

Workbench and JFace Foundations

Part One, of a two part tutorial series

About the Speakers

- **Tod Creasey**
 - Senior Software Developer, Platform UI Committer
 - working with the IBM OTI Ottawa labs (formerly known as Object Technology International) since 1993 on a variety of IDEs starting with Envy developer and progressing through the VisualAge family of Smalltalk and Java IDEs
 - joined the Eclipse team during the tech preview and has been a member of the Workbench team since
- **Michael Van Meekeren**
 - Platform UI Team Leader, Platform UI Committer
 - working with the IBM OTI Ottawa labs (formerly known as Object Technology International) since 1994, and has played an active role in the development of Envy developer, IBM Smalltalk, VisualAge for Java and WebSphere Studio Device Developer
 - also has J2ME and embedded programming experience
- We have a lot of experience on the UI team for Eclipse, however do not know everything, will try to give answers to as many questions as possible

Getting Started:

- Survey
 - Eclipse Experience
 - Specific things to cover
- Laptop Setup
- Goals for this section
- What this tutorial is NOT
- JFace and Workbench plug-in structure
- Questions/Comments

Setup

- Start your engines!!!
- If you have a laptop
 - Eclipse SDK installed?
 - Tutorial Examples installed?
 - Java 1.4.2 VM
- No laptop?
 - Think of it as an opportunity to meet someone new
- CDs/Flash memory cards available
- NOTE: questions and comments are welcome at any point

Conventions used in this presentation

Courier text refers to code or XML in the text

`Turquoise boxes are code or xml snippets`



TIP: Yellow boxes are for tips

Goals

- Understand the Workbench and JFace plug-in structure
- Learn the components that make up JFace
 - build a simple “Image Browser” application
 - take a basic SWT application
 - migrate it into a JFace application
- Learn the basic components of the Workbench
 - migrate the JFace application into the Workbench
- Questions/Comments



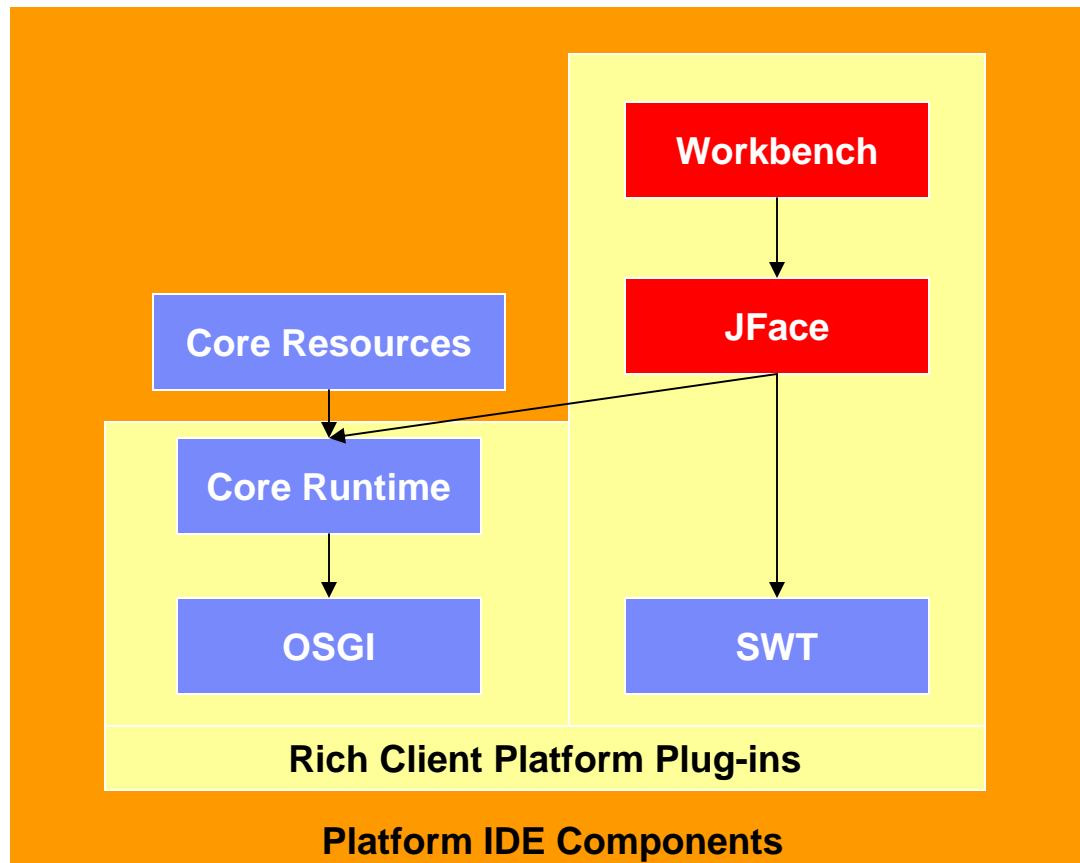
TIP: SWT, JFace and the Workbench plug-ins are designed in layers, the order for doing hands on work reflects these layers

