<http://www.vogella.com/tutorials/EclipseWizards/article.html>

**Eclipse Wizards**

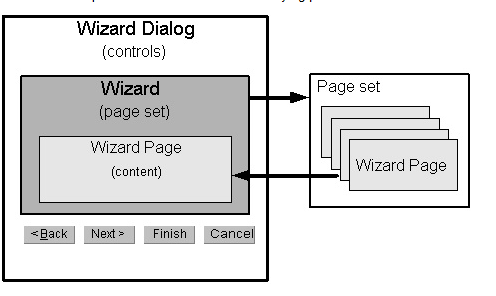
## Wizards

Wizards are used to guide the user through a sequenced set of tasks. Your plug-in can contribute wizards at predefined extension points in the workbench. It can also create and launch its own wizards.

When you contribute to a workbench wizard extension point, the actions that launch the wizard are already set up by the workbench. You need only supply the wizard that will be used.

If you need to launch other wizards that are not already defined in workbench wizard extension points, you must launch them yourself. You can launch your own wizards by adding an action to a view, editor, popup, or an action set.

A wizard is composed of several different underlying parts



### Wizard dialog

The wizard dialog (**[WizardDialog](http://127.0.0.1:50000/help/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/wizard/WizardDialog.html)**) is the top level dialog in a wizard. It defines the standard wizard buttons and manages a set of pages that are provided to it.

When you contribute to a workbench wizard extension, you do not have to create a wizard dialog. One is created on your behalf by the workbench, and your wizard is set into it.

The wizard dialog performs the enabling and disabling of the **Next**, **Back**, and **Finish** buttons based on information it obtains from the wizard and the current wizard page.

### Wizard

The wizard (**[IWizard](http://127.0.0.1:50000/help/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/wizard/IWizard.html)**) controls the overall appearance and behavior of the wizard, such as title bar text, image, and the availability of a help button. Wizards often use a corresponding [**DialogSettings**](http://127.0.0.1:50000/help/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/dialogs/DialogSettings.html) to obtain (and store) the default values for the settings of controls on the wizard pages.

The [**Wizard**](http://127.0.0.1:50000/help/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/wizard/Wizard.html) class implements many of the details for standard wizard behavior. You typically extend this class to implement behavior specific to your wizard. The primary responsibilities of your wizard will include:

* Creating and adding your pages to your wizard
* Implementing the behavior that should occur when the user presses the **Finish** button.

### Wizard page

The wizard page (**[IWizardPage](http://127.0.0.1:50000/help/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/wizard/IWizardPage.html)**) defines the controls that are used to show the content of the wizard page. It responds to events in its content areas and determines when the page is completed.

Your wizard page typically extends the [**WizardPage**](http://127.0.0.1:50000/help/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/jface/wizard/WizardPage.html) class. The primary responsibilities of your wizard page will include:

* creating the SWT controls that represent the page
* determing when the user has supplied enough information to complete the page (that is, when the user can move to the next page.)

**How to do:**

#### Wizard

The wizard is responsible for creating the pages and providing the "finish" logic.

The basic patterns for implementing a wizard include:

* Implement the **init** method to set up local variables for context information such as the workbench and the current selection.
* public void init(IWorkbench workbench,IStructuredSelection selection) {
* this.workbench = workbench;
* this.selection = selection;
* setWindowTitle(MessageUtil.getString("New\_Readme\_File"));
* setDefaultPageImageDescriptor(ReadmeImages.README\_WIZARD\_BANNER);
* }
* Implement **addPages** by creating instances of the pages.
* public void addPages() {
* mainPage = new ReadmeCreationPage(workbench, selection);
* addPage(mainPage);
* }
* Implement **performFinish** to finish the task.  
  Multi-page wizards typically handle the finish logic in the wizard itself, since each page will contribute information that determines how the task is implemented. Single page wizards can implement the logic in the wizard or ask the page to finish the job. The approach you take largely depends on where your important state is kept. In the case of the readme wizard, we are going to ask our page to handle the finish processing.
* public boolean performFinish() {
* return mainPage.finish();
* }

#### Pages

The workbench provides base wizard page classes that support the type of processing performed for each wizard extension point. You can use these pages, or extend them to add additional processing.

The goal of the **ReadmeCreationWizard** is to create a new file, add the required content to the file, and as an option, open an editor on the file. Our page needs to define the controls that let the user specify what content goes in the file and whether an editor should be launched.

We create the wizard page, **ReadmeCreationPage**, by extending[**WizardNewFileCreationPage**](http://127.0.0.1:50243/help/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/ui/dialogs/WizardNewFileCreationPage.html). The controls for a wizard page are defined in a fashion similar to the definition of the controls for a view or an editor. The page implements a **createControl** method, creating the necessary SWT widgets as children of the supplied[**Composite**](http://127.0.0.1:50243/help/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/swt/widgets/Composite.html). Since the superclass already adds widgets that support new file processing, we need only extend the **createControl** method in our wizard page to add the additional checkboxes that control generation of sections and opening of the editor.

public void createControl(Composite parent) {

// inherit default container and name specification widgets

super.createControl(parent);

Composite composite = (Composite)getControl();

...

// sample section generation group

Group group = new Group(composite,SWT.NONE);

group.setLayout(new GridLayout());

group.setText(MessageUtil.getString("Automatic\_sample\_section\_generation"));

group.setLayoutData(new GridData(GridData.GRAB\_HORIZONTAL |

GridData.HORIZONTAL\_ALIGN\_FILL));

...

// sample section generation checkboxes

sectionCheckbox = new Button(group,SWT.CHECK);

sectionCheckbox.setText(MessageUtil.getString("Generate\_sample\_section\_titles"));

sectionCheckbox.setSelection(true);

sectionCheckbox.addListener(SWT.Selection,this);

subsectionCheckbox = new Button(group,SWT.CHECK);

subsectionCheckbox.setText(MessageUtil.getString("Generate\_sample\_subsection\_titles"));

subsectionCheckbox.setSelection(true);

subsectionCheckbox.addListener(SWT.Selection,this);

...

// open file for editing checkbox

openFileCheckbox = new Button(composite,SWT.CHECK);

openFileCheckbox.setText(MessageUtil.getString("Open\_file\_for\_editing\_when\_done"));

openFileCheckbox.setSelection(true);

...

}

You should be able to follow this code if you understand the concepts in[Standard Widget Toolkit](http://127.0.0.1:50243/help/topic/org.eclipse.platform.doc.isv/guide/swt.htm).

The basic patterns for implementing a page include:

* Add listeners to any controls that affect dynamic behavior of the page. For example, if selecting an item in a list or checking a box affects the state of other controls of the page, add a listener so you can change the state of the page.
* Populate the controls with data based on the current selection when the wizard was launched. Some of the data may depend on the values in other controls. Some of the controls may use dialog settings to initialize their values.
* Use **setPageComplete(true)** when enough information is provided by the user to exit the page (and move to the next page or finish the wizard.)

The **ReadmeCreationPage** class inherits a lot of this behavior from the[**WizardNewFileCreationPage**](http://127.0.0.1:50243/help/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/ui/dialogs/WizardNewFileCreationPage.html). Browse the implementation of these classes for further information.



大多数的类都封装好了在这个包里面能找到。

**NewFileWizard**

