

# PTC® Live Global

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

**Mark Stallard**

Software Developer  
Raytheon Company  
stallard@raytheon.com

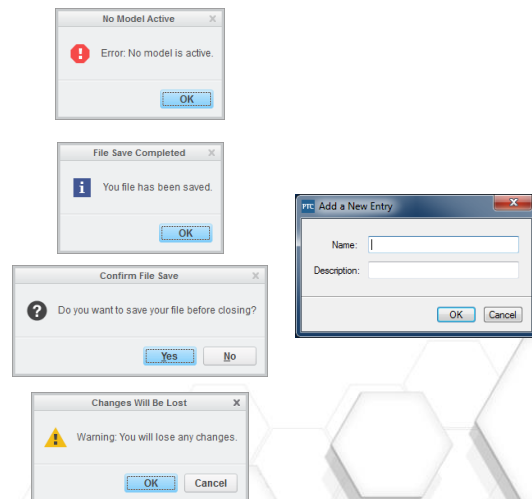
Tuesday, June 9, 2015



### Topics Presented

PTC® Live  
Global

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



- Integrated Development Environment (IDE)

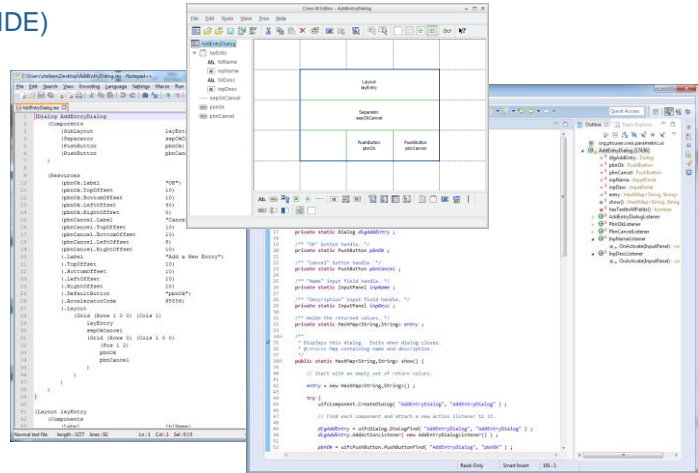
- Eclipse for Java
- Visual Studio for C++

- 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



3

# Why Use PTC UI with Java?

Java already has multiple UI frameworks, such as Swing

- OTK Java Maintains Creo Look & Feel

- Creo UI Editor Generates Code

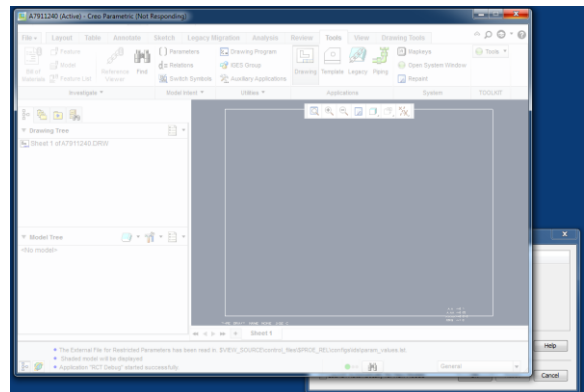
- This will get you started
- Less Work than Swing

- 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

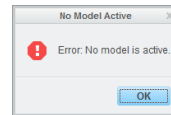
- Recent Example:
  - Display Web Page from Jlink
  - Needed Creo Message Dialog
  - Stack Issues Returned



