RCP Mail 2.0 -- Data Binding, Commands and Common Navigator Handout

Frank Gerhardt, Michael Scharf, Kai Tödter, Boris Bokowski, Francis Upton IV, Paul Webster

Abstract

The RCP Mail example shows its age. Many new and useful features have been added to the Eclipse Platform since the example was written, and it is time to take the example to new levels.

In a hands-on way, we will bring the example up to date by adopting some of the more interesting new APIs developed since the original RCP Mail example was written.

In particular, we will show the following:

- How to use the new Commands API to contribute to menus, toolbars, and context menus, and how to create key bindings.
- How to use the Common Navigator to provide a view that shows multiple sets of unrelated content.
- How to use the data binding framework to make the UI code easier to write, and easier to test.

Explanations of the concepts will be given by members of the Eclipse Platform UI team, while the concrete examples will be explained by experienced Eclipse professionals.

We will also show the different tools that are provided by Eclipse PDE and JDT to help you developing RCP applications.

http://www.eclipsecon.org/2009/sessions?id=641 [2]

List of Slides

The Plan

In this tutorial you will learn about

- Data binding - Frank and Boris
- Command framework - Kai and Paul
- Common Navigator Framework (CNF) - Michael and Francis

We start with the RCP Mail you probably know already

Then we add one feature at a time, with explanations and live pair-programming

You can follow us along in your IDE using the provided samples

We will have a programming exercise at the end of each block (3)

Steps

00: Original RCP Mail 1.0
01: Introduction of the Model (and artwork)
02: The New Server Wizard
03: Adding data binding
04: Adding validator and error messages
05: Adding field decorations
06: Using data binding in the NavigationView (ContentProvider)
07: Adding a view with a table of Messages
08: MessageView: 1. use selection of NavigationView, 2. bind table contents to selection details
09: **Exercise:** port binding with validator
10: Commands: add/change About, New Server command, ways of placing it in the UI, MarkAsSpam command, consolidate predefined actions/commands, Sync command, enabledWhen, visibleWhen, Delete command
11: **Exercise:** File/Exit and New Window command: exercise
12: CNF instead of SimpleNavigator, change in-place
13: Add Contacts plugin model and content/label providers
14: **Exercise:** Hook Contacts plugin to CNF

