PTC® Live Global

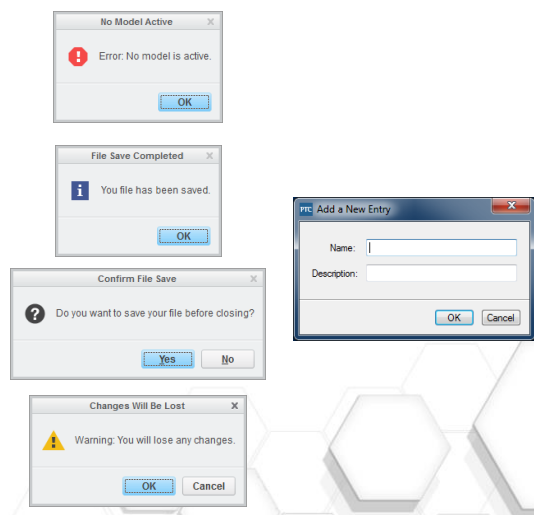
CUST 211 - Customizing PTC Creo Parametric User Interfaces in Java and C++

Mark Stallard
Software Developer
Raytheon Company
stallard@raytheon.com

Tuesday, June 9, 2015

Topics Presented

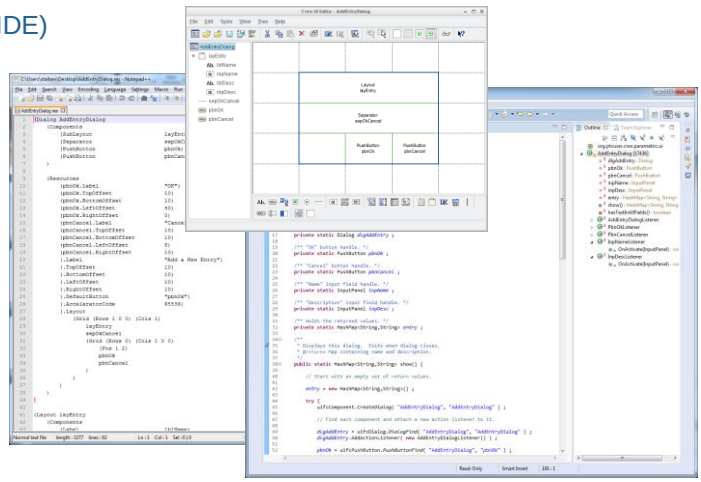
- UI Basics
  - Tools for UI Creation
  - Creo Message Dialogs
  - Ribbon Menu
- Creo UI Editor
  - Creating Dialog Boxes
  - Generating Code
- Organizing UI Code
  - Naming Conventions



Tools for Editing Creo UIs

PTC® Live Global

- Integrated Development Environment (IDE)
  - Eclipse for Java
  - Visual Studio for C++
- PTC Creo UI Editor
  - New in PTC Creo Parametric 3.0
  - Speeds dialog creation
  - Finds errors in old files
- Advanced Text Editor
  - Instead of Windows Notepad..
    - Try Notepad++, Textpad or UltraEdit
  - View multiple files
  - Reload changed files
  - Brace Matching
    - Very important with UI files

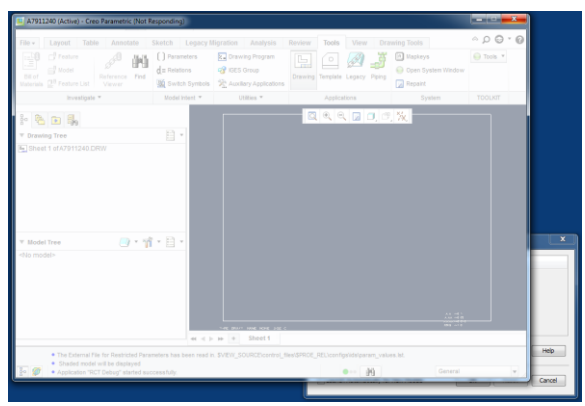


Why Use PTC UI with Java?

PTC® Live Global

Java already has multiple UI frameworks, such as Swing

- OTK Java Maintains Creo Look & Feel
- Creo UI Editor Generates Code
  - This will get you started
  - Less Work than Swing
- Also… Window Stack Issues
  - Swing Dialog Stuck Behind Creo
  - Blocks Input to Creo
  - User Doesn't Know Alt+Tab
  - Use Dummy Jframe Workaround
- Problems Combining Swing & Message Dialogs
  - Recent Example:
    - Display Web Page from Jlink
    - Needed Creo Message Dialog
    - Stack Issues Returned

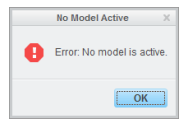


Message Dialogs

PTC® Live Global

Some Problems with Jlink & OTK Message Dialogs

- Complicated Setup
  - Create MessageDialogOptions
  - Add MessageButtons
  - Add Icon
  - Set Title
  - Get Session handle
  - Display Dialog with Message
- Needs try/catch Block
  - Program may already be in catch
  - Don't want to nest try/catch
- Too much code to show an error
- We need a MessageDialog class