What this tutorial is NOT

- Not a tutorial about how to write plug-ins or use the PDE (Plug-in Development Environment)
- Not an advanced Workbench tutorial
 - that's later on today
- Not a Rich Client Platform tutorial
 - that's also later today
- Not 4 hours of two guys talking
 - hands on tutorial

Part 1 - JFace

Platform Plug-in Structure



Why did we create JFace?

- JFace provides some tools for using SWT that are independent of the Eclipse Workbench
- Is a layer on top of SWT
- Has no extension points
- Has only a few dependencies on org.eclipse.core.runtime plug-in
- JFace is not about working as an integrated Eclipse plug-in

When is SWT enough?

- SWT is a thin layer on top of the Operating System
- Does not handle lifecycle of Operating System resources
- Works in terms of Strings, Colours etc., not Objects
- Provides common API for all Operating Systems
- Great for small/lightweight GUI applications



TIP: Use SWT when all of your content is already in the format SWT requires (i.e. Strings and Images)

What can you do with JFace?

- Represent your objects and their relationships using SWT widgets.
- Create wizards and preferences
- Manage your OS resources like Images, Colours and Fonts
- Defines common actions that can be placed in Menus and Toolbars



TIP: Use JFace for lightweight applications that represent Objects in a simple way.

What are we covering in Part 1 (JFace)

- Taking an SWT application and showing how it is easier to develop using JFace
- Viewers and their components
- Actions
- Preferences, preference dialogs and field editors
- Progress monitors
- Dialogs
- Registries for operating system resources (e.g. Images)
- Building and launching a wizard

What are we going to build?

- Building an Image browser by:
 - Taking an SWT application and making it more useful with JFace
 - Creating viewers and their components
 - Using JFace registries for fonts, colours and images
 - Building and launching a wizard
 - Managing resources
 - Making preference dialogs and field editors
 - Creating menus with Actions

Lets start with SWT only

- Create a shell with:
 - a Text widget to display a directory path
 - a Button to browse for paths
 - a Tree to show the contents of the directory
 - a Canvas to display an image when we select it



TIP: You need to have the SWT libraries on your path when you run an application stand alone. Use `-Djava.library.path` to specify this.

Launching our examples

- We have provided
`org.eclipsecon.jface.example.TutorialLauncher`
- We made a Launcher so that it is easier to launch all of the examples
- Select TutorialLauncher in the PackageExplorer
- Select Run -> Run... from the window toolbar
- Select Java Application
- Add the following line to your vm arguments

```
-Djava.library.path=  
<your directory structure>\eclipse\plugins\org.eclipse.swt.win32_3.1.0\os\win32\x86
```

- Run it
- Select "SWT - Image File Browser" to open it

Questions and Comments?



Viewers

- See `JFaceImageFileBrowser`
- Convert the SWT Tree to a `TreeViewer`
- Create a Label provider
 - we want to show the user something better than `toString()` for Files
- Create a content provider
 - Use a `java.io.FileFilter` to filter on image files
- Don't forget to `setInput()` to get the ball rolling



TIP: Viewers do not start generating contents until `setInput()` is called so set your content and label providers first.

Actions

- SWT has listeners on operating system events
- Does not have the concept of a reusable Action
- Add action for opening preferences
- Launch it from the SWT selection callback on the button

Preferences

- SWT does not have the concept of preferences
- JFace uses the `IPreferenceStore`
- Core has Preferences as well but it is more tied to the file system
- In the Workbench we will see a `ScopedPreferenceStore` which is an `IPreferenceStore` that has Core as a backend

What If Our Preference Needs Are Simple?

- We only want a preference store based on properties files – we don't need the workbench
- See `JFaceImageFileBrowser`
- Put a preference page in a preferences dialog and put in a simple preference
 - starting directory for images folder
- Launch `JFaceImageFileBrowser` and select the “Set Preferences” button

ProgressMonitor

- To make it more interesting we are going to draw as many images as we can show in a directory
- See `JFaceMultiImageFileBrowserWithProgress`
- Sometime operations are long and you want to give the user feedback
- SWT only provides OS dialogs
 - SWT provides the system dialogs
 - E.g. `MessageDialog`, `DirectoryDialog`
 - JFace provides other useful dialogs such as:
 - `ProgressMonitorDialog`
 - `ErrorDialog` / `MessageDialog`
- Use a `ProgressMonitorDialog` to open with progress when loading an image
- Create an `IRunnableWithProgress`

Questions and Comments?



