Popups:

Example plugin.xml

<?xml version="1.0" encoding="UTF-8"?>

<?eclipse version="3.4"?>

<plugin>

<extension

point="org.eclipse.ui.popupMenus">

<objectContribution

objectClass="org.eclipse.ui.IEditorInput"

nameFilter="\*.\*"

id="tamajit.demo.popup.contribution1">

<menu

label="New Submenu"

path="additions"

id="tamajit.demo.popup.menu1">

<separator

name="group1">

</separator>

</menu>

<action

label="New Action"

class="tamajit.demo.popup.popup.actions.NewAction"

menubarPath="tamajit.demo.popup.menu1/group1"

enablesFor="1"

id="tamajit.demo.popup.newAction">

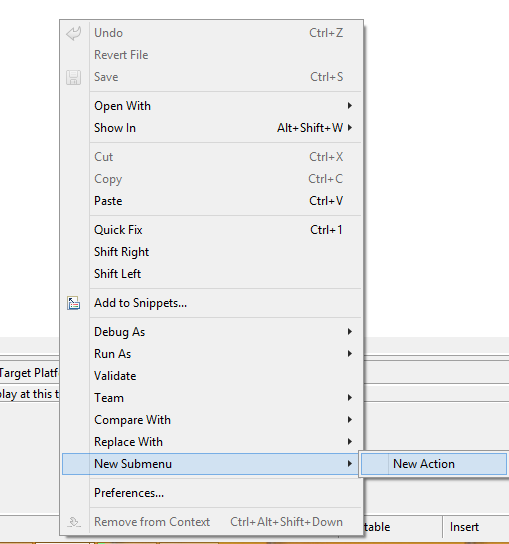
</action>

</objectContribution>

</extension>

</plugin>

Here NewAction **implements** IObjectActionDelegate and overwritten method run() acts as handler. Snapshot is given below.



Examples1:

We can add popup in editor for all \*.java file using the following configuration .

We can also use \*.\*

<objectContribution

objectClass="org.eclipse.ui.IEditorInput"

nameFilter="\*.java"

id="tamajit.demo.popup.contribution1">

Example2:

We can add all \*.xml file using the following config from project explorer or package explorer. \*.java will only work from Navigator.

<objectContribution

objectClass="org.eclipse.core.resources.IFile"

nameFilter="\*.xml"

id="tamajit.demo.popup.contribution1">

Example3:

You can select any folder from navigator.

<objectContribution

objectClass="org.eclipse.core.resources.IFolder"…

Example4:

You can select \*.java from package explorer and project explorer

<objectContribution

objectClass="org.eclipse.jdt.core.ICompilationUnit"

Example5:

Select any project from project explorer and navigator.

<objectContribution

objectClass="org.eclipse.core.resources.IProject"

**How do I load and save plug-in preferences?**

private void savePluginSettings() {

// saves plugin preferences at the workspace level

Preferences prefs =

InstanceScope.INSTANCE.getNode(MY\_PLUGIN\_ID); // does all the above behind the scenes

prefs.put(KEY1, this.someStr);

prefs.put(KEY2, this.someBool);

try {

// prefs are automatically flushed during a plugin's "super.stop()".

prefs.flush();

} catch(BackingStoreException e) {

//TODO write a real exception handler.

e.printStackTrace();

}

}

private void loadPluginSettings() {

Preferences prefs = new InstanceScope().getNode(MY\_PLUGIN\_ID);

// you might want to call prefs.sync() if you're worried about others changing your settings

this.someStr = prefs.get(KEY1);

this.someBool= prefs.getBoolean(KEY2);

}

### **Creating custom dialogs**

import org.eclipse.jface.dialogs.Dialog;

import org.eclipse.swt.SWT;

import org.eclipse.swt.events.SelectionAdapter;

import org.eclipse.swt.events.SelectionEvent;

import org.eclipse.swt.graphics.Point;

import org.eclipse.swt.layout.GridData;

import org.eclipse.swt.widgets.Button;

import org.eclipse.swt.widgets.Composite;

import org.eclipse.swt.widgets.Control;

import org.eclipse.swt.widgets.Shell;

public class MyDialog extends Dialog {

public MyDialog(Shell parentShell) {

super(parentShell);

}

*@Override*

protected Control createDialogArea(Composite parent) {

Composite container = (Composite) super.createDialogArea(parent);

Button button = new Button(container, SWT.PUSH);

button.setLayoutData(new GridData(SWT.BEGINNING, SWT.CENTER, false,

false));

button.setText("Press me");

button.addSelectionListener(new SelectionAdapter() {

*@Override*

public void widgetSelected(SelectionEvent e) {

System.out.println("Pressed");

}

});

return container;

}

// overriding this methods allows you to set the

// title of the custom dialog

*@Override*

protected void configureShell(Shell newShell) {

super.configureShell(newShell);

newShell.setText("Selection dialog");

}

*@Override*

protected Point getInitialSize() {

return new Point(450, 300);

}

}

**Custom dialog which accepts input :**

**import** org.eclipse.swt.\*;

**import** org.eclipse.swt.events.\*;

**import** org.eclipse.swt.layout.\*;

**import** org.eclipse.swt.widgets.\*;

/\*\*

\* This class demonstrates how to create your own dialog classes. It allows users