4

## 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**



```

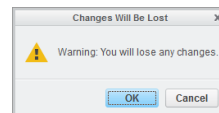
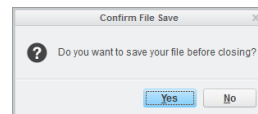
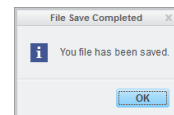
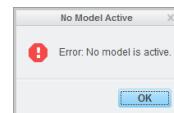
31 // Need to report an error to the user.
32
33 String title = "No Model Active";
34 String message = "Error: No model is active.";
35
36 try {
37     // Create dialog options with an OK button and an Error icon.
38     MessageDialogOptions options = pfcUI.MessageDialogOptions_create();
39     MessageButtons buttons = MessageButtons_create();
40     buttons.insert( 0, MessageButton.MESSAGE_BUTTON_OK );
41     options.SetButtons( buttons );
42     options.SetMessageDialogType( MessageDialogType.MESSAGE_ERROR );
43
44     // Add the dialog title
45     // Show the dialog
46
47     options.SetDialogLabel( title );
48     Session session = pfcGlobal.GetProSession();
49     session.UIShowMessageDialog( message, options );
50 } catch ( jthrowable ex ) {
51
52     // Can't display a dialog? Now we're really in trouble!
53
54     StringBuilder sb = new StringBuilder( "Error: can't display error dialog.\n" );
55     sb.append( "title: " ).append( title ).append( "\n" );
56     sb.append( "Message: " ).append( message ).append( "\n" );
57     System.out.println( sb.toString() );
58 }
59
60

```

# Designing a MessageDialog Class

## 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



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 ;
4
5 /**
6  * Collection of static methods for quickly displaying Creo
7  * MessageDialogs.
8  * @author Mark Stallard
9  */
10 public class MessageDialog {
11
12     /**
13      * Shows a Creo error dialog with an "OK" button.
14      * @param title Title bar text.
15      * @param message Error message to show in dialog body.
16      */
17     public static void showError( String title, String message ) {}
18
19     /**
20      * Shows a Creo error dialog with an "OK" button.
21      * @param title Title bar text.
22      * @param exception Exception with message to show in dialog body.
23      */
24     public static void showError( String title, jxthrowable exception ) {}
25
26     /**
27      * Shows a Creo information dialog with an "OK" button.
28      * @param title Title bar text.
29      * @param message Message to show in dialog body.
30      */
31     public static void showInfo( String title, String message ) {}
32
33     /**
34      * Shows a Creo question dialog with "Yes" and "No" buttons.
35      * @param title Title bar text.
36      * @param message Question to show in dialog body.
37      * @return true if user clicked "Yes".
38      */
39     public static boolean showQuestion( String title, String message ) {}
40
41     /**
42      * Shows a Creo question dialog with "OK" and "Cancel" buttons.
43      * @param title Title bar text.
44      * @param message Warning to show in dialog body.
45      * @return true if user clicked "OK".
46      */
47     public static boolean showWarning( String title, String message ) {}
48 }
49

```

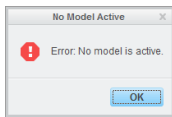
Here's the expanded code for method *showError*

- You can write your own message
- Or show an exception's message:

```

110     MessageDialog.showError( "No Model Active",
111         "Error: No model is active." );
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```



```

180     /**
181      * Shows a Creo error dialog with an "OK" button.
182      * @param title Title bar text.
183      * @param message Error message to show in dialog body.
184      */
185     public static void showError( String title, String message ) {
186         try {
187             // Create dialog options with an OK button and an Error icon.
188             MessageDialogOptions options = pfUI.MessageDialogOptions_create();
189             MessageButtons buttons = MessageButtons_create();
190             buttons.insert( 0, MessageButton.MESSAGE_BUTTON_OK );
191             options.SetButtons( buttons );
192             options.SetMessageDialogType( MessageDialogType.MESSAGE_ERROR );
193
194             // Add the dialog title
195             // Show the dialog
196             options.SetDialogTitle( title );
197             Session session = pfGlobal.GetProSession();
198             session.UIShowMessageDialog( message, options );
199         } catch ( jxthrowable ex ) {
200             StringBuilder sb = new StringBuilder( "Error: can't display error dialog.\n" );
201             sb.append( "Title: " ).append( title ).append( "\n" );
202             sb.append( "Message: " ).append( message ).append( "\n" );
203             System.out.println( sb.toString() );
204         }
205     }
206
207     /**
208      * Shows a Creo error dialog with an "OK" button.
209      * @param title Title bar text.
210      * @param exception Exception with message to show in dialog body.
211      */
212     public static void showError( String title, jxthrowable exception ) {
213         StringBuilder sb = new StringBuilder( "Exception caught: " );
214         sb.append( exception.toString() );
215         showError( title, sb.toString() );
216     }
217

```

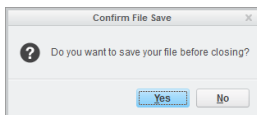
Here's the expanded code for method `showQuestion`

