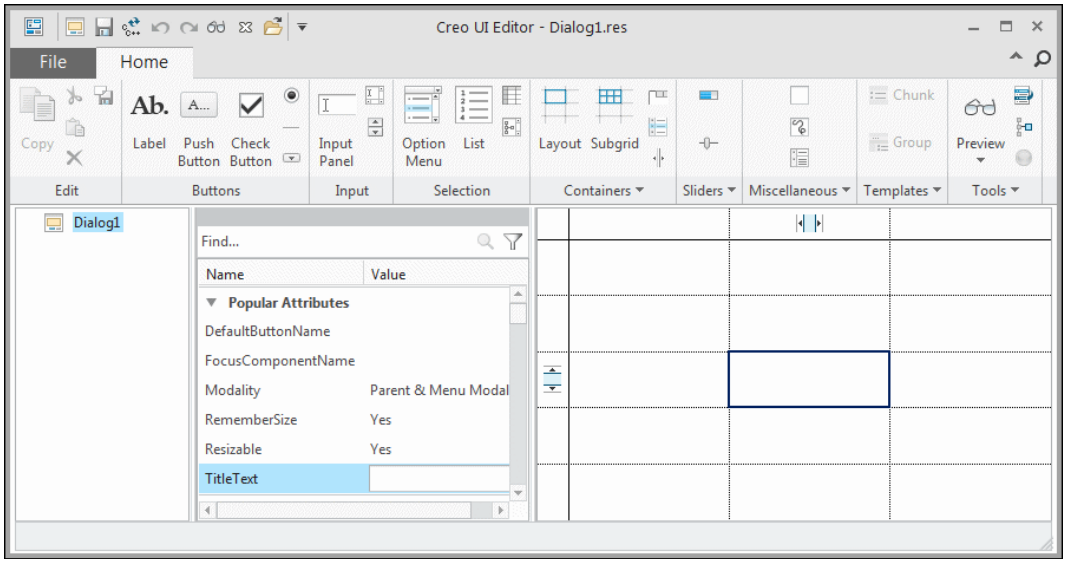
You can create dialog boxes using the Creo UI Editor. The dialog boxes are saved as resource files.

The Creo UI Editor user interface consists of the following elements:

- File menu
- Ribbon
- Quick Access Toolbar
- Tree
- Attribute list
- Work Area

Each Creo UI Editor dialog box opens in its own window. You can perform many operations from the ribbon in multiple windows without cancelling pending operations.

The following figure shows the various elements of the Creo UI Editor:



From Creo UI Editor 4.0 F000 onward, you can create dialog boxes which follow the Creo guidelines. Templates for dialog boxes are available which follow the Creo guidelines. The guidelines define the margins, positions and sometimes labels of components in the template. Using these templates consistency can be maintained in the look and feel of PTC products. The user interface created using these template enables a seamless integration in the relevant PTC product.

Using Creo UI Editor you can create two types of resource files:

- Dialog resource file—When you save a dialog box, a dialog resource file is created. Refer to the section Creating a New Dialog Box on page 19, for more information on Dialog boxes.
- Layout resource file—When you save a layout, a layout resource file is created. You can use the layout resource files in dialog boxes. Refer to the section Creating a Layout on page 26, for more information on Layout.

About the File Menu

The **File** menu allows you to create a new dialog box or work with an existing dialog box.

It has the following commands:

Command Name	Icon on File Menu or Quick Access Toolbar	Description
New Dialog		Creates a new dialog box. Select the required template. Refer to the section Creating a New Dialog Box on page 19, for more information on Dialog boxes.
New Layout		Creates a new layout. Select the required template. Refer to the section Creating a Layout on page 26, for more information on Layout.
Open		Opens an existing dialog box.
Save		Saves the dialog box.
Save As		Saves a copy of the dialog box as a resource file (`.res`).
Generate Code		Saves the source code to control the dialog box programmatically.
Close		Closes the current dialog box.
Help ▸ About Creo UI Editor		Displays the copyright and release information for Creo UI Editor.

Command Name	Icon on File Menu or Quick Access Toolbar	Description
Options		Enables you to change the general settings of Creo UI Editor. It also enables you to customize the ribbon and quick access toolbar.
Exit		Exits the Creo UI Editor.

Ribbon

The ribbon contains the command buttons organized within a set of tabs. On each tab, the related buttons are grouped. You can customize the ribbon.