Set-up

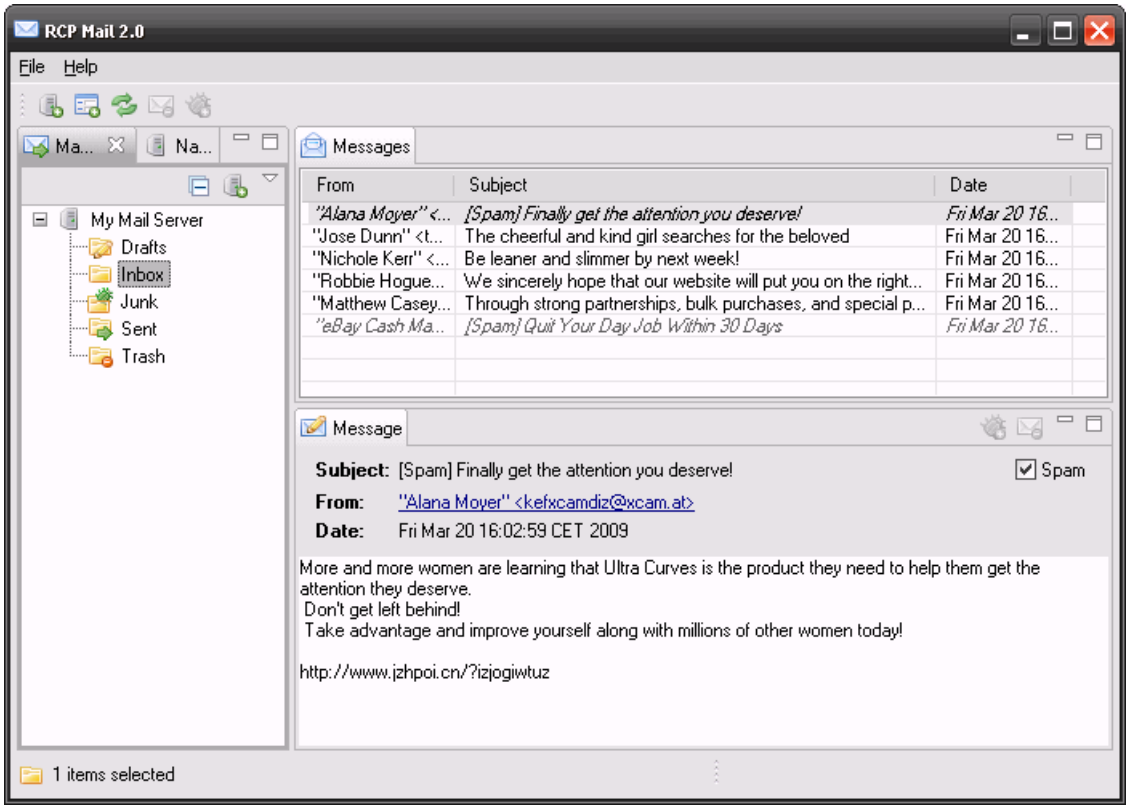
You need

- Eclipse SDK 3.4.x or higher as your IDE
- Eclipse SDK 3.5M6 as your target platform
- Our samples
  - Code for each step, rcpmail-00 to rcpmail-99
  - Handout (these slides)

You can get it from

- Download (slow in the conference center)
  - SDK from eclipse.org
  - Our samples from Kai's server. [3] **Continuous build!**
- USB sticks
  - A snapshot of our samples

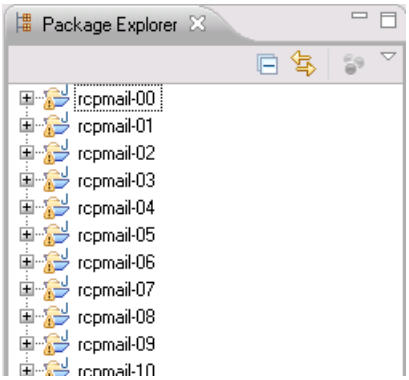
The Result



1 minute demo of the features

Samples

The zip file, e.g. **rcpmail-tutorial-2.0.0.v2009*.zip**, contains workspace projects

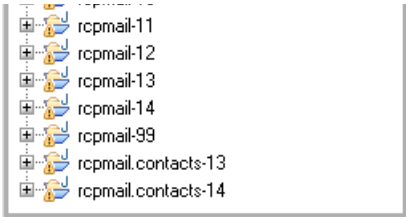


Open Eclipse with a new empty workspace

If you are not using 3.5M6 for development, set your **target platform** to 3.5M6

Import the projects from the zip file using **Import > Existing projects into Workspace**. Choose "Select archive file"

Every step contains a **launch configuration**. Try starting rcpmail-99

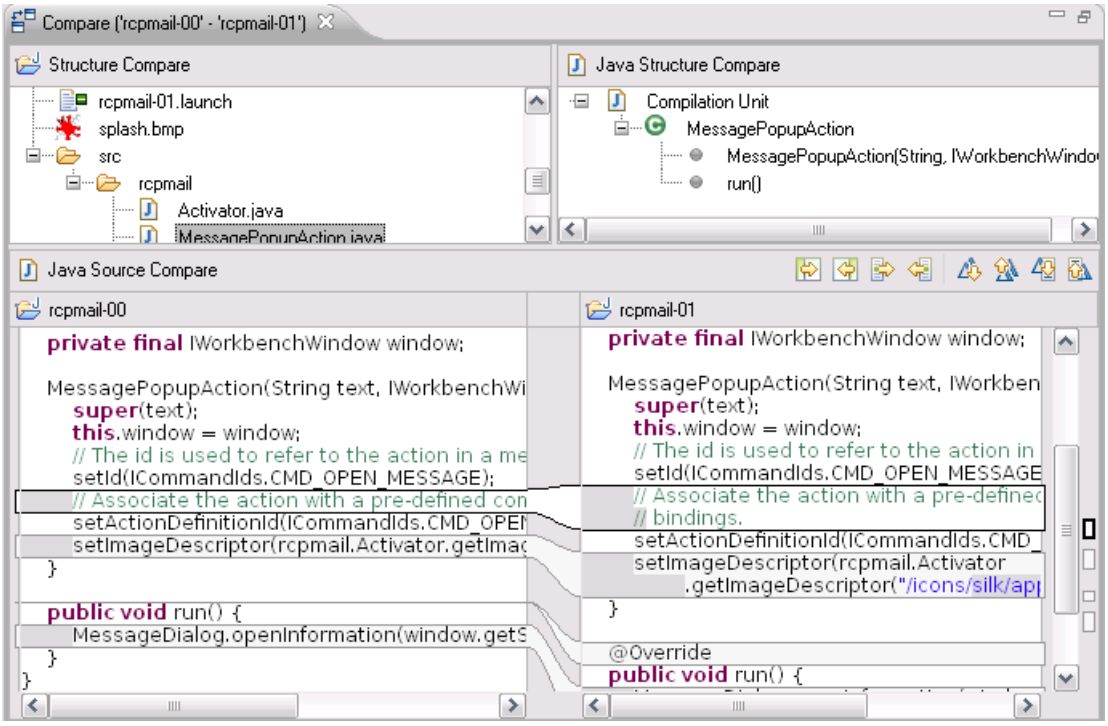


If you have problems, **raise your hand** and someone of us will come and help you

