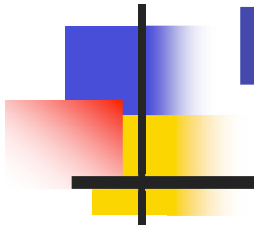
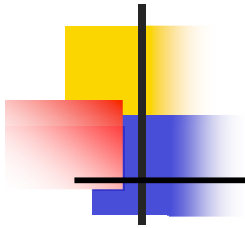


Advanced User Interface Programming Using the Eclipse Rich Client Platform

Tod Creasey

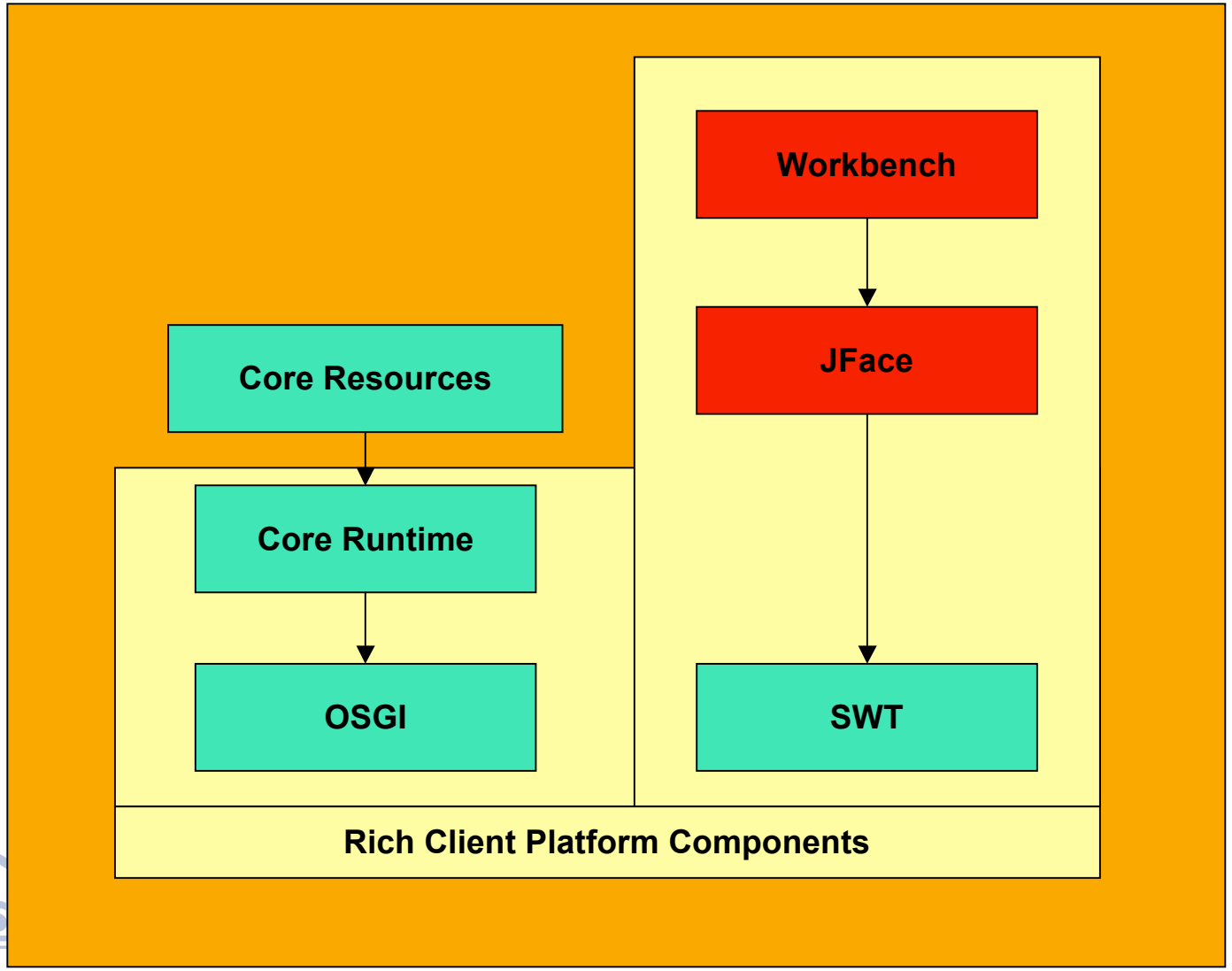
IBM Canada

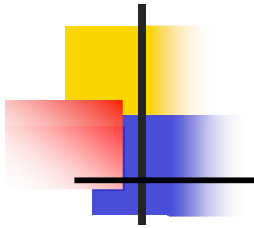


About the Speaker

- 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

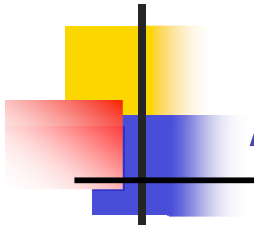
Platform Component Structure





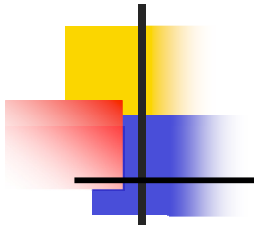
Plug-in Structure

- JFace is dependant on SWT and a small utility jar shared with Core Runtime and OSGi
- The workbench uses JFace and Core Runtime to build a full-featured application framework integrated with Eclipse
- The IDE layer adds a dependency on Core Resources and exposes the workspace resource model
- The IDE is an RCP application

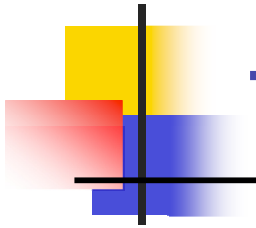


Applications

- See `org.eclipse.core.runtime.applications`
- You need at least one application to run Eclipse with the Workbench
- The IDE is an RCP application
- If you use your own application you may not have access to the IDE classes and extensions
- `org.eclipse.ui.ide` assumes that the IDE application is the one running



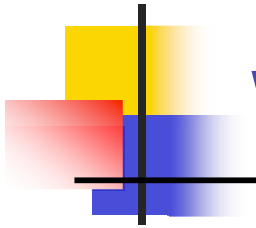