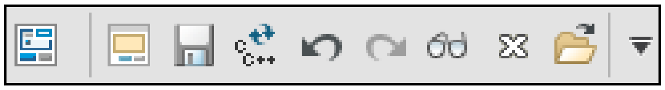
Right-click the ribbon and click **Customize the Ribbon**. Alternatively, click **File** ▶ **Options**.

You can perform the following customizations:

- Add the **Common** tab to the ribbon
- Add a new tab
- Add a new group
- Rename a tab or group
- Hide a tab or group
- Change the order of tabs, groups, commands, or cascades
- Add a new cascade to a group on the ribbon
- Modify the style of commands

Quick Access Toolbar

The Quick Access toolbar is available regardless of which tab is selected on the ribbon. By default it is located at the top of the Creo UI Editor window. It provides quick access to frequently used buttons, such as buttons for opening and saving files, creating new dialog boxes, generating code, closing dialog boxes, undo, redo, and so on. In addition, you can customize the Quick Access toolbar to include other frequently used buttons and cascading lists from the ribbon.



You can perform the following customizations on the Quick Access Toolbar:

- Add a command
- Remove a command
- Change the order of commands
- Change the position of the Quick Access toolbar
- Add a new cascade

Tree

The dialog box components are represented as a tree. Each branch of the tree corresponds to either individual components or parent container components that hold the child components. You can expand or collapse the tree on a branch level or on an individual layout level. When you select a component in the tree, the component is highlighted in the grid area. If the option **Preview ▶ Highlight in Preview** is selected, then the component in highlighted in the dialog box preview also.

Attribute List

The attribute list contains a list of attributes along with their default values for a component. All the possible values for an attribute are also listed. From the box, type or select the required value for the attribute. You can search for an attribute. You can also filter the attributes based on the following types of attributes:

- **Guideline Attributes**—Lists all the attributes whose values has been set according to the Creo guidelines. These attributes are indicated by yellow highlight in the attribute list. In the grid also, the attributes that follow Creo guidelines are indicated in yellow with the icon .
- **Modified Attributes**—Lists all the attributes that have been modified. These attributes are indicated by green highlight in the attribute list.
- **Other Attributes**—Lists the remaining attributes after excluding **Guideline Attributes** and **Modified Attributes**. These attributes are indicated by white highlight in the attribute list.

The attribute list panel can be moved and placed anywhere in the graphics area.

Command Search Tool

The command search tool enables you to find commands faster and preview the location of the command on the user interface. You can preview the location only if the command is located on the ribbon, Quick Access toolbar, or **File** menu. You can also run a command by clicking the command in the search list.

The tool displays the commands under following categories in the search list:

- **Commands**—All the commands on the ribbon, **File** menu, and Quick Access toolbar.
- **Commands not in the ribbon**—All the commands that not included in the ribbon.

To search for a command, follow these steps:

1. Click . A box appears next to .
2. Type a command name in the box. As you start typing, the commands that match the string are listed along with their respective icons (if available) under the following categories:
   - **Commands**
   - **Commands not in the ribbon**

   A **Setup** button is displayed at the end of the list.
3. Place your pointer over a command in the list. Creo UI Editor displays the command tooltip and a preview of the command location on the interface. The location is indicated by a different background color.
4. Do one of the following steps:
   - Click a command in the list to execute it and close the list.
   - To close the list without executing a command, click in the search box.
   - To refine search, do the following.
     a. Click **Setup**. The **Command Search Settings** dialog box opens.

        Note

        To open the **Command Search Settings** dialog box, you can also right-click the box next to and click **Setup**.

     b. Specify search criteria.
        - **Commands**—Select to search for commands on the ribbon, **File** menu, and Quick Access toolbar.
        - **Search in tooltip**—Select to search in tooltips.
        - **Match case**—Select to search only for commands that match the case of the word or the phrase that you typed.
        - **Match criteria** — Allows you to further refine the search. Select one of the following criteria.
          - **Any word beginning with**—Searches for commands beginning with the string specified in the **Command Search** box.

    - **Containing**—Searches for commands that contain the string specified in the **Command Search** box.
    - **Ending with**—Searches for commands that end with the string specified in the **Command Search** box.

c. Click **OK** and type a command name in the box. As you start typing, the command names that match the search criteria are listed.