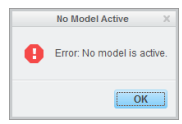

```
31      // Need to report an error to the user.
32
33      String title = "No Model Active" ;
34      String message = "Error: No model is active." ;
35
36      try {
37          // Create dialog options with an OK button and an Error icon.
38
39          MessageDialogOptions options = pfcUI.MessageDialogOptions_Create() ;
40          MessageButtons buttons = MessageButtons.create() ;
41          buttons.insert( 0, MessageButton.MESSAGE_BUTTON_OK ) ;
42          options.SetButtons( buttons ) ;
43          options.SetMessageDialogType( MessageDialogType.MESSAGE_ERROR ) ;
44
45          // Add the dialog title
46          // Show the dialog
47
48          options.SetDialogLabel( title ) ;
49          Session session = pfcGlobal.GetProESession() ;
50          session.UIShowMessageDialog( message, options ) ;
51      } catch ( jxthrowable ex ) {
52
53          // Can't display a dialog?  Now we're really in trouble!
54
55          StringBuilder sb = new StringBuilder( "Error: can't display error dialog.\n" ) ;
56          sb.append( "Title:   " ).append( title ).append( "\n" ) ;
57          sb.append( "Message: " ).append( message ).append( "\n" ) ;
58          System.out.println( sb.toString() ) ;
59      }
```

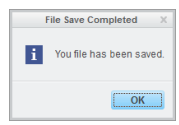

Designing a MessageDialog Class

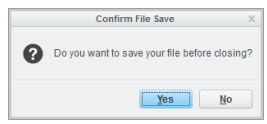
PTC® Live Global

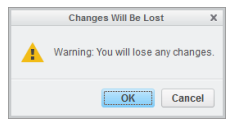
Reducing each MessageDialog to One Line

- Available in Jlink and OTK Java
- Create static methods
  - Call without new()
  - Display only one at a time
- Support common usage
  - Error Icon with OK Button
  - Information Icon with OK Button
  - Question Icon with Yes and No Buttons
    - Returns boolean value
  - Warning Icon with OK and Cancel Buttons
    - Returns boolean value









Writing a MessageDialog Class

PTC® Live Global

A high-level view of this class

- No MessageDialog class in Jlink or OTK
  - …but put yours in your own package namespace
  - I chose org.ptcuser.creo.parametric.ui
- Use static methods
- Separate methods for each dialog type
- showError is overloaded
  - Show a simple error message
  - Show a jxthrowable message

```
1  package org.ptcuser.creo.parametric.ui ;
2
3  import com.ptc.cipjava.jxthrowable ;
11
12 /**
13  * Collection of static methods for quickly displaying Creo
14  * MessageDialogs.
15  * @author Mark Stallard
16  */
17 public class MessageDialog {
18     /**
19      * Shows a Creo error dialog with an "OK" button.
20      * @param title   Title bar text.
21      * @param message Error message to show in dialog body.
22      */
23     public static void showError( String title, String message ) {
36
37     /**
38      * Shows a Creo error dialog with an "OK" button.
39      * @param title     Title bar text.
40      * @param exception Exception with message to show in dialog body.
51      */
52     public static void showError( String title, jxthrowable exception ) {
57
58     /**
59      * Shows a Creo information dialog with an "OK" button.
60      * @param title   Title bar text.
61      * @param message Message to show in dialog body.
62      */
63     public static void showInfo( String title, String message ) {
76
77     /**
78      * Shows a Creo question dialog with "Yes" and "No" buttons.
79      * @param title   Title bar text.
80      * @param message Question to show in dialog body.
91      * @return        true if user clicked "Yes".
92      */
93     public static boolean showQuestion( String title, String message ) {
106
107    /**
108     * Shows a Creo question dialog with "OK" and "Cancel" buttons.
109     * @param title   Title bar text.
110     * @param message Warning to show in dialog body.
130     * @return        true if user clicked "OK".
131     */
132    public static boolean showWarning( String title, String message ) {
154 }
155
```


Code for Error Dialog

PTC® Live Global

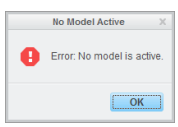
Here's the expanded code for method *showError*

- You can write your own message

```
110          MessageDialog.showError( "No Model Active",
111              "Error: No model is active." ) ;
```