Dialogs

- The Dialog class provides many common Dialog features
 - modality
 - OK/Cancel buttons
- Also provides some API to keep your Dialogs consistent with Eclipse dialogs
 - images: `Dialog.DLG_IMG_MESSAGE_INFO`
 - spacing
 - dialog areas and button bar

Converting to a Dialog

- See `JFaceImageFileDialog`
- Make the example a JFace Dialog now
- Gives us access to the API images Dialog provides which we will use for showing info
- Move the code for filling the shell to `createDialogArea()`
- We now get OK and Cancel buttons for free
- Spacing is consistent with other Eclipse dialogs
- We no longer have to handle `Shell` and `Display` objects
 - We get the display from the Tutorial Launcher

Resource management via descriptors

- Operating System resources have to be managed by the application
- No garbage collector for Images, Fonts or Colours
- We use descriptors to do this
 - Images use `ImageDescriptor` (JFace)
 - Fonts use `FontData` (SWT)
 - Colours use `RGB` (SWT)



TIP: Descriptors do not allocate system resources and are a good way to specify resources that you may or may not use.

Image Cache

- JFace also provides the ImageCache which disposes an Image when there are no more references using WeakReferences
- Allows you to share images and not worry about disposing one other applications may use.
- See `JFaceExampleBuilder#getCache` and `#disposeCache`



TIP: Weak References are disposed when there are no more references to the Object in the VM. This is a good thing, but be sure to hold onto any images you need in a field if you don't want them disposed.

Wizards

- See `JFaceImageFileDialogWithWizard`
- Wizards are multi page dialogs with some useful space for images, title and status
- `WizardDialogs` are an `IWizardContainer` that can handle all of this
- An `IWizard` has an `IWizardContainer` and some `IWizardPages`
- `WizardPages` implement what you need for your `IWizardPages`

Questions and Comments?



What we will cover in Part 2

- Views
- Workspace
- Preferences
- Editors
- Actions

Part 2 – Workbench Basics

Self hosting

- What is self hosting?
- Why is it useful
- Creating a new configuration



**TIP: Self hosting is also referred to as:
running an inner Eclipse
running a runtime-workspace**

Extension Points

- An extension point is an xml markup for elements of the workbench
- Prevents the need for more API
- All that is required is defined in the schema
- You can read the schema by selecting Help->Contents->Platform Plug-in Developer Guide->Reference->Extension Points Reference
- PDE tooling uses the schemas to help you define extensions



TIP: You should only use classes that are defined in a package that does not have the word **internal** in it as these are subject to change at any time.

Views

- See `org.eclipsecon.workbench.example.ImageNavigator`
- We use the views extension point
- Create a view stub using PDE in a category
- Create a `ViewPart` subclass to satisfy the schema
- Move the create contents code
 - take the contents of `createDialogArea()` in the Dialog
 - move it to `createPartControl()` of the View
- Self host and open the view using Window > Show View



TIP: When creating extensions the plug-in containing the extension point definition must be listed as a required plug-in in order for it to be visible.

Workbench

- What is the workbench
 - user interface concept - `org.eclipse.ui.workbench` defines it
- What is the workspace
 - core concept - `org.eclipse.core.resources` defines it
 - what is a resource
 - what is the workspace root
- We will convert the viewer starting directory to be the workspace root
- Launch again

Preferences

- Define a preference page in an extension point
- Create a page for the preference
- We now have a preference store from our plug-in
- Delete the properties file preference store
- Migrate to the plug-in preference store
- NOTE: preferences are created for plug-ins that subclass `AbstractUIPlugin` for free



TIP: Any subclass of `AbstractUIPlugin` has an `IPreferenceStore` that can be used whenever one is required.

Questions and Comments?