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

```

131     boolean wantSave = MessageDialog.showQuestion( "Confirm File Save",
132         "Do you want to save your file before closing?" );
133
134     if ( wantSave ) {
135         MessageDialog.showInfo( "Saving File",
136             "File will be saved." );
137     } else {
138         MessageDialog.showInfo( "Discarding Changes",
139             "Changes will be discarded." );
140     }
141
142
143
144
145

```



```

87    /**
88     * Shows a Creo question dialog with "Yes" and "No" buttons.
89     * @param title Title bar text.
90     * @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 ) {
94        boolean wasYesClicked = false;
95
96        try {
97            //Create dialog options with Yes/No buttons and a Question icon.
98
99            MessageDialogOptions options = pfUI.MessageDialogOptions_Create();
100            MessageButtons buttons = MessageButtons_create();
101            buttons.insert( 0, MessageButton.MESSAGE_BUTTON_YES );
102            buttons.insert( 0, MessageButton.MESSAGE_BUTTON_NO );
103            options.SetButtons( buttons );
104            options.SetMessageDialogType( MessageDialogType.MESSAGE_QUESTION );
105
106            // Add the dialog title
107            // Show the dialog
108
109            options.SetDialogTitle( title );
110            Session session = pfGlobal.getProSession();
111            MessageButton clickedButton = session.UIShowMessageDialog( message, options );
112
113            // Determine which button the user clicked.
114
115            wasYesClicked = ( clickedButton == MessageButton.MESSAGE_BUTTON_YES );
116        } catch ( Throwable ex ) {
117            StringBuilder sb = new StringBuilder( "Error: can't display information dialog.\n" );
118            sb.append( "Title: " ).append( title ).append( "\n" );
119            sb.append( "Message: " ).append( message ).append( "\n" );
120            System.out.println( sb.toString() );
121        }
122
123        return wasYesClicked;
124    }
125
126
127
128
129

```

## A C++ MessageDialog Class

I wrote this before I had Object Toolkit C++

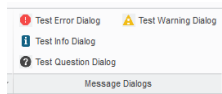
- C++ class is `Pro::UI::MessageDialog`
  - Namespace `Pro::UI`
  - Class `MessageDialog`
- Method `Pro::UI::MessageDialog::Display`
  - Wraps Pro/Toolkit method `ProUIMessageDialogDisplay`
  - Throws exceptions if error found
    - Subclasses of `Pro::Tk::Exception`
    - Wrote these exceptions myself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

```

13 // Displays the UI Message Dialog
14
15 ProUIMessageDialog Pro::UI::MessageDialog::Display( ProUIMessageType type, wstring title_cpp,
16     wstring msg_txt_cpp, vector<ProUIMessageButton> buttons_cpp, ProUIMessageButton def_button ) {
17     ProUIMessageButton *buttons = Pro::Array::FromVector( buttons_cpp );
18     ProUIMessageButton user_choice;
19     ProError error = ProUIMessageDialogDisplay( type, ( wchar_t * ) title_cpp_c_str(),
20         ( wchar_t * ) msg_txt_cpp_c_str(), buttons, def_button, &user_choice );
21
22     switch ( error ) {
23     case PRO_TK_NO_ERROR :
24         break;
25     case PRO_TK_BAD_INPUTS :
26         throw Pro::Tk::Bad_Inputs( L"ProUIMessageDialogDisplay",
27             L"Bad inputs" );
28     case PRO_TK_GENERAL_ERROR :
29         throw Pro::Tk::General_Error( L"ProUIMessageDialogDisplay",
30             L"The function failed <!-- Objectname: ProUI -->" );
31     default :
32         throw Pro::Tk::UndocumentedException( L"ProUIMessageDialogDisplay",
33             L"An undocumented error occurred." );
34     }
35
36     return user_choice;
37 }
38
39 // Displays an info dialog with a single "OK" button.
40
41 void Pro::UI::MessageDialog::DisplayOk( wstring title_cpp, wstring msg_txt_cpp ) {
42     vector<ProUIMessageButton> btn_list;
43     btn_list.push_back( PRO_UI_MESSAGE_OK );
44
45     Pro::UI::MessageDialog::Display( PROUIMESSAGE_INFO, title_cpp, msg_txt_cpp,
46         btn_list, PRO_UI_MESSAGE_OK );
47 }
48
49 // Displays an info dialog with a single "OK" button.
50
51 void Pro::UI::MessageDialog::DisplayOk( string title_cpp, string msg_txt_cpp ) {
52     DisplayOk( wstring( title_cpp.begin(), title_cpp.end() ),
53         wstring( msg_txt_cpp.begin(), msg_txt_cpp.end() ) );
54 }
55
56