Questions and Comments?



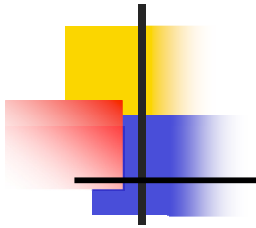
Workbench and IDE Advanced Topics

- Views
- Workbench
- Preferences
- Editors
- Commands
- Wizards
- Perspectives
- Startup and Shutdown
- Jobs
- Adaptable

Where We Started and Where We Are Going

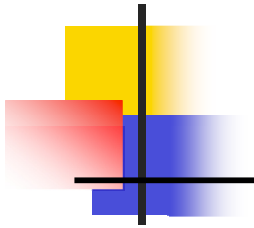


- Created an SWT application to display Images in a directory
- Converted it to a JFace Dialog
- Used none of the Platform Core support for integration
- Now going to integrate it into the Workbench
- Demo of JFaceImageFileDialog to see where we started



Extension Points

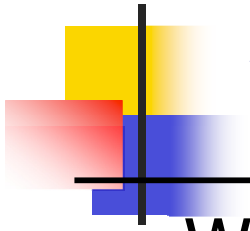
- An extension point is an XML markup for elements of the workbench
- Prevents the need for more API for hooking up an extension
- All that is required is defined in the schema



Extension Points

- Allows the extension provider to decide how to use it
- For instance the newWizards extension point is used by the File->New internal item
- PDE tooling uses the schemas to help you define extensions

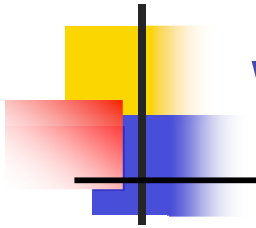
TIP: You should only use classes that are defined in a package that does not have the word internal in it, as these are subject to change at any time.