- Or show an exception's message:

```
38      try {
39
40          // Load the ribbon definition file.
41
42          Session session = pfcGlobal.GetProESession() ;
43          session.RibbonDefinitionfileLoad( "messagedialogs_home.rbn" ) ;
44
45          // ...more code here...
46
47      } catch ( jxthrowable ex ) {
48          MessageDialog.showError( "Exception Caught", ex ) ;
49      }
```




```
18     /**
19      * Shows a Creo error dialog with an "OK" button.
20      * @param title   Title bar text.
21      * @param message Error message to show in dialog body.
22      */
23     public static void showError( String title, String message ) {
24         try {
25             // Create dialog options with an OK button and an Error icon.
26
27             MessageDialogOptions options = pfcUI.MessageDialogOptions_Create() ;
28             MessageButtons buttons = MessageButtons.create() ;
29             buttons.insert( 0, MessageButton.MESSAGE_BUTTON_OK ) ;
30             options.SetButtons( buttons ) ;
31             options.SetMessageDialogType( MessageDialogType.MESSAGE_ERROR ) ;
32
33             // Add the dialog title
34             // Show the dialog
35
36             options.SetDialogLabel( title ) ;
37             Session session = pfcGlobal.GetProESession() ;
38             session.UIShowMessageDialog( message, options ) ;
39         } catch ( jxthrowable ex ) {
40             StringBuilder sb = new StringBuilder( "Error: can't display error dialog.\n" ) ;
41             sb.append( "Title:   " ).append( title ).append( "\n" ) ;
42             sb.append( "Message: " ).append( message ).append( "\n" ) ;
43             System.out.println( sb.toString() ) ;
44         }
45     }
46
47     /**
48      * Shows a Creo error dialog with an "OK" button.
49      * @param title     Title bar text.
50      * @param exception Exception with message to show in dialog body.
51      */
52     public static void showError( String title, jxthrowable exception ) {
53         StringBuilder sb = new StringBuilder( "Exception caught: " ) ;
54         sb.append( exception.toString() ) ;
55         showError( title, sb.toString() ) ;
56     }
```

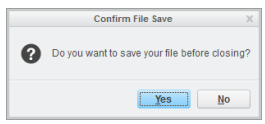
Code for Question Dialog

PTC® Live Global

Here's the expanded code for method *showQuestion*

- *showQuestion* returns a boolean value
  - True if user clicked "Yes"
  - False if user clicked "No"
- Sample usage:

```
boolean wantSave = MessageDialog.showQuestion( "Confirm File Save",
    "Do you want to save your file before closing?" ) ;

if ( wantSave ) {
    MessageDialog.showInfo( "Saving File",
        "File will be saved." ) ;
} else {
    MessageDialog.showInfo( "Discarding Changes",
        "Changes will be discarded." ) ;
}
```




```
/**
 * Shows a Creo question dialog with "Yes" and "No" buttons.
 * @param title    Title bar text.
 * @param message Question to show in dialog body.
 * @return        true if user clicked "Yes".
 */
public static boolean showQuestion( String title, String message ) {
    boolean wasYesClicked = false ;

    try {
        //Create dialog options with Yes/No buttons and a Question icon.

        MessageDialogOptions options = pfcUI.MessageDialogOptions_Create() ;
        MessageButtons buttons = MessageButtons.create() ;
        buttons.insert( 0, MessageButton.MESSAGE_BUTTON_YES ) ;
        buttons.insert( 0, MessageButton.MESSAGE_BUTTON_NO ) ;
        options.SetButtons( buttons ) ;
        options.SetMessageDialogType( MessageDialogType.MESSAGE_QUESTION ) ;

        // Add the dialog title
        // Show the dialog

        options.SetDialogLabel( title ) ;
        Session session = pfcGlobal.GetProESession() ;
        MessageButton clickedButton = session.UIShowMessageDialog( message, options ) ;

        // Determine which button the user clicked.

        wasYesClicked = ( clickedButton == MessageButton.MESSAGE_BUTTON_YES ) ;
    } catch ( jxthrowable ex ) {
        StringBuilder sb = new StringBuilder( "Error: can't display information dialog.\n" ) ;
        sb.append( "Title:   " ).append( title ).append( "\n" ) ;
        sb.append( "Message: " ).append( message ).append( "\n" ) ;
        System.out.println( sb.toString() ) ;
    }

    return wasYesClicked ;
}
```


A C++ MessageDialog Class

PTC® Live Global

I wrote this before I had Object Toolkit C++

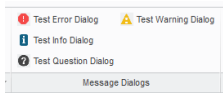
- C++ class is Pro::UI::MessageDialog
  - Namespace Pro::UI
  - Class MessageDialog
- Method Pro::UI::MessageDialog::Display
  - Wraps Pro/Toolkit method ProUIMessageDialogDisplay
  - Throws exceptions if error found
    - Subclasses of Pro::Tk::Exception
    - Wrote these exceptions myself
- Method Pro::UI::MessageDialog::DisplayOk
  - Information dialog with OK button
  - Implemented for both 8- and 16-bit strings

```
// Displays the UI Message Dialog

ProUIMessageButton Pro::UI::MessageDialog::Display( ProUIMessageType type, wstring title_cpp,
    wstring msg_txt_cpp, vector<ProUIMessageButton> buttons_cpp, ProUIMessageButton def_button ) {
    ProUIMessageButton *buttons = Pro::Array::FromVector( buttons_cpp ) ;
    ProUIMessageButton user_choice ;
    ProError error = ProUIMessageDialogDisplay( type, ( wchar_t * ) title_cpp.c_str(),
        ( wchar_t * ) msg_txt_cpp.c_str(), buttons, def_button, &user_choice ) ;

    switch ( error ) {
    case PRO_TK_NO_ERROR :
        break ;
    case PRO_TK_BAD_INPUTS :
        throw Pro::Tk::Bad_Inputs( L"ProUIMessageDialogDisplay",
            L"Bad inputs" ) ;
    case PRO_TK_GENERAL_ERROR :
        throw Pro::Tk::General_Error( L"ProUIMessageDialogDisplay",
            L"The function failed <!-- Objectname: ProUI -->" ) ;
    default :
        throw Pro::Tk::UndocumentedException( L"ProUIMessageDialogDisplay",
            L"An undocumented error occured." ) ;
    }

    return user_choice ;
}

// Displays an info dialog with a single "OK" button.

void Pro::UI::MessageDialog::DisplayOk( wstring title_cpp, wstring msg_txt_cpp ) {
    vector<ProUIMessageButton> btn_list ;
    btn_list.push_back( PRO_UI_MESSAGE_OK ) ;

    Pro::UI::MessageDialog::Display( PROUIMESSAGE_INFO, title_cpp, msg_txt_cpp,
        btn_list, PRO_UI_MESSAGE_OK  ) ;
}

// Displays an info dialog with a single "OK" button.

void Pro::UI::MessageDialog::DisplayOk( string title_cpp, string msg_txt_cpp ) {
    DisplayOk( wstring( title_cpp.begin(), title_cpp.end() ),
        wstring( msg_txt_cpp.begin(), msg_txt_cpp.end() )) ;
}
```