```

- 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`

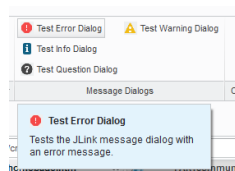


```

13@/**
14 * JLink application to test Creo message dialogs.
15 * @author: Mark Stallard
16 */
17 public class MessageDialogTest {
18     /**
19     * Called by Creo when application launches.
20     */
21     public static void start() {
22         createCommands();
23     }
24
25     /**
26     * Called by Creo when application terminates.
27     */
28     public static void stop() {
29     }
30
31     /**
32     * Creates commands invoked by the ribbon UI.
33     */
34     private static void createCommands() {}
35
36     /**
37     * Inner class for handling the "Test Error Dialog" command.
38     */
39     static class TestErrorDialogListener extends DefaultUICommandActionListener {}
40
41     /**
42     * Inner class for handling the "Test Info Dialog" command.
43     */
44     static class TestInfoDialogListener extends DefaultUICommandActionListener {}
45
46     /**
47     * Inner class for handling the "Test Question Dialog" command.
48     */
49     static class TestQuestionDialogListener extends DefaultUICommandActionListener {}
50
51     /**
52     * Inner class for handling the "Test Warning Dialog" command.
53     */
54     static class TestWarningDialogListener extends DefaultUICommandActionListener {}
55 }

```

- 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



```

31@/**
32 * Creates commands invoked by the ribbon UI.
33 */
34 private static void createCommands() {
35     try {
36         // Load the ribbon definition file.
37         Session session = pfcGlobal.GetProfSession();
38         session.RibbonDefinitionFileLoad( "messagedialogs_how.rbn" ); // Disable until you create .rbn file
39
40         // Create Test Error Dialog Command.
41
42         UICommandActionListener lntestErrorDialog = new TestErrorDialogListener();
43         UICommand cmdtestErrorDialog = session.UICreateCommand( "Test Error Dialog", lntestErrorDialog );
44         cmdtestErrorDialog.SetIcon( "Error.png" );
45         cmdtestErrorDialog.Designate( "msg_messagedialog.txt",
46             "MessageDialog Test Error -label-",
47             "MessageDialog Test Error -help-",
48             "MessageDialog Test Error -desc-" );
49
50         // Create Test Info Dialog Command.
51
52         UICommandActionListener lntestInfoDialog = new TestInfoDialogListener();
53         UICommand cmdtestInfoDialog = session.UICreateCommand( "Test Info Dialog", lntestInfoDialog );
54         cmdtestInfoDialog.SetIcon( "Info.png" );
55         cmdtestInfoDialog.Designate( "msg_messagedialog.txt",
56             "MessageDialog Test Info -label-",
57             "MessageDialog Test Info -help-",
58             "MessageDialog Test Info -desc-" );
59
60         // Create Test Question Dialog Command.
61
62         UICommandActionListener lntestQuestionDialog = new TestQuestionDialogListener();
63         UICommand cmdtestQuestionDialog = session.UICreateCommand( "Test Question Dialog", lntestQuestionDialog );
64         cmdtestQuestionDialog.SetIcon( "Question.png" );
65         cmdtestQuestionDialog.Designate( "msg_messagedialog.txt",
66             "MessageDialog Test Question -label-",
67             "MessageDialog Test Question -help-",
68             "MessageDialog Test Question -desc-" );
69
70         // Create Test Warning Dialog Command.
71
72         UICommandActionListener lntestWarningDialog = new TestWarningDialogListener();
73         UICommand cmdtestWarningDialog = session.UICreateCommand( "Test Warning Dialog", lntestWarningDialog );
74         cmdtestWarningDialog.SetIcon( "Warning.png" );
75         cmdtestWarningDialog.Designate( "msg_messagedialog.txt",
76             "MessageDialog Test Warning -label-",
77             "MessageDialog Test Warning -help-",
78             "MessageDialog Test Warning -desc-" );
79
80     } catch ( Throwable ex ) {
81         MessageDialog.showError( "Command Setup Failed", "Can't create commands for Creo UI." );
82     }
83 }