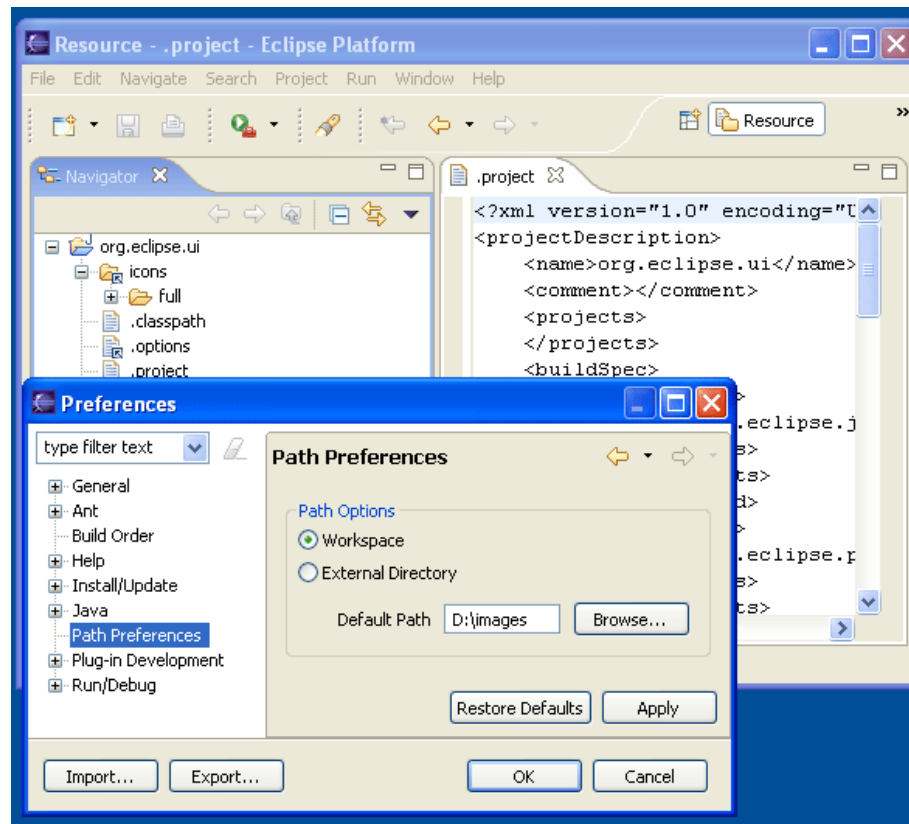
Editors

- Workbench windows reserve an editor area for you
- We want to use the image canvas in an editor
- Define an editor using the editors extension point
 - this is used by several views to determine editors associated with the file they are displaying
 - based on file extensions
- Create an editor associated with “gif”

Actions

- This editor will also be used by the Resource Navigator as it builds its editor list from the extension point
- Self host again
- Look for the editor in the Resource Navigator
- Some actions are determined by extension points
- If we register an editor we will get an action for it in any viewer that builds editor actions from the registry

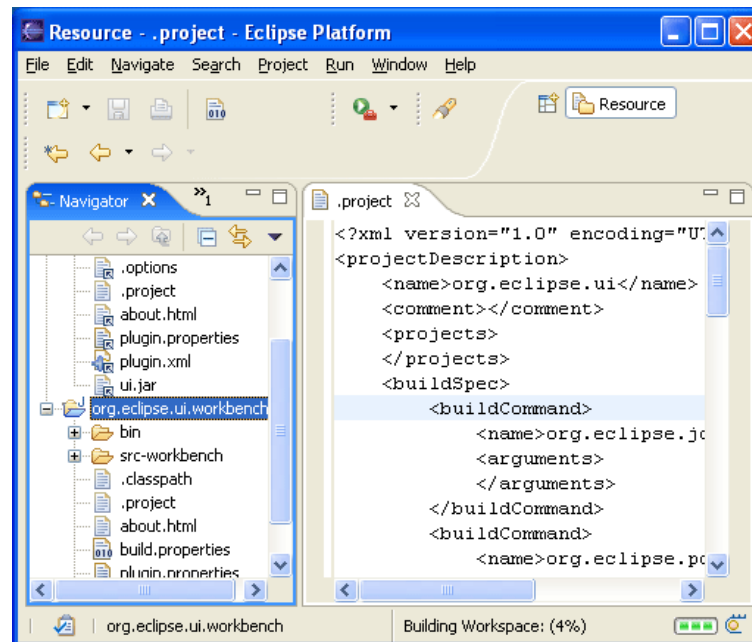
Workbench Components Recap



Workbench Components Recap

- What have we done so far
 - Views
 - Workspace
 - Preferences
 - Editors
 - Actions

Workbench Components in the next section



Workbench Components in the Advanced Tutorial

- Filters
- ActionSets/Commands/Keys
- More on Editors
- Wizards
- Perspectives
- Startup and Shutdown
- Jobs
- Decorators
- Adaptable
- Common Expressions
- Help

Questions and Comments?