Creating a New Dialog Box

To create a new dialog box, click **File** ▸ **New Dialog** or click on the Quick Access toolbar. It opens the **Select a Template** dialog box, which contains templates of dialog boxes. These templates follow the Creo guidelines. Select the required template. You can also select an empty dialog box template.

Note

It is recommended to use the templates provided with Creo UI Editor to create new dialog boxes. It is also recommended not to change the values set for attributes which follow Creo guidelines.

The work area displays grid cells and the tree area displays the name of dialog box as the parent node. Select components from the ribbon and add them to the dialog box. These components appear as child nodes of the parent node in the tree.

Click **File** ▸ **Save**. The dialog box is saved as a resource file with the same name as that of the parent node in the tree.

Note

You cannot save the resource file if you do not add components in it.

Refer to section Adding Components to the Dialog Box on page 19, for more information on adding components.

Adding Components to the Dialog Box

To add components:

1. Click the component from the ribbon.
2. Place the component in a single grid cell in the work area. You can add additional components as required to create the dialog box. Place the additional components in empty grid cells.
3. Double-click the component on the ribbon to add multiple instances of the component in the dialog box.

Opening and Closing the Dialog Box

You work with resource files when you open a dialog box and edit it.

1. Click **File** ▶ **Open** or click on the Quick Access toolbar. The **Open File** dialog box opens. The directory in the address bar defaults to one of the following items:
   - The directory in which Creo UI Editor has been installed.
   - The directory you last accessed to open, save, or save a copy of your file.
2. Locate the file to open in the default directory or select a different directory.

   To open the resource file, double-click it or click **OK**. The dialog box along with its components appears in the work area.

To close the dialog box, click **File** ▶ **Close** or click on the Quick Access toolbar.

Saving the Dialog Box

To save a resource file, click **File** ▶ **Save** or click on the Quick Access toolbar. The file is saved with the same name as displayed in the parent node of the tree.

To change the name of the dialog box before saving it:

1. Right-click the dialog box in the tree and select **Rename**. The name of the dialog box becomes editable in the tree.
2. Type a new name for the new dialog box.
3. To save the dialog box, click **File** ▶ **Save**.

Note

If you rename an existing dialog box and save the file, it is saved as a copy of the original file with the new name.

Saving a Copy of the Dialog Box

To save a copy of the dialog box:

1. Click **File** ► **Save As** or click on the Quick Access toolbar. The **Save As** dialog box opens. The directory in the address bar defaults to one of the following items:
   - The directory in which Creo UI Editor has been installed.
   - The directory you last accessed to open, save, or save a copy of your file.
2. You can accept the default directory or browse to a new directory.
3. In the **File Name** box, type a different name for the resource file.
4. Click **OK** in the **Save As** dialog box. The dialog box is saved as a resource file.

Saving the Code File

Once you create a dialog box using the Creo UI Editor, you can automatically generate the code files using the **Generate Code** command. This code can invoke the resource file to invoke the dialog box at runtime.

1. Click **File** ► **Generate Code** or click on the Quick Access toolbar. The **Generate Code** dialog box opens.
2. In the **Options** tab, select the language in which you want to save the code. You can generate the resource file code in:
   - **C++**—The resource file is saved as a `.cxx` file.
   - **Java**—The resource file is saved as a `.java` file.
3. In the **Actions** tab, specify the following:
   - **Classes**—Select the component class.
   - **Used Actions**—Move the actions that are valid for the component class from the **All Actions** list to the **Used Actions** list. Click **>>** or **<<** to move the actions across lists.
4. Click **OK**. The code is saved.

To Edit Properties of a Component

To edit the properties of a component, right-click and select the required command from the shortcut menu:

- **Cut**—Cuts the selected component from the tree.
- **Copy**—Copies the selected component from the tree.
- **Paste**—Pastes the copied component in the tree.
- **Delete**—Deletes the selected component from the tree.

- **Rename**—Renames the selected component in the tree.
- **Select Parent**—Selects the parent of the component in the tree.
- **Place**—Places the selected components in a **Subgrid**, **Chunk** or **Group**.
- **Reset to Guidelines Default**—Resets modified values to the default values for the attributes which follow the Creo guidelines in the selected component.
- **What's This?**—Displays the context sensitive help for the selected component.

Previewing a Dialog Box

Click **Home ▶ Preview** to preview the current dialog box. The preview is dynamically updated as you modify the dialog box.