```

## 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

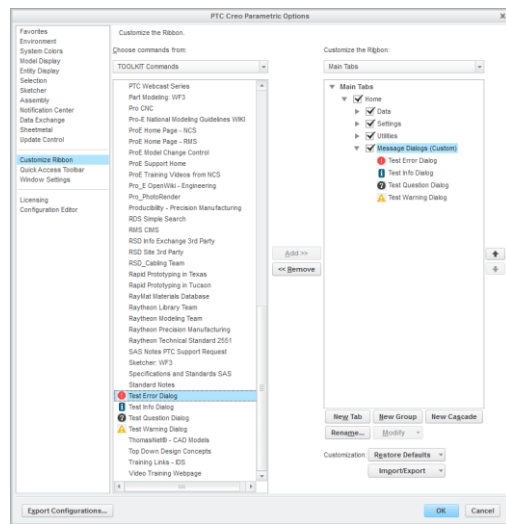
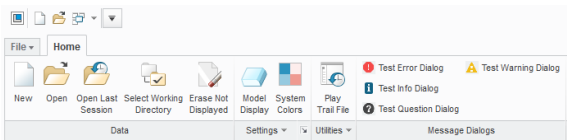
```

130 /**
14  * Link application to test CCG message dialogs.
15  * @author Mark Stallard
16  */
17 public class MessageDialogTest {
18     /**
19      * Called by CCG when application launches.
20      */
21     public static void start() {
22         createCommands();
23     }
24
25     /**
26      * Called by CCG when application terminates.
27      */
28     public static void stop() {
29     }
30
31     /**
32      * Creates commands invoked by the ribbon UI.
33      */
34     private static void createCommands() {
35
36         /**
37          * Inner class for handling the "Test Error Dialog" command.
38          */
39         static class TestErrorDialogListener extends DefaultUICommandActionListener {
40             public void onClick() {
41                 MessageDialog.showError( "No Model Active",
42                     "Error: No model is active." );
43             }
44         }
45
46         /**
47          * Inner class for handling the "Test Info Dialog" command.
48          */
49         static class TestInfoDialogListener extends DefaultUICommandActionListener {
50
51         }
52
53         /**
54          * Inner class for handling the "Test Question Dialog" command.
55          */
56         static class TestQuestionDialogListener extends DefaultUICommandActionListener {
57
58         }
59
60         /**
61          * Inner class for handling the "Test Warning Dialog" command.
62          */
63         static class TestWarningDialogListener extends DefaultUICommandActionListener {
64
65         }
66     }
67 }

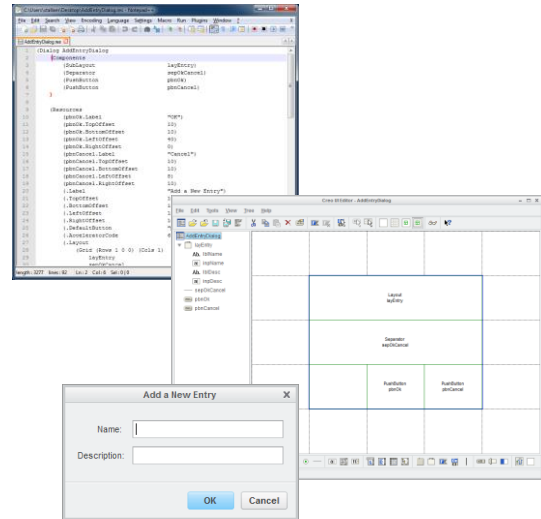
```

# Notes on the Ribbon

- 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



- **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
    - OptionMenu.VisibleRows vs. DropDownHeight
  - Entries for Non-Existent Attributes
  - Can't Re-Parent or Re-Order Components

