

Foundations of User Interface Programming Using the Eclipse Rich Client Platform



Tod Creasey
IBM Canada





About the Speaker

- **Tod Creasey**
 - Senior Software Developer, Platform UI Committer
 - working with the IBM Rational Software since 1994, and has played an active role in the development of Envy developer, IBM Smalltalk, VisualAge for Java
 - moved on to the Eclipse Platform UI team during the 1.0 development cycle



Goals

- Understand the JFace toolkit structure
- Learn the components that make up JFace
 - build a simple “Image Browser” application
 - take a basic SWT application and migrate it to JFace

SWT, JFace and the Workbench plug-ins are designed in layers, the order for doing this work reflects these layers

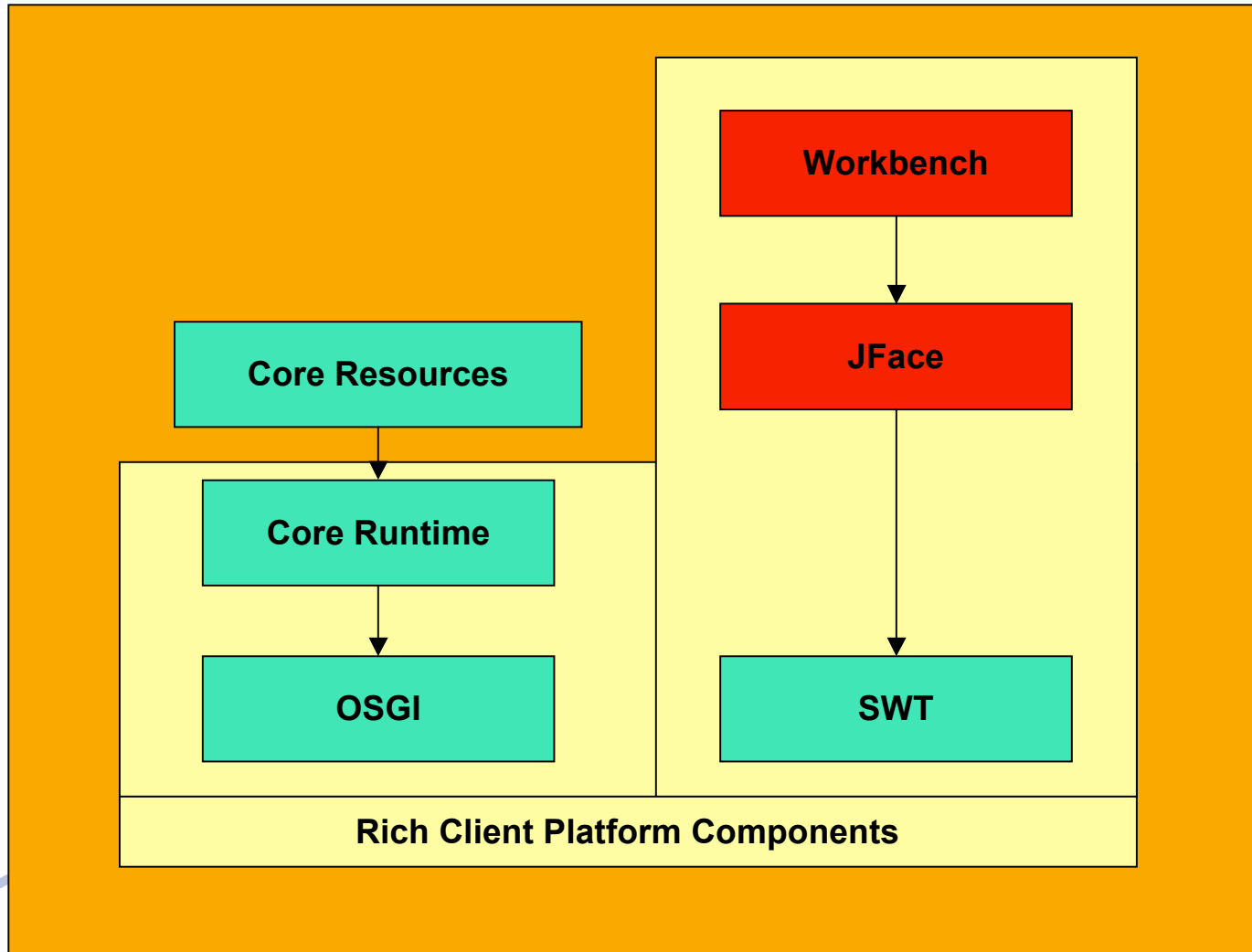


What This Talk Is NOT

- Not a talk about how to write plug-ins or use the PDE (Plug-in Development Environment)

- Not an Advanced RCP talk (that is at another time)

Platform Component 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 dependencies on a utility jar shared with Core Runtime and OSGI
- JFace is not about working as an integrated Eclipse plug-in
- Designed to support a JFace + SWT only application



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



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



What Are We Covering (JFace)

- Take an SWT application and show how it is easier to develop using JFace
- Viewers and their components
- Actions
- Preferences, preference dialogs and field editors
- Progress monitors
- Dialogs
- Management 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 managers for fonts, colours and images
 - Building and launching a wizard
 - Managing resources
 - Making preference dialogs and field editors
 - Creating menus with Actions