Views

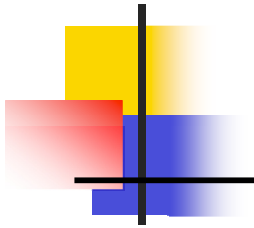
- We use the views extension point from `org.eclipse.ui.workbench`
- Create a category and specify it as the category for the view
- Create a `ViewPart` subclass to satisfy the schema
- Move the create contents code from the Dialog
  - take the contents of `createDialogArea()` in the Dialog
  - move it to `createPartControl()` of the 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.



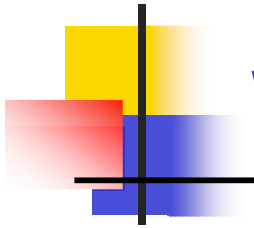
Workbench

- 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?



Preferences

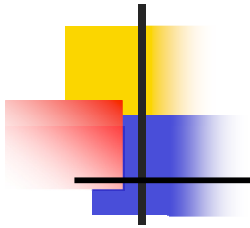
- Extension point to hook into the Workbench preferences dialog
- Define a preference page in an extension point
- Create a page for the preference
- This page will show up in the preference dialog
- Can be used to set up anything you like



Where Are Preferences Stored?

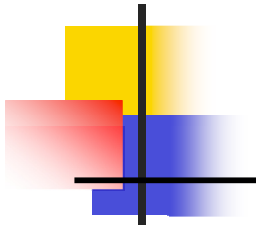
- Wherever you like
- Eclipse has a preferences mechanism
- Initialized using an `PreferenceInitializer`
- Any subclass of `AbstractUIPlugin` has an `IPreferenceStore`

TIP: Be sure to initialize your preference store using a PreferenceInitializer, as it will only get populated if you access it.

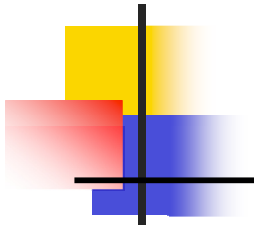


Preferences

- First define a `PreferenceInitializer` to be run when the plug-ins preference store is first accessed
  - use the `org.eclipse.core.runtime.preferences` extension point to define the initializer
  - pre 3.0 we set these up on plug-in startup, but this is inefficient if the preference is never referenced, and adds to startup time
- Now add a preference listener to the view, so the refresh will occur as soon as the preference changes

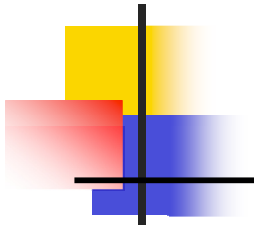


ImageFileView Demo Questions and Comments?



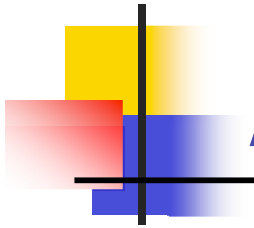
Editors

- Workbench windows reserve an editor area for you
- We want to re-use the image canvas from the previous dialog 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



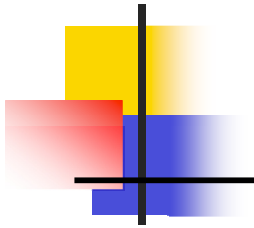
Editors and IResource

- The contents of the Workspace are defined in terms of IResource
- IResource can be based on files
- As our editor is displaying the contents of a file, we will use the resources plug-in from core
- This lets us use the contents of the Resource Navigator and anything else that shows files (like the Package Explorer)

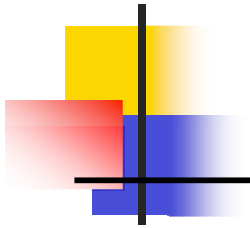


Actions for Opening an Editor

- This editor will also be used by the Resource Navigator and Package Explorer, as they build their editor list from the editors extension point
- Look for the editor in the Open With context menu for a "gif" file from the Resource Navigator
- If we register an editor, we will get a context menu entry for it in any viewer that builds the Open With menu from the registry