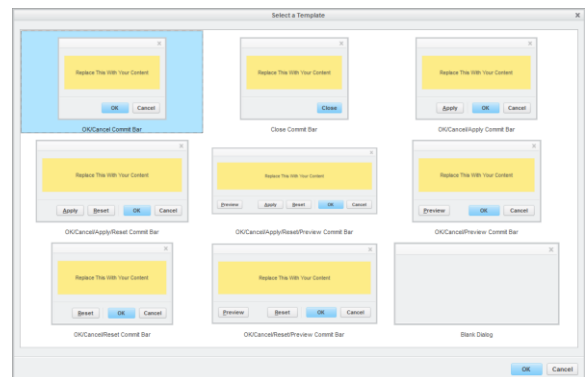


Customizing PTC Creo Parametric User Interfaces in Java and C++ | Mark Stallard | Raytheon Company

## Template Dialogs

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

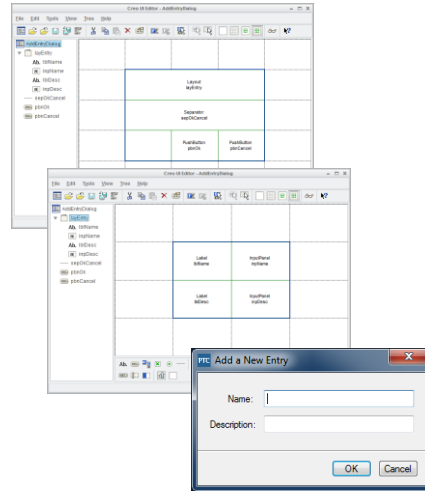


Customizing PTC Creo Parametric User Interfaces in Java and C++ | Mark Stallard | Raytheon Company



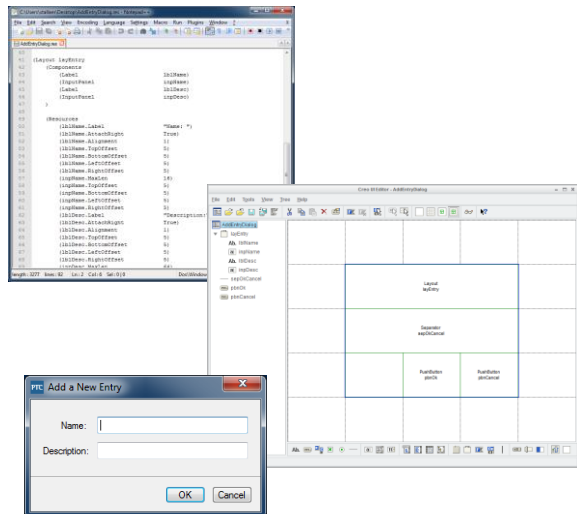
Using the Creo UI Editor

- 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?