Comparing Steps

You can easily compare two steps to see what has changed

Select the old and the new project in the package explorer and invoke from the context menu **Compare With > Each Other**



Choosing the first and second selection carefully puts the previous project always on the left and the next on the right

Getting Closer

The first topic area is Data Binding

But: we need to bore you with some ground work first

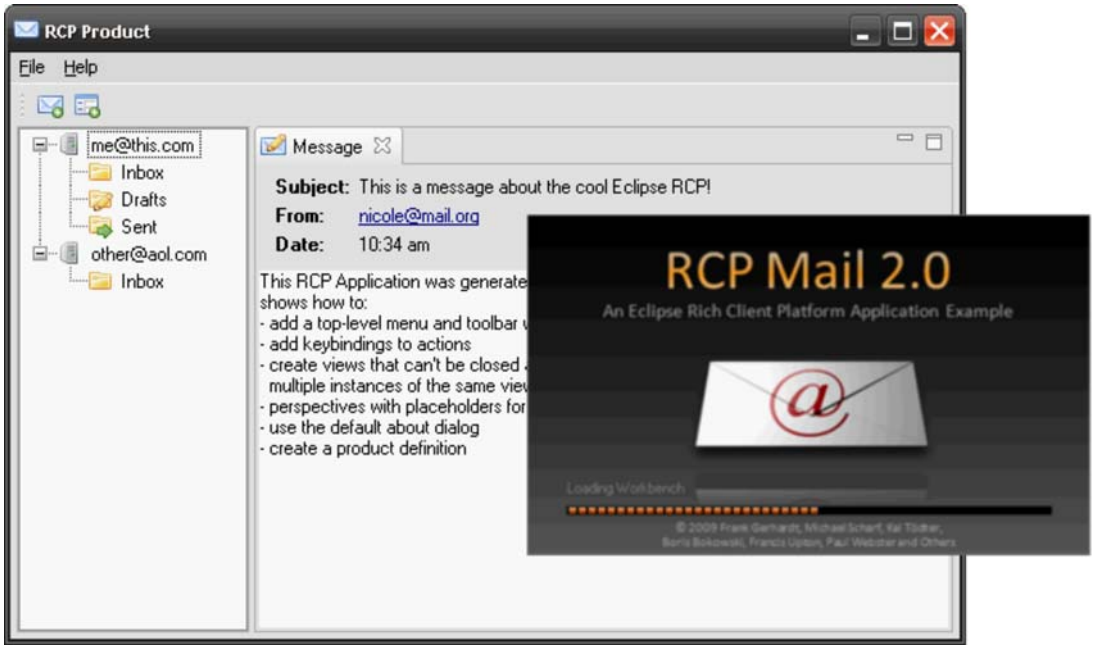
For a meaningful demo of data binding we introduce a domain model

While we are laying the foundation, we also update the artwork

Before

Have a look at rcpmail-00 in your workspace and launch it using the provided launch config

After



Have a look at rcpmail-01 in your workspace and launch it using the provided launch config

The Model

The model is a simple JavaBean model

ModelObject is the base class providing PropertyChangeSupport

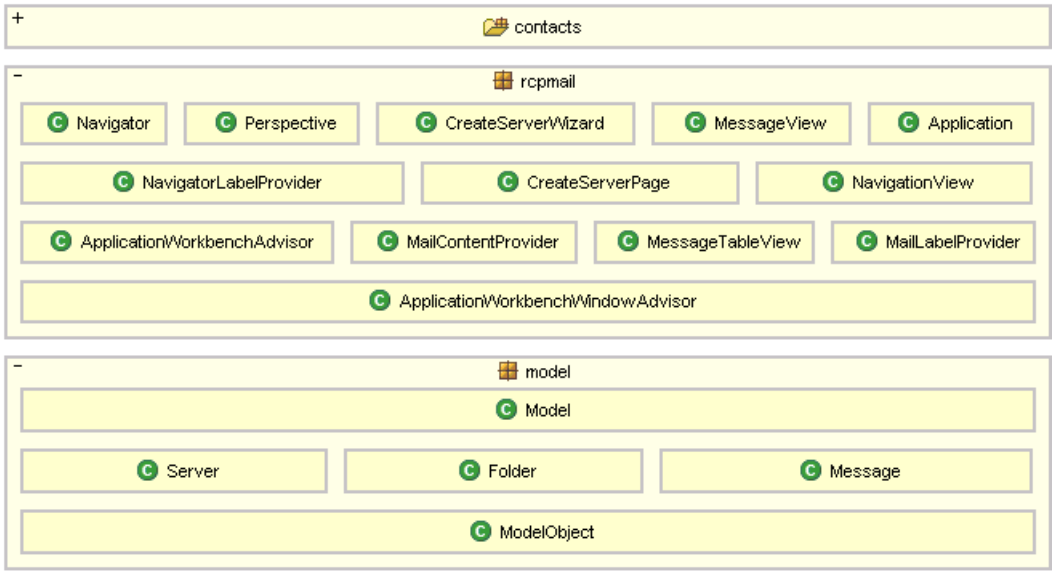
Changes to the template generated by the SDK