Testing Your MessageDialog Class

PTC® Live Global

- Write a Simple Test Application
  - Application is named MessageDialogTest
  - Creates ribbon buttons to test dialogs
  - Use as “skeleton” for future projects
- Class MessageDialogTest
  - Application class
  - Contains methods *start* and *stop*
  - Contains inner classes for ActionListeners

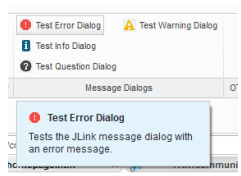



```
/**
 * JLink application to test Creo message dialogs.
 * @author Mark Stallard
 */
public class MessageDialogTest {
    /**
     * Called by Creo when application launches.
     */
    public static void start() {
        createCommands() ;
    }

    /**
     * Called by Creo when application terminates.
     */
    public static void stop() {
    }

    /**
     * Creates commands invoked by the ribbon UI.
     */
    private static void createCommands() {

    /**
     * Inner class for handling the "Test Error Dialog" command.
     */
    static class TestErrorDialogListener extends DefaultUICommandActionListener {

    /**
     * Inner class for handling the "Test Info Dialog" command.
     */
    static class TestInfoDialogListener extends DefaultUICommandActionListener {

    /**
     * Inner class for handling the "Test Question Dialog" command.
     */
    static class TestQuestionDialogListener extends DefaultUICommandActionListener {

    /**
     * Inner class for handling the "Test Warning Dialog" command.
     */
    static class TestWarningDialogListener extends DefaultUICommandActionListener {
}
```


Creating the Commands

PTC® Live Global

- Loading the Ribbon
  - Disable *RibbonDefinitionFileLoad* line
  - Run program
  - Create .rbn file
  - Enable *RibbonDefinitionFileLoad* line
- Creating an ActionListener
  - Use small, individual classes for each button
  - Inner classes work best
    - Better named than anonymous
- Create a command
  - Add icon
  - Tie to message file entries




```
    /**
     * Creates commands invoked by the ribbon UI.
     */
    private static void createCommands() {
        try {

            // Load the ribbon definition file.

            Session session = pfcGlobal.GetProESession() ;
            session.RibbonDefinitionFileLoad( "messagedialogs_home.rbn" ) ;  // Disable until you create .rbn file

            // Create Test Error Dialog Command.

            UICommandActionListener lsnErrorDialog = new TestErrorDialogListener() ;
            UICommand cmdErrorDialog = session.UICreateCommand( "Test Error Dialog", lsnErrorDialog ) ;
            cmdErrorDialog.SetIcon( "Error.png" ) ;
            cmdErrorDialog.Designate( "msg_messagedialog.txt",
                    "MessageDialog Test Error -label-",
                    "MessageDialog Test Error -help-",
                    "MessageDialog Test Error -desc-" ) ;

            // Create Test Info Dialog Command.

            UICommandActionListener lsnInfoDialog = new TestInfoDialogListener() ;
            UICommand cmdInfoDialog = session.UICreateCommand( "Test Info Dialog", lsnInfoDialog ) ;
            cmdInfoDialog.SetIcon( "Info.png" ) ;
            cmdInfoDialog.Designate( "msg_messagedialog.txt",
                    "MessageDialog Test Info -label-",
                    "MessageDialog Test Info -help-",
                    "MessageDialog Test Info -desc-" ) ;

            // Create Test Question Dialog Command.

            UICommandActionListener lsnQuestionDialog = new TestQuestionDialogListener() ;
            UICommand cmdQuestionDialog = session.UICreateCommand( "Test Question Dialog", lsnQuestionDialog ) ;
            cmdQuestionDialog.SetIcon( "Question.png" ) ;
            cmdQuestionDialog.Designate( "msg_messagedialog.txt",
                    "MessageDialog Test Question -label-",
                    "MessageDialog Test Question -help-",
                    "MessageDialog Test Question -desc-" ) ;

            // Create Test Warning Dialog Command.

            UICommandActionListener lsnWarningDialog = new TestWarningDialogListener() ;
            UICommand cmdWarningDialog = session.UICreateCommand( "Test Warning Dialog", lsnWarningDialog ) ;
            cmdWarningDialog.SetIcon( "Warning.png" ) ;
            cmdWarningDialog.Designate( "msg_messagedialog.txt",
                    "MessageDialog Test Warning -label-",
                    "MessageDialog Test Warning -help-",
                    "MessageDialog Test Warning -desc-" ) ;
        } catch ( jxthrowable ex ) {
            MessageDialog.showError( "Command Setup Failed", "Can't create commands for Creo UI." ) ;
        }
    }
```