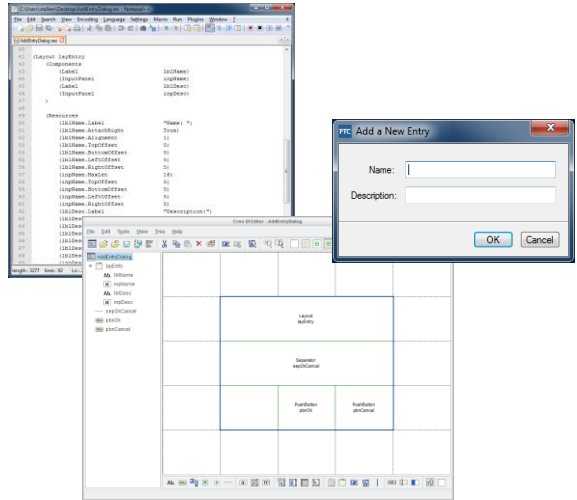
Good Naming Helps Code Organization

- 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**



## 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

## 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

```
16 .....*/
17 | Package Details:
18 | Name: AddEntryDialog.java
19 | Purpose: Display the AddEntryDialog dialog
20 | .....*/
21 | Package org.ptcuser.creo.parametric.ui
22 | Import com.ptc.uifc.uifComponent.*;
23 |
24 | public class AddEntryDialog
25 | {
26 |     private String addentrydialog_dialog = "AddEntryDialog";
27 |     private String addentrydialog_pbok = "PbOk";
28 |     private String addentrydialog_pbcancel = "PbCancel";
29 |     private String addentrydialog_inpname = "InpName";
30 |     private String addentrydialog_inpdescription = "InpDescription";
31 |
32 |     class AddEntryDialogDialogListener extends DefaultDialogListener
33 |     {
34 |         class PbOkPushButtonListener extends DefaultPushButtonListener
35 |         {
36 |         }
37 |         class PbCancelPushButtonListener extends DefaultPushButtonListener
38 |         {
39 |         }
40 |         class InpNameInputPanelListener extends DefaultInputPanelListener
41 |         {
42 |         }
43 |         class InpDescriptionInputPanelListener extends DefaultInputPanelListener
44 |         {
45 |         }
46 |
47 |     }
48 |
49 |     public void AddEntryDialog_main()
50 |     {
51 |     }
52 |
53 |     try
54 |     {
55 |         uifComponent.CreateDialog(addentrydialog_dialog, addentrydialog_dialog);
56 |         Dialog addentrydialog = uifDialog.DialogFind(addentrydialog_dialog, addentrydialog_dialog);
57 |         AddEntryDialogDialogListener addentrydialogli = new AddEntryDialogDialogListener();
58 |         addentrydialog.AddActionListener(addentrydialogli);
59 |
60 |         PushButton pbok = uifPushButton.PushButtonFind(addentrydialog_dialog, addentrydialog_pbok);
61 |         pbok.AddActionListener(new PbOkPushButtonListener());
62 |         pbok.AddActionListener(addentrydialogli);
63 |
64 |         PushButton pbcancel = uifPushButton.PushButtonFind(addentrydialog_dialog, addentrydialog_pbcancel);
65 |         pbcancel.AddActionListener(new PbCancelPushButtonListener());
66 |         pbcancel.AddActionListener(addentrydialogli);
67 |
68 |         InputPanel inpname = uifInputPanel.InputPanelFind(addentrydialog_dialog, addentrydialog_inpname);
69 |         inpname.AddActionListener(new InpNameInputPanelListener());
70 |         inpname.AddActionListener(addentrydialogli);
71 |
72 |         InputPanel inpdescription = uifInputPanel.InputPanelFind(addentrydialog_dialog, addentrydialog_inpdescription);
73 |         inpdescription.AddActionListener(new InpDescriptionInputPanelListener());
74 |         inpdescription.AddActionListener(addentrydialogli);
75 |
76 |         uifComponent.DestroyDialog(addentrydialog_dialog);
77 |
78 |     } catch (Throwable e) {}
79 | }
80 | }
```

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

```

1 package org.ptcuser.creo.parametric.ui ;
2
3 import java.util.HashMap;
4
5 import com.ptc.cipjava.jthrowable;
6 import com.ptc.uifc.uifComponent;
7 import com.ptc.uifc.uifDialog;
8 import com.ptc.uifc.uifPushButton;
9 import com.ptc.uifc.uifInputPanel;
10
11 /**
12  * Manages a dialog for entering name and description fields.
13  * @author Mark Stallard <stallard@raytheon.com>
14  */
15 public class AddEntryDialog {
16     /** Displayed Creo dialog handle. */
17     private static Dialog dlgAddEntry;
18
19     /** "OK" button handle. */
20     private static PushButton pbmOK;
21
22     /** "Cancel" button handle. */
23     private static PushButton pbmCancel;
24
25     /** "Name" input field handle. */
26     private static InputPanel inpName;
27
28     /** "Description" input field handle. */
29     private static InputPanel inpDesc;
30
31     /** Holds the returned values. */
32     private static HashMap<String,String> entry;
33
34     /** Displays this dialog. Exits when dialog closes. */
35     public static HashMap<String,String> show() {
36
37         /** Confirms whether this dialog has text in every single Input Panel. */
38         private static boolean hasTextInAllFields() throws jthrowable {
39
40             /** Inner ActionListener class to support direct dialog closure. */
41             public static class AddEntryDialogListener extends DefaultDialogListener {
42
43                 /** Inner ActionListener class to support the OK button. */
44                 public static class PbmOkListener extends DefaultPushButtonListener {
45
46                     /** Inner ActionListener class to support the Cancel button. */
47                     public static class PbmCancelListener extends DefaultPushButtonListener {
48
49                         /** Inner ActionListener class to support the Name InputPanel. */
50                         public static class InpNameListener extends DefaultInputPanelListener {
51
52                             /** Inner ActionListener class to support the Description InputPanel. */
53                             public static class InpDescListener extends DefaultInputPanelListener {
54
55                             }
56                         }
57                     }
58                 }
59             }
60         }
61     }
62 }