Click **Home ▶ Preview ▶ Highlight in Preview** to highlight the component in the dialog box preview when it is selected in the tree.

Use the command **Home ▶ Locate in Tree** to locate a component in the tree, when it is selected in the preview.

Compatibility with Previous Releases

From Creo UI Editor 4.0 F000 onward, the format of the resource file has been changed. However, you can continue updating the resource files from releases prior to Creo UI Editor 4.0 F000 in the compatibility mode.

Working with Resource Files from Previous Releases

When you open resource files from a release prior to Creo 4.0 F000, by default the **Compatibility Mode** is enabled. When the **Compatibility Mode** is enabled, advanced Creo UI Editor 4.0 functionality is not available. On the **Home** tab, the group **Tools** is not available. Creo guidelines while creating dialog boxes is also not available. You can work with the resource files, add, edit, or remove components. When you save the resource file, it is saved in the old format. You can open and work with the resource files in the previous releases of Creo UI Editor.

While working with new or Creo UI Editor 4.0 files, **Compatibility Mode** is not enabled.

Converting Resource Files from Previous Releases

You can convert resource files from releases prior to Creo 4.0 F000 to Creo 4.0 files. Click **Home** ▶ **Compatibility Mode**. A warning message is displayed. Click **Yes** to exit the **Compatibility Mode**. When the **Compatibility Mode** is disabled, the Creo 4.0 functionality is available. When you save the resource file, it will be saved in the new Creo UI Editor 4.0 format. You cannot work with in releases prior to Creo 4.0 F000.

Note

If you have exited the **Compatibility Mode**, you can enter the **Compatibility Mode** again, by using the **Undo** command.

Converting Resource Files to Follow Creo Guidelines

From Creo 4.0 M010 onward, the new option **Guidelines Mode** enables you to convert resource files to follow the Creo guidelines. The following resource files can be converted to follow the Creo guidelines:

- Resource files created in releases prior to Creo 4.0 F000
- Resource files created in Creo 4.0 release using the **Blank Dialog** template.

Note

Creo guidelines are supported only in Creo 4.0 releases.

To convert your resource files to follow Creo guidelines, perform the following steps:

1. Select **Home** ▶ **Tools** ▶ **Guidelines Mode**.

   The resource file is updated to follow the Creo guidelines. The resource file is converted to the new Creo UI Editor 4.0 format. The resource file will no longer work in releases prior to Creo 4.0 F000.

   For example, the offset and resizing attributes of the components will be set as per the Creo guidelines.
2. In the tree, right-click the parent node, and select **Convert to Dialog Template**. The **Convert to Dialog Template** dialog box appears.

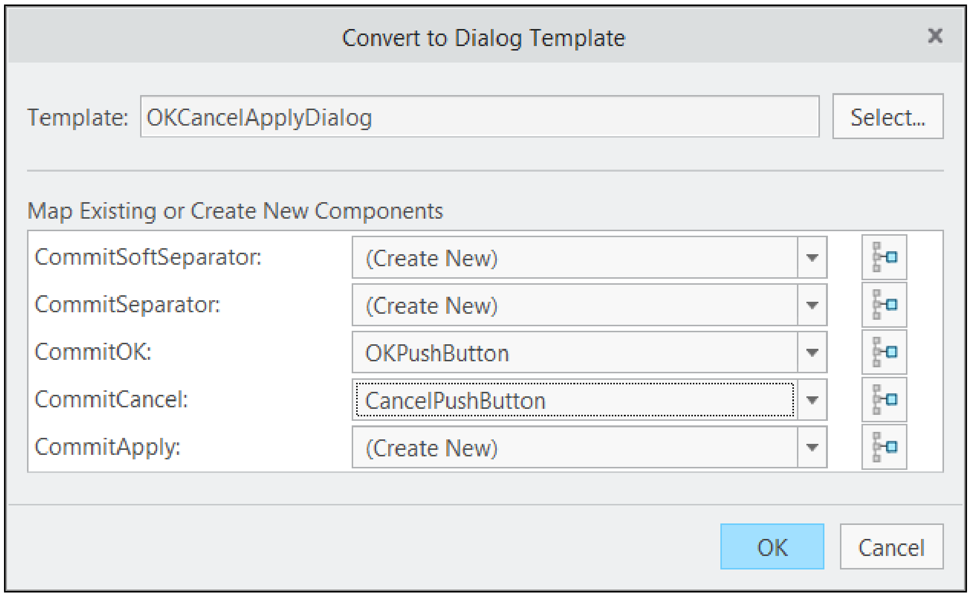
3. In the **Template** field, depending on the design of your existing resource file a dialog box template is recommended. You can also select another template.