More About ActionListeners

PTC® Live Global

A Tiny Class for Each Button

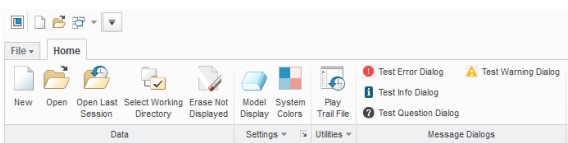
- Each Button Needs an ActionListener Class
- Keep ActionListeners Brief
  - Best if they call other classes or method
- Implement as Inner Classes
  - They can still call “outer” methods
- Avoid Anonymous Inner Classes
  - May confuse the less experienced
  - Good names are helpful
  - Bad names are better than no names

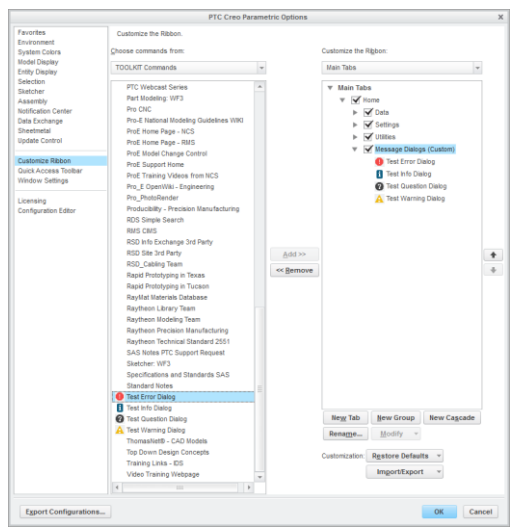
```
/**
 * JLink application to test Creo message dialogs.
 * @author Mark Stallard
 */
public class MessageDialogTest {
    /**
     * Called by Creo when application launches.
     */
    public static void start() {
        createCommands() ;
    }

    /**
     * Called by Creo when application terminates.
     */
    public static void stop() {
    }

    /**
     * Creates commands invoked by the ribbon UI.
     */
    private static void createCommands() {

    /**
     * Inner class for handling the "Test Error Dialog" command.
     */
    static class TestErrorDialogListener extends DefaultUICommandActionListener {
        public void OnCommand() {
            MessageDialog.showError( "No Model Active",
                    "Error: No model is active." ) ;
        }
    }

    /**
     * Inner class for handling the "Test Info Dialog" command.
     */
    static class TestInfoDialogListener extends DefaultUICommandActionListener {

    /**
     * Inner class for handling the "Test Question Dialog" command.
     */
    static class TestQuestionDialogListener extends DefaultUICommandActionListener {

    /**
     * Inner class for handling the "Test Warning Dialog" command.
     */
    static class TestWarningDialogListener extends DefaultUICommandActionListener {
}
```


Notes on the Ribbon

PTC® Live Global

- Separate files for each supported mode
  - For example: Part, Assembly and Drawing
- Name your files accordingly
  - Part: test_msg_dlg_part.rbn
  - Assembly: test_msg_dlg_assy.rbn
  - Drawing: test_msg_dlg_dwg.rbn
- Home Ribbon is Good for Testing
  - Available with no active model
  - Tests quickly
  - Plenty of room

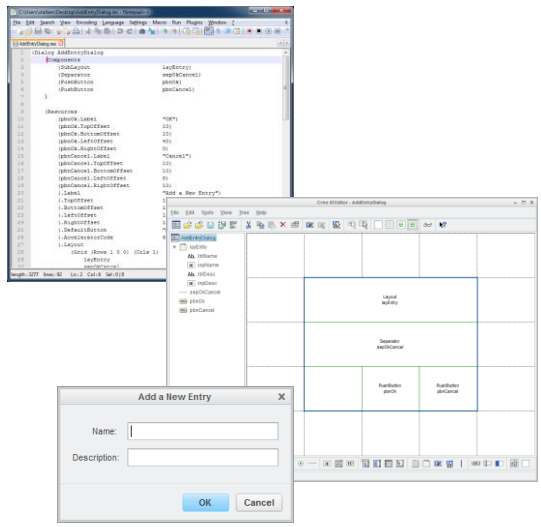




Using the PTC Creo UI Editor

PTC® Live Global

- Big Improvement
  - Create dialogs quickly
  - Start from template dialogs
  - Edit and test quickly
  - Find errors in existing dialogs
  - Write Java or C++ Source Code
- …But Hang on to Your Text Editor
  - Some Components (like Sash) Cause Errors
  - Misnamed Attributes
    - OptionMenu.VisibleRows vs. DropDownHeight
  - Entries for Non-Existent Attribues
  - Can't Re-Parent or Re-Order Components

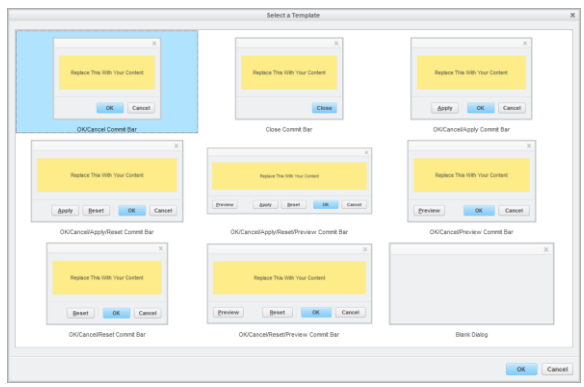


Template Dialogs

PTC® Live Global

Creo UI Editor can give you a running start.