```

## The show Method

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

```

31 /** Holds the returned values. */
32 private static HashMap<String,String> entry;
33
34 /**
35  * Displays this dialog. Exits when dialog closes.
36  * @returns Map containing name and description.
37  */
38 public static HashMap<String,String> show() {
39
40     // Start with an empty set of return values.
41     entry = new HashMap<String,String>();
42
43     try {
44         uifComponent.CreateDialog( "AddEntryDialog", "AddEntryDialog" );
45
46         // Find each component and attach a new action listener to it.
47         dlgAddEntry = uifDialog.DialogFind( "AddEntryDialog", "AddEntryDialog" );
48         dlgAddEntry.AddActionListener( new AddEntryDialogListener() );
49
50         pbmOK = uifPushButton.PushButtonFind( "AddEntryDialog", "pbmOK" );
51         pbmOK.AddActionListener( new PbmOkListener() );
52
53         pbmCancel = uifPushButton.PushButtonFind( "AddEntryDialog", "pbmCancel" );
54         pbmCancel.AddActionListener( new PbmCancelListener() );
55
56         inpName = uifInputPanel.InputPanelFind( "AddEntryDialog", "inpName" );
57         inpName.AddActionListener( new InpNameListener() );
58
59         inpDesc = uifInputPanel.InputPanelFind( "AddEntryDialog", "inpDesc" );
60         inpDesc.AddActionListener( new InpDescListener() );
61
62         // Display the dialog for user input.
63         // Code returns from this line after dialog closes.
64         uifComponent.ActivateDialog( "AddEntryDialog" );
65
66         // Dialog has closed. Destroy it now so it can be re-opened later.
67         uifComponent.DestroyDialog( "AddEntryDialog" );
68     } catch ( jthrowable e ) {
69         MessageDialog.showError( "Error Managing Dialog", e );
70     }
71     return entry;
72 }
73
74 }
75
76 }
77
78 }

```

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

```

790  /**
791  * Confirms whether this dialog has text in every single Input Panel.
792  * @return true if every Input Panel has text in it.
793  * @throws JThrowable failure to extract text from an Input Panel.
794  */
795  private static boolean hasTextInAllFields() throws JThrowable {
796      String name = InputDesc.GetStringValue();
797      String desc = InputDesc.GetStringValue();
798      // Are both Input Panels NON-empty?
799      boolean hasName = ! name.isEmpty();
800      boolean hasDesc = ! desc.isEmpty();
801      return hasName && hasDesc;
802  }
803
804  * Inner ActionListener class to support direct dialog closure.[]
805  public static class AddEntryDialogListener extends DefaultDialogListener {}
806
807  * Inner ActionListener class to support the OK button.[]
808  public static class PbnOkListener extends DefaultPushButtonListener {}
809
810  * Inner ActionListener class to support the Cancel button.[]
811  public static class PbnCancelListener extends DefaultPushButtonListener {}
812
813  * Inner ActionListener class to support the Name InputPanel.[]
814  public static class InputNameListener extends DefaultInputPanelListener {
815      /**
816       * Called when user edits this Input Panel.
817       * @param handle Handle to the edited Input Panel.
818       */
819      public void OnActivate( InputPanel handle ) {
820          try {
821              PbnOk.SetEnabled( hasTextInAllFields() );
822          } catch ( JThrowable e ) {
823              e.printStackTrace();
824              MessageDialog.showError( "Error Reading Input Panel", e );
825          }
826      }
827  }
828
829  * Inner ActionListener class to support the Description InputPanel.[]
830  public static class InputDescListener extends DefaultInputPanelListener {
831      /**
832       * Called when user edits this Input Panel.
833       * @param handle Handle to the edited Input Panel.
834       */
835      public void OnActivate( InputPanel handle ) {
836          try {
837              PbnOk.SetEnabled( hasTextInAllFields() );
838          } catch ( JThrowable e ) {
839              e.printStackTrace();
840              MessageDialog.showError( "Error Reading Input Panel", e );
841          }
842      }
843  }
844  }

```

- 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