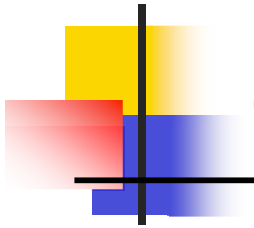
Demo of ImageEditor and Comments



Commands/Keys

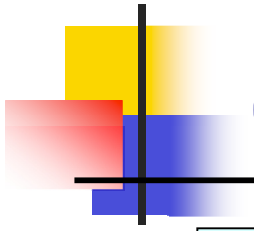
- Define a command in the extension point (`org.eclipse.ui.commands` for the action and provide a key binding in the `org.eclipse.ui.bindings` extension point
  - bind to our action that opened the filter settings dialog
  - provide a key binding for the command (Ctrl+Alt+Shift+I)

TIP: Eclipse applications must share key bindings between plug-ins, so it is easy to run out of available key sequences. As the user can set these themselves using the Keys Preference Page, it is not a requirement that you define key sequences for all of your commands.



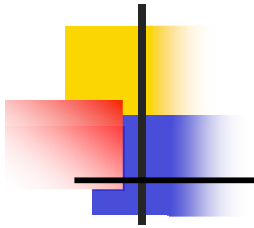
Commands Extensions

- org.eclipse.ui.commands extension point provides:
  - Commands
    - represents a request from the user
- org.eclipse.ui.handlers
  - Defines how a command will be executed



Command and Handler Definition

```
<extension point="org.eclipse.ui.commands">
    <command
        description="Allows setting of filter options for images"
        id="org.eclipse.ui.rcptalk.imagesFilter"
        name="Image Filter Settings"/>
 </extension>
<extension
       point="org.eclipse.ui.handlers">
    <handler
        class="org.eclipse.ui.rcptalks.workbench.ImageFilterHandler"
        commandId="org.eclipse.ui.rcptalk.imagesFilter">
    </handler>
 </extension>
```

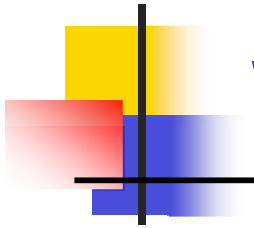


Bindings

- org.eclipse.ui.bindings
  - specified binding between commands and key sequences

```
<extension point="org.eclipse.ui.bindings">
    <key
        commandId="org.eclipse.ui.rcptalk.imagesFilter"
        contextId="org.eclipse.ui.contexts.window"
        schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"
        sequence="M1+M2+M3+I"/>
 </extension>
```

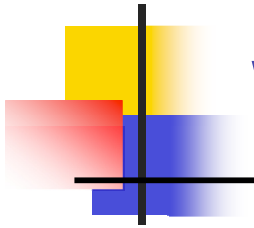
TIP: When defining a key sequence, it is better to use M1 (instead of Ctrl) and M2 (instead of Shift) for better portability across platforms



Wizards

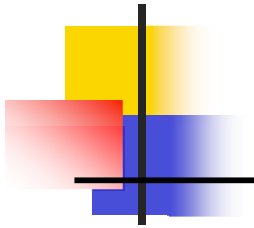
- The workbench has three wizard types defined in extension points
  - New
  - Import
  - Export
- When one of the wizard actions are invoked, we open a WizardDialog which provides a selection page (generated internally from the extension points) to choose one of the wizards defined in the extension points

Demo of the Command, Questions and Comments



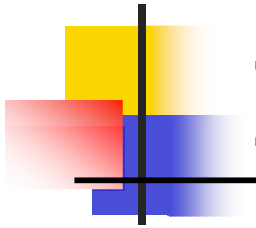
Wizards

- The workbench has three wizard types defined in extension points
  - New
  - Import
  - Export
- When one of the wizard actions are invoked we open a WizardDialog, which provides a selection page (generated internally from the extension points) to choose one of the wizards defined in the extension points



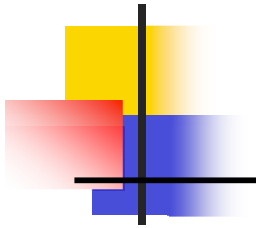
New Wizards

- New wizards have a series of re-usable first pages
  - `BasicNewResourceWizard` (abstract)
  - `BasicNewFileResourceWizard`
  - `BasicNewFolderResourceWizard`
  - `BasicNewProjectResourceWizard`
- Use these in your resource based wizards so that the resource is already created for you
- Although the extensions are in the Workbench, these classes are in the IDE plug-in so you may not have access to them if you define your own application.