- Select File > New
  - Dialog at right appears
- Choose From Nine Templates
  - First two are most common
- Templates Have Base Components
  - Bottom button row
  - Horizontal separator
  - Token label component
- Base Components are Already Positioned
  - This will save you time

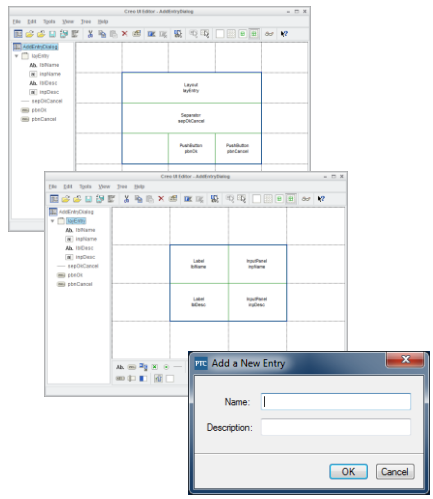


Creating a Simple Dialog

PTC® Live Global

Using the Creo UI Editor

- Pick a Template with OK & Cancel buttons
- Add Layout layEntry
- Populate layEntry
  - Label lblName
  - Input Panel inpName
  - Label lblDesc
  - Input Panel inpDesc
- Renamed Separator to sepOkCancel
- Do you patterns in these names?

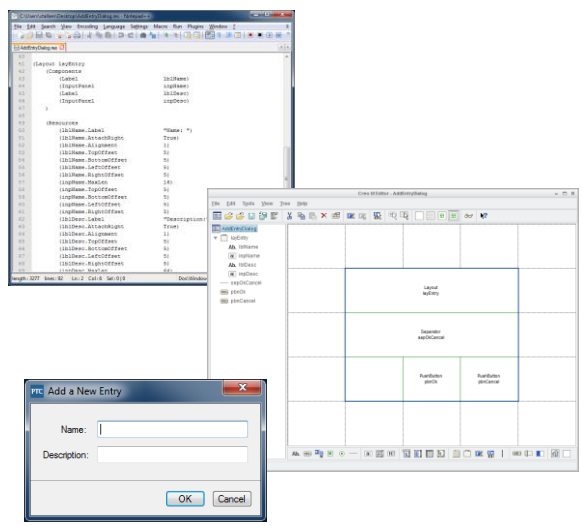


UI Naming Conventions

PTC® Live Global

Good Naming Helps Code Organization

- Why is naming a big deal?
  - Too many things to name
  - Too many name clashes
- Need to name active things…
  - Dialog Boxes, Push Buttons, Input Panels…
- …and *inactive* things:
  - Labels, Layouts, Separators
- Avoid using default names
  - "Label1" means very little
- Name must show **type, purpose** and **relationships**

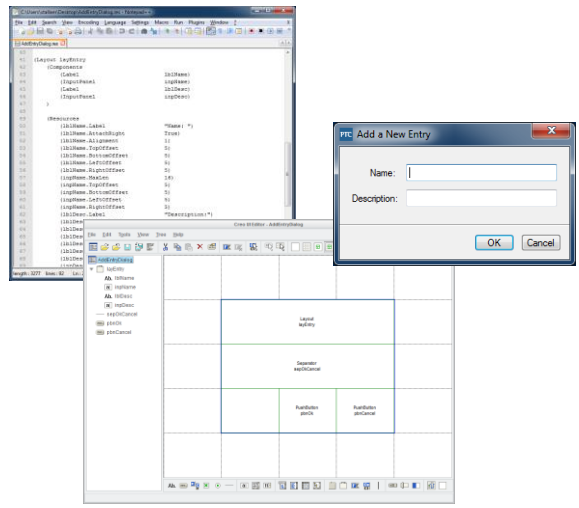


UI Naming Conventions

PTC® Live Global

Leszynski Naming Convention

- Names contain type and purpose
  - Camel-case names (example: lblDesc)
  - 3-letter prefix for type (“lbl” = “label”)
  - Suffix for purpose (“Desc” = “Description”)
- Related components use common suffix
  - lblName and inpName
  - lblDesc and inpDesc
- Won’t my IDE help me here?
  - Not with Creo Parametric dialogs
  - Source code uses text literals, like “inpDesc”
  - IDE won’t know PTC’s .res format