\* to input a String

\*/

**public** **class** InputDialog **extends** Dialog {

**private** String message;

**private** String input;

/\*\*

\* InputDialog constructor

\*

\* **@param** parent the parent

\*/

**public** InputDialog(Shell parent) {

// Pass the default styles here

**this**(parent, SWT.***DIALOG\_TRIM*** | SWT.***APPLICATION\_MODAL***);

}

/\*\*

\* InputDialog constructor

\*

\* **@param** parent the parent

\* **@param** style the style

\*/

**public** InputDialog(Shell parent, **int** style) {

// Let users override the default styles

**super**(parent, style);

setText("Input Dialog");

setMessage("Please enter a value:");

}

/\*\*

\* Gets the message

\*

\* **@return** String

\*/

**public** String getMessage() {

**return** message;

}

/\*\*

\* Sets the message

\*

\* **@param** message the new message

\*/

**public** **void** setMessage(String message) {

**this**.message = message;

}

/\*\*

\* Gets the input

\*

\* **@return** String

\*/

**public** String getInput() {

**return** input;

}

/\*\*

\* Sets the input

\*

\* **@param** input the new input

\*/

**public** **void** setInput(String input) {

**this**.input = input;

}

/\*\*

\* Opens the dialog and returns the input

\*

\* **@return** String

\*/

**public** String open() {

// Create the dialog window

Shell shell = **new** Shell(getParent(), getStyle());

shell.setText(getText());

createContents(shell);

shell.pack();

shell.open();

Display display = getParent().getDisplay();

**while** (!shell.isDisposed()) {

**if** (!display.readAndDispatch()) {

display.sleep();

}

}

// Return the entered value, or null

**return** input;

}

/\*\*

\* Creates the dialog's contents

\*

\* **@param** shell the dialog window

\*/

**private** **void** createContents(**final** Shell shell) {

shell.setLayout(**new** GridLayout(2, **true**));

// Show the message

Label label = **new** Label(shell, SWT.***NONE***);

label.setText(message);

GridData data = **new** GridData();

data.horizontalSpan = 2;

label.setLayoutData(data);

// Display the input box

**final** Text text = **new** Text(shell, SWT.***BORDER***);

data = **new** GridData(GridData.***FILL\_HORIZONTAL***);

data.horizontalSpan = 2;

text.setLayoutData(data);

// Create the OK button and add a handler

// so that pressing it will set input

// to the entered value

Button ok = **new** Button(shell, SWT.***PUSH***);

ok.setText("OK");

data = **new** GridData(GridData.***FILL\_HORIZONTAL***);

ok.setLayoutData(data);

ok.addSelectionListener(**new** SelectionAdapter() {

**public** **void** widgetSelected(SelectionEvent event) {

input = text.getText();

shell.close();

}

});

// Create the cancel button and add a handler

// so that pressing it will set input to null

Button cancel = **new** Button(shell, SWT.***PUSH***);

cancel.setText("Cancel");

data = **new** GridData(GridData.***FILL\_HORIZONTAL***);

cancel.setLayoutData(data);

cancel.addSelectionListener(**new** SelectionAdapter() {

**public** **void** widgetSelected(SelectionEvent event) {

input = **null**;

shell.close();

}

});

// Set the OK button as the default, so

// user can type input and press Enter

// to dismiss

shell.setDefaultButton(ok);

}

}

# [How do I get the current method from the active Eclipse editor?](http://stackoverflow.com/questions/9667615/how-do-i-get-the-current-method-from-the-active-eclipse-editor)

IWorkbenchPage page = PlatformUI.getWorkbench().getActiveWorkbenchWindow().getActivePage();

ITextEditor editor = (ITextEditor) page.getActiveEditor();

IJavaElement elem = JavaUI.getEditorInputJavaElement(editor.getEditorInput());

if (elem instanceof ICompilationUnit) {

ITextSelection sel = (ITextSelection) editor.getSelectionProvider().getSelection();

IJavaElement selected = ((ICompilationUnit) elem).getElementAt(sel.getOffset());

if (selected != null && selected.getElementType() == IJavaElement.METHOD) {

return (IMethod) selected;

}

}

return null;

**OR**

IWorkbenchPage page = PlatformUI.getWorkbench().getActiveWorkbenchWindow().getActivePage();

IEditorPart activeEditor = page.getActiveEditor();

if(activeEditor instanceof JavaEditor) {

ITypeRoot root = EditorUtility.getEditorInputJavaElement(this, false);

TextSelection sel = ((TextSelection) activeEditor.getSelection());

IJavaElement elt = root.codeSelect(sel.getOffset(), sel.getLength();

if (elt.getElementType == IJavaElement.METHOD) {

return (IMethod) elt;

}

}

return null;

# [How can I get IEditorPart from ICompilationUnit](http://stackoverflow.com/questions/12638198/how-can-i-get-ieditorpart-from-icompilationunit)

IEditorPart editor = org.eclipse.jdt.internal.ui.javaeditor.EditorUtility.isOpenInEditor(unit);