   If Creo UI Editor cannot find a template most suitable for your existing resource file design, you will have to specify a template.

   Depending on the template, Creo UI Editor maps or creates new components. In the **Map Existing or Create New Components** field, a list of existing mapped components and new components is displayed.

   - It lists the existing components in the resource file that can be directly mapped to the components of the selected template. In case the resource file has more than one component that can be mapped, Creo UI Editor maps the most suitable component. You can select another component from the list or use to select the component in the preview.

     For example, if your existing resource file has two buttons, **OK** and **Cancel**, and you have selected a template with **Apply**, **OK**, and **Cancel** buttons, the Creo UI Editor recommends the mapping as below for **OK** and **Cancel** buttons.

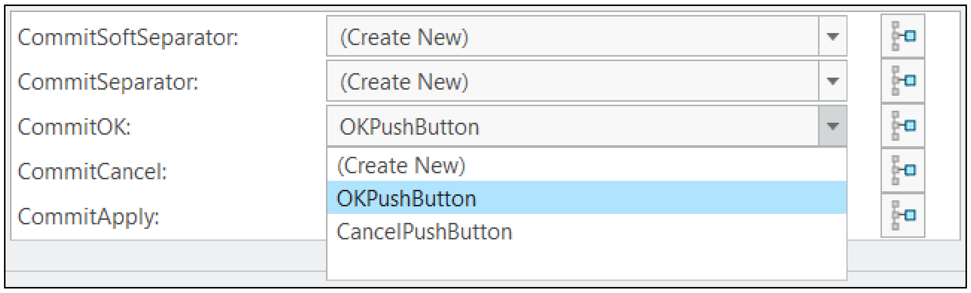


Template: OKCancelApplyDialog

Map Existing or Create New Components

CommitSoftSeparator:	(Create New)
CommitSeparator:	(Create New)
CommitOK:	OKPushButton
CommitCancel:	CancelPushButton
CommitApply:	(Create New)

     You can also select another component from the list.


CommitSoftSeparator: (Create New)
CommitSeparator: (Create New)
CommitOK: OKPushButton
CommitCancel:
CommitApply:
(Create New)
OKPushButton
CancelPushButton

- If the selected template requires additional components, then Creo UI Editor recommends and automatically creates the new components. In the image above, new components are created for the **Apply** button and separators.

4. Click **OK** to apply the template.

Note

If you want to exit the **Guidelines Mode**, and go back to the original design without Creo guidelines, use the **Undo** command in the current session. If you save the resource file and exit the session, you will not be able to return to the original design.

Changing the Tab Order in a Dialog Box or Component

The tab order of a user interface determines the order in which the components will receive mouse or keyboard input focus. In a dialog box, by default the tab order is determined by the position of the components in the grid, that is, from left to right and top to bottom. To change the tab order, perform the following steps:

1. Select the dialog box or parent component in the tree.
2. Click **Home** ▶ **Tab Order**.

   The **Tab Order** dialog box opens. It lists the components in the tab order for the selected component.
3. Select a component and use the up and down arrows to move the component and change the tab order.
4. After you have set the new tab order, click **OK**.

Creating a Layout

Layout is a collection of various components. This collection of components is treated as a single entity on the user interface. When you can save a Layout, it creates a Layout resource file. This resource file can be used in dialog boxes. To create a Layout, perform the following steps:

1. Click **File ▶ New Layout** or click on the Quick Access toolbar. It opens the **Select a Template** dialog box, which contains following templates of layouts
   - **Layout**—Creates a blank Layout without any guidelines.
   - **Layout With Dialog Guidelines**—Creates a Layout which follows the Creo guidelines. It is recommended to use **Layout With Dialog Guidelines** template.
2. Select the required template.
3. Click **OK**.

   The work area displays grid cells and the tree area displays the name of layout as the parent node.
4. Select components from the ribbon and add them to the layout. These components appear as child nodes of the parent node in the tree.
5. Click **File ▶ Save**. The layout is saved as a resource file with the same name as that of the parent node in the tree.