Java Code Generated by Creo UI Editor

PTC® Live Global

Code Generated for Class AddEntryDialog

- This code was generated by the UI Editor
  - Some sections are collapsed
  - Some blank lines were removed
- Inner ActionListener classes are most empty
  - AddEntryDialogListener contains method onClose
  - We don’t necessarily want all of these
- To use this class:
  - Create an instance
    - AddEntryDialog dialog = new AddEntryDialog() ;
  - Invoke the _main method:
    - Dialog.AddEntryDialog_main()
- This code is a good starting point…
- …but we can make improvements

```
/*-------------------------------------------------------------------------*\
 |  Module Details:
 |  Name:    AddEntryDialog.java
 |  Purpose: Display the AddEntryDialog dialog
\*-------------------------------------------------------------------------*/
package org.ptcuser.creo.parametric.ui ;

import com.ptc.uifc.uifcComponent.*;

public class AddEntryDialog
{
  private String addentrydialog_dialog = "AddEntryDialog";
  private String addentrydialog_pbnok = "pbnOK";
  private String addentrydialog_pbncancel = "pbnCancel";
  private String addentrydialog_inpname = "inpName";
  private String addentrydialog_inpdescription = "inpDescription";

  class AddEntryDialogDialogListener extends DefaultDialogListener

  class pbnOKPushButtonListener extends DefaultPushButtonListener

  class pbnCancelPushButtonListener extends DefaultPushButtonListener

  class inpNameInputPanelListener extends DefaultInputPanelListener

  class inpDescriptionInputPanelListener extends DefaultInputPanelListener

  public void AddEntryDialog_main()
  {
    try
    {
      uifcComponent.CreateDialog(addentrydialog_dialog, addentrydialog_dialog);

      Dialog addentrydialog = uifcDialog.DialogFind(addentrydialog_dialog, addentrydialog_dialog);
      AddEntryDialogDialogListener addentrydialogLis = new AddEntryDialogDialogListener();
      addentrydialog.AddActionListener(addentrydialogLis);

      PushButton pbnok = uifcPushButton.PushButtonFind(addentrydialog_dialog, addentrydialog_pbnok);
      pbnOKPushButtonListener pbnokLis = new pbnOKPushButtonListener();
      pbnok.AddActionListener(pbnokLis);

      PushButton pbncancel = uifcPushButton.PushButtonFind(addentrydialog_dialog, addentrydialog_pbncancel);
      pbnCancelPushButtonListener pbncancelLis = new pbnCancelPushButtonListener();
      pbncancel.AddActionListener(pbncancelLis);

      InputPanel inpname = uifcInputPanel.InputPanelFind(addentrydialog_dialog, addentrydialog_inpname);
      inpNameInputPanelListener inpnameLis = new inpNameInputPanelListener();
      inpname.AddActionListener(inpnameLis);

      InputPanel inpdescription = uifcInputPanel.InputPanelFind(addentrydialog_dialog, addentrydialog_inpdescription);
      inpDescriptionInputPanelListener inpdescriptionLis = new inpDescriptionInputPanelListener();
      inpdescription.AddActionListener(inpdescriptionLis);

      uifcComponent.ActivateDialog (addentrydialog_dialog);
      uifcComponent.DestroyDialog (addentrydialog_dialog);
    }
    catch (Throwable e) {}
  }
}
```

Revising the AddEntryDialog

PTC® Live Global

After several iterations of changes….

- Use static members
  - You need to display just one AddEntryDialog at time
- Replace strings with components
- Rename AddEntryDialog_main to show
  - Invoke as AddEntryDialog.show()
- Rename ActionListeners
- Use Javadoc comments
  - Creates IDE tooltips
  - Create HTML documentation

```
package org.ptcuser.creo.parametric.ui ;

import java.util.HashMap;

import com.ptc.cipjava.jxthrowable;
import com.ptc.uifc.uifcComponent.* ;
import com.ptc.uifc.uifcDialog.* ;
import com.ptc.uifc.uifcPushButton.* ;
import com.ptc.uifc.uifcInputPanel.* ;

/**
 * Manages a dialog for entering name and description fields.
 * @author Mark Stallard <stallard@raytheon.com>
 */
public class AddEntryDialog {
    /** Displayed Creo dialog handle. */
    private static Dialog dlgAddEntry ;

    /** "OK" button handle. */
    private static PushButton pbnOk ;

    /** "Cancel" button handle. */
    private static PushButton pbnCancel ;

    /** "Name" input field handle. */
    private static InputPanel inpName ;

    /** "Description" input field handle. */
    private static InputPanel inpDesc ;

    /** Holds the returned values. */
    private static HashMap<String,String> entry ;

    /**  Displays this dialog.  Exits when dialog closes.
    public static HashMap<String,String> show() {

    /**  Confirms whether this dialog has text in every single Input Panel.
    private static boolean hasTextInAllFields() throws jxthrowable {

    /**  Inner ActionListener class to support direct dialog closure.
    public static class AddEntryDialogListener extends DefaultDialogListener {

    /**  Inner ActionListener class to support the OK button.
    public static class PbnOkListener extends DefaultPushButtonListener {

    /**  Inner ActionListener class to support the Cancel button.
    public static class PbnCancelListener extends DefaultPushButtonListener {

    /**  Inner ActionListener class to support the Name InputPanel.
    public static class InpNameListener extends DefaultInputPanelListener {

    /**  Inner ActionListener class to support the Description InputPanel.
    public static class InpDescListener extends DefaultInputPanelListener {
}
```