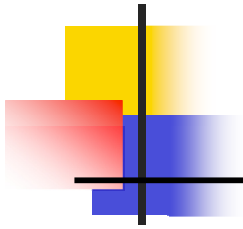


Let's Start with SWT only

- Create a Display and then 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 the specialized Eclipse launch for SWT. Run As > SWT Application

Demo of SWTImageFileBrowser and Questions





Viewers

- Most people use Viewers instead of Widgets for representing multi element information
 - Lists
 - Tables
 - Trees
 - Combos
- The Viewer API deals with model objects so there is no need to convert to objects SWT can understand



Components of a Viewer

- **LabelProviders**

- Bridge between the model and the widgets
- Allows you to specify text, images and sometimes fonts and colours for model objects from the content provider
- Management of widgets handled by JFace

- **ContentProviders**

- Provides the model objects to the viewer.

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

Some Not So Obvious Advantages of Viewers



- Widget lifecycle is handled for us
- Fonts, colours, text and images are queried and updated using the label provider
- Reuse of existing widgets, creation and disposing
 - i.e. If a Tree need to be refreshed the TreeViewer will
 - Reuse those that it can
 - Create new TreeItems if required
 - Delete old TreeItems



Filters

- Viewers can have ViewerFilter
- Filters test the contents to see if they will be displayed
- Want to reduce the shown elements to just those that are images
- We had to write a FileFilter for the SWT example
- FileFilter forced us to filter at the content provider
- If contents were slow to get we would need to recalculate whenever the filtering changes



Sorters

- Viewers can have ViewerSorters
- Also allows changes to a viewer without having to ask for contents again
- We want to sort the contents alphabetically

TIP: The sorters and filters are a nice way for a reusable view to modify the contents of the view without having to ask for elements again. This is used a lot by TableViews which sort by column.



Actions

- SWT has listeners on operating system events
- These need to be implemented in a widget specific way
- SWT does not have the concept of a reusable Action
- The Workbench (see part 2) uses Actions to build menus, toolbars, *etc.*

- JFace Actions can be reused in many places
 - Menus
 - Toolbars
 - Commands/Key bindings

- We will write an action to open a preference dialog and invoke it from a button



Preferences

- SWT does not have the concept of preferences
- JFace uses the `IPreferenceStore` which is open about it's implementation
- 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
- We can create an `IPreferenceStore` using `java.util.Properties` as a back end
- Add an action for opening preferences
- Put a preference page in a preferences dialog and put in a simple preference



starting directory for images folder



Preferences Support in JFace

- `PreferencePage` is set up to handle
 - Restoring Defaults
 - OK
 - Apply
 - Accessing the `IPreferenceStore`
- `IPreferencePageContainer` defines the interaction between `PreferencePages` and their container
- JFace provides the `PreferenceDialog` as a standard `IPreferencePageContainer`



ProgressMonitor

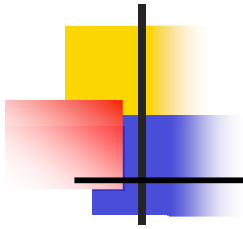
- To make it more interesting, we are going to draw as many images as we can show from a directory
- Sometimes operations are long and you want to give feedback
- SWT only provides OS dialogs
 - SWT provides the system dialogs
 - *e.g.* `MessageDialog`, `DirectoryDialog`
 - JFace provides other useful dialogs such as:
 - `ProgressMonitorDialog`
 - `ErrorDialog / MessageDialog\`



ProgressMonitor

- Use a `ProgressMonitorDialog` to open with progress when loading an image
- Create an `IRunnableWithProgress`

Demo of JFaceImageFileBrowser and Questions



Dialogs

- The Dialog class provides many common Dialog features
 - modality
 - OK/Cancel buttons
 - Provide a specialized window used for narrow-focused communication with the user by wrapping an SWT shell widget
 - Handles much of the setup work for you
 - Creates a Shell
 - Creates a contents and buttons area



Dialogs

- Also provides some API to keep your Dialogs consistent with Eclipse dialogs
 - images:
 - `Dialog.DLG_IMG_MESSAGE_INFO`
 - `Dialog.DLG_IMG_MESSAGE_WARNING`
 - `Dialog.DLG_IMG_MESSAGE_ERROR`
 - spacing
 - dialog areas and button bar
- Dialog also initializes the JFace ImageRegistry so make sure you have a Display created first



Converting to a Dialog

- 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
 - Use convenience methods in Dialog for this
- We no longer have to handle `Shell`

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



ResourceManager

- JFace also provides the ResourceManager which disposes an Image when there are no more references using reference counting
- Allows you to share images and not worry about disposing ones other applications may be sharing
- See `ResourceManager#dispose()`
- If your Images are going to be unused periodically (*i.e.* they are only used in one Dialog) you may want to create a `LocalResourceManager` to cut down your application's size



Wizards

- 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`
- **Create a wizard and wizard dialog, then add some wizard pages to it**

Demo of JFaceDialogLauncher to Open JFaceImageFileDialog and Questions