Import/Export Wizards

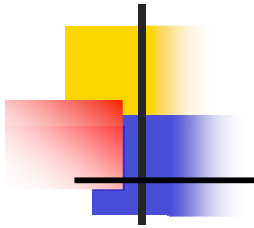
- Several classes are provided to assist In building an import or export wizard
  - `WizardResourceImportPage / WizardResourceExportPage`
    - provides basic layout for source and destination
    - provides an options group
  - `FileSystemStructureProvider`
    - provides structure for the `ImportOperation` based on `java.io.File`
  - `ImportOperation/ExportOperation`
    - basic operations for performing imports and exports
    - see `ImageImportPage`



Perspectives

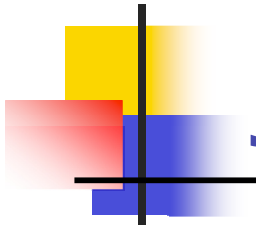
- Can also define our own perspectives
- Use a `PageLayout` positions parts and setup other standard menus, etc.
- Create an "Image Navigation" perspective
  - Define our own `IPerspectiveFactory`
  - Add our view in the Page Layout
  - Add the TasksView as a fast view
  - Add the Navigator View to the short list of views

Demo of the Wizards, Image Perspective Questions and Comments



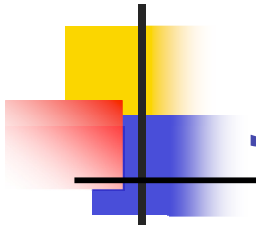
Startup and Shutdown

- The workbench tries to avoid loading other plug-ins by allowing all required information to be defined in the XML of an extension point
- Use the `startup()` and `shutdown()` methods in your plug-in to initialize the things you want to load, such as Images, Colors, etc.
- Plug-in loading should be avoided in general, but it is particularly noticeable when it slows down startup. You can avoid this by a lazy initialization approach whenever possible.



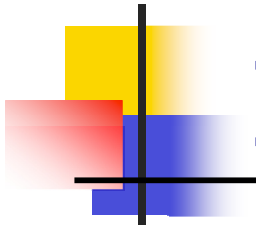
Jobs

- SWT and JFace must be accessed in the main Thread
- Without creating Jobs or Threads, Eclipse is single Threaded (referred to as the UI or main Thread)
- We want to process long operations that do not require SWT in a background Thread using a Job



Jobs

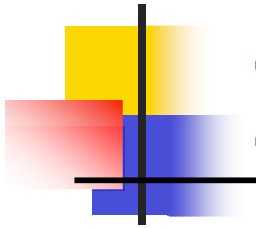
- Progress reporting is done in the progress area and the progress view
- Any work done in a Job does not block the main Thread, which makes the UI more responsive as it does not stop the user from working



IAdaptable

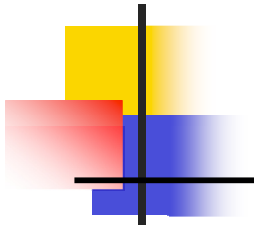
- What is it?
  - IAdaptable is a mechanism for accessing an Object of one type via another Object of a different type, without using interfaces

```
if (object instanceof IAdaptable) {//See if it supports IAdaptable

    //See if it adapts to IFile
    Object resource = ((IAdaptable) object).getAdapter(IFile.class);

    if (resource == null)//Returns null if not
            return;

    //Now we can safely cast to IFile and continue
    doFileOperation((IFile) resource);
}
```



IAdaptable

- Why do you care?
  - this is a good way to give access to an object without defining additional interfaces
    - JDT uses this for CompilationUnits (IFile), Packages (IFolder) and Java Projects (IProject)
    - Not just for IResource – you can adapt any Object to any other Object using the org.eclipse.core.runtime.adapters extension point
- Do we need it here?
  - no - we are using resources directly



The End

- Questions?