The show Method

PTC® Live Global

- Returns HashMap entry
- Creates internal dialog object
- Saves references to dialog and components
- Activates dialog
  - Dialog displays
  - ActivateDialog exits when dialog closes
- Destroys dialog

```
    /** Holds the returned values. */
    private static HashMap<String,String> entry ;

    /**
     * Displays this dialog.  Exits when dialog closes.
     * @returns Map containing name and description.
     */
    public static HashMap<String,String> show() {

        // Start with an empty set of return values.

        entry = new HashMap<String,String>() ;

        try {
            uifcComponent.CreateDialog( "AddEntryDialog", "AddEntryDialog" ) ;

            // Find each component and attach a new action listener to it.

            dlgAddEntry = uifcDialog.DialogFind( "AddEntryDialog", "AddEntryDialog" ) ;
            dlgAddEntry.AddActionListener( new AddEntryDialogListener() ) ;

            pbnOk = uifcPushButton.PushButtonFind( "AddEntryDialog", "pbnOk" ) ;
            pbnOk.AddActionListener( new PbnOkListener() ) ;

            pbnCancel = uifcPushButton.PushButtonFind( "AddEntryDialog", "pbnCancel" ) ;
            pbnCancel.AddActionListener( new PbnCancelListener() ) ;

            inpName = uifcInputPanel.InputPanelFind( "AddEntryDialog", "inpName" ) ;
            inpName.AddActionListener( new InpNameListener() ) ;

            inpDesc = uifcInputPanel.InputPanelFind( "AddEntryDialog", "inpDesc" ) ;
            inpDesc.AddActionListener( new InpDescListener() ) ;

            // Display the dialog for user input.
            // Code returns from this line after dialog closes.

            uifcComponent.ActivateDialog( "AddEntryDialog" ) ;

            // Dialog has closed. Destroy it now so it can be re-opened later.

            uifcComponent.DestroyDialog( "AddEntryDialog" ) ;
        } catch ( jxthrowable e ) {
            MessageDialog.showError( "Error Managing Dialog", e ) ;
        }

        return entry ;
    }
```

Enabling the OK Button

PTC® Live Global

Requires Text in Both Input Panels

- Input Panel Listeners can Enable or Disable OK
- hasTextInAllFields checks both fields
  - Neither field may be empty
- Result will enable or Disable OK button

```
79  /**
80   * Confirms whether this dialog has text in every single Input Panel.
81   * @return              true if every Input Panel has text in it.
82   * @throws jxthrowable Failure to extract text from an Input Panel.
83   */
84  private static boolean hasTextInAllFields() throws jxthrowable {
85      String name = inpName.GetStringValue() ;
86      String desc = inpDesc.GetStringValue() ;
87
88      // Are both Input Panels NON-empty?
89
90      boolean hasName = ! name.isEmpty() ;
91      boolean hasDesc = ! desc.isEmpty() ;
92
93      return hasName && hasDesc ;
94  }
95
96   * Inner ActionListener class to support direct dialog closure.
100 public static class AddEntryDialogListener extends DefaultDialogListener {
115
117  * Inner ActionListener class to support the OK button.
120 public static class PbnOkListener extends DefaultPushButtonListener {
135
137  * Inner ActionListener class to support the Cancel button.
140 public static class PbnCancelListener extends DefaultPushButtonListener {
154
156  * Inner ActionListener class to support the Name InputPanel.
159 public static class InpNameListener extends DefaultInputPanelListener {
160     /**
161      * Called when user edits this Input Panel.
162      * @param handle Handle to the edited Input Panel.
163      */
164     public void OnActivate( InputPanel handle ) {
165         try {
166             pbnOk.SetEnabled( hasTextInAllFields() ) ;
167         } catch ( jxthrowable e ) {
168             MessageDialog.showError( "Error Reading Input Panel", e ) ;
169         }
170     }
171 }
172
174  * Inner ActionListener class to support the Description InputPanel.
177 public static class InpDescListener extends DefaultInputPanelListener {
178     /**
179      * Called when user edits this Input Panel.
180      * @param handle Handle to the edited Input Panel.
181      */
182     public void OnActivate( InputPanel handle ) {
183         try {
184             pbnOk.SetEnabled( hasTextInAllFields() ) ;
185         } catch ( jxthrowable e ) {
186             MessageDialog.showError( "Error Reading Input Panel", e ) ;
187         }
188     }
189 }
190 }
```


Summary

PTC® Live Global

- Use OTK Java Dialogs to Avoid UI Problems
  - Window Stack Order Issues
  - Inconsistent Look and Feel
- Write Reusable Modules
  - Wrapper for Message Dialogs
  - Test Program as Skeleton
- Use the PTC Creo UI Editor
  - But also have a good text editor
  - Use Template Dialogs
- Name UI Components Carefully
  - Show both type and purpose
  - Suggest component relationships

- Your feedback is valuable
- Don’t miss out on the chance to provide your feedback
- Gain a chance to win an instant price!
- Complete your session evaluation now

PTC® Live Global

PTC® PRODUCT & SERVICE ADVANTAGE®