- Model, Server, Folder, Message
- Increased bundle version to 2.0.0

- Including test data
- Icons, splash

Look at the model in the rcpmail.model package of rcpmail-01

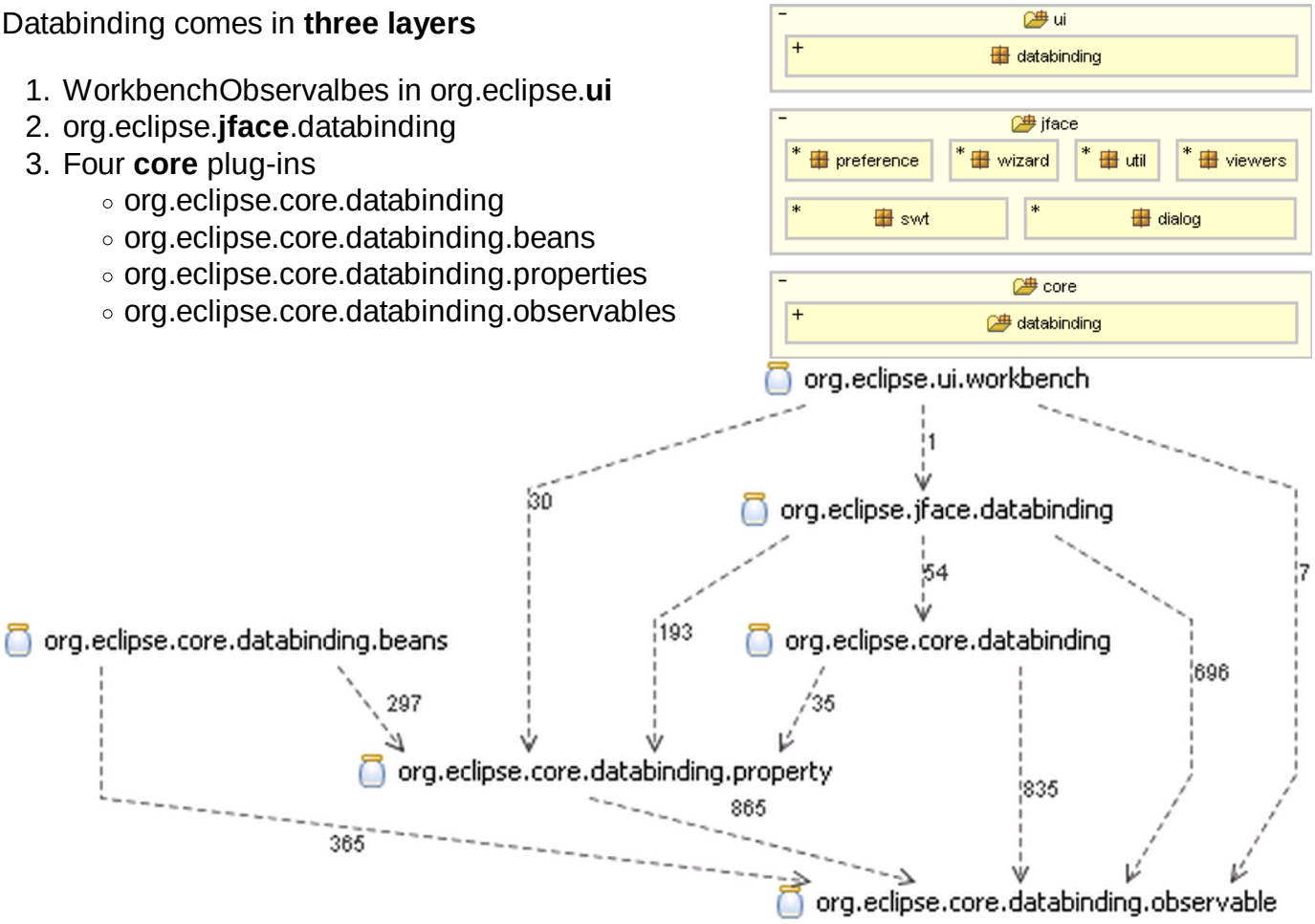
RCP Mail Classes



Databinding Plug-ins

Databinding comes in **three layers**

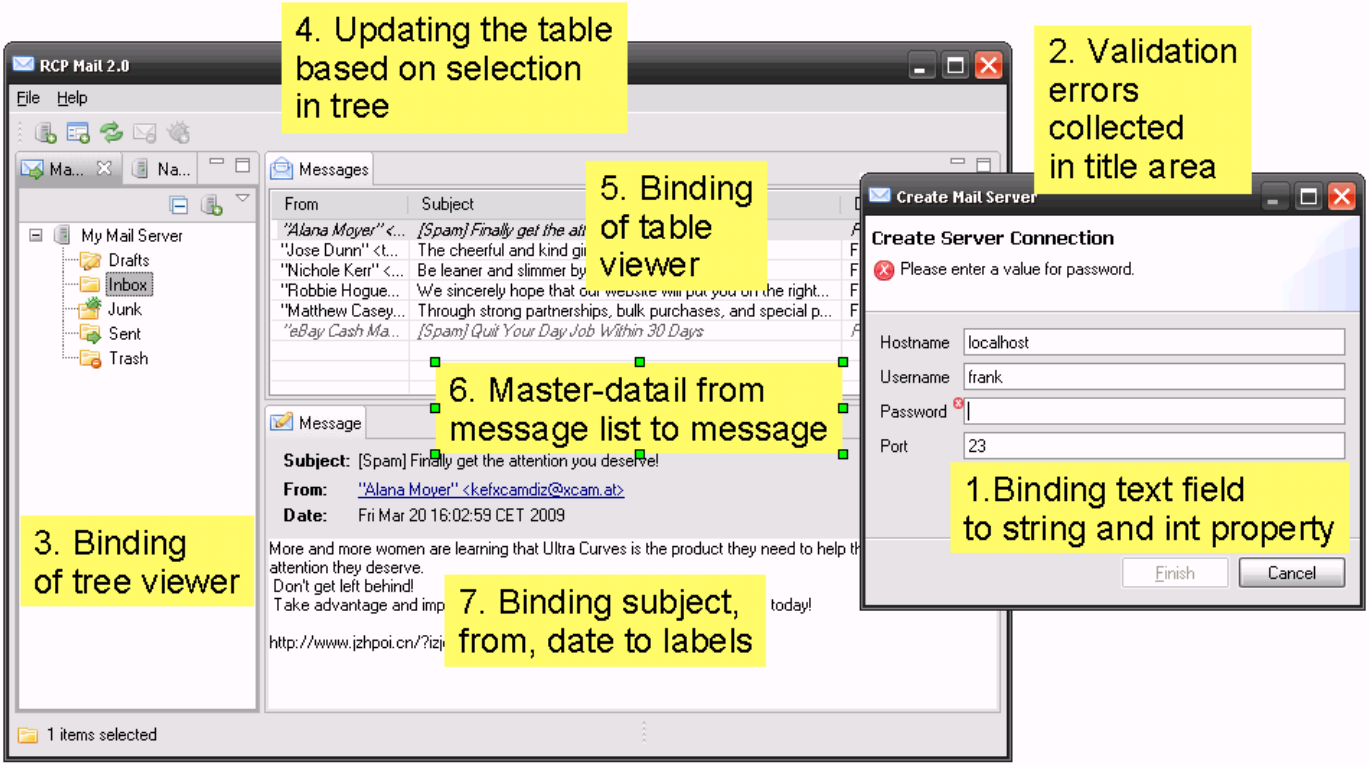
1. WorkbenchObservalbes in org.eclipse.**ui**
2. org.eclipse.**jface**.databinding
3. Four **core** plug-ins
   - org.eclipse.core.databinding
   - org.eclipse.core.databinding.beans
   - org.eclipse.core.databinding.properties
   - org.eclipse.core.databinding.observables



Data Binding Tour

We introduce data binding in four areas

1. **Wizard** to create a new mail server **(new in 2.0)**
2. **Tree** for servers and folders
3. **Table** of messages **(new in 2.0)**
4. Message display



At the end of the data binding tour will be an exercise where you will add your on binding to a text field in the wizard.

Optional: new column, or third level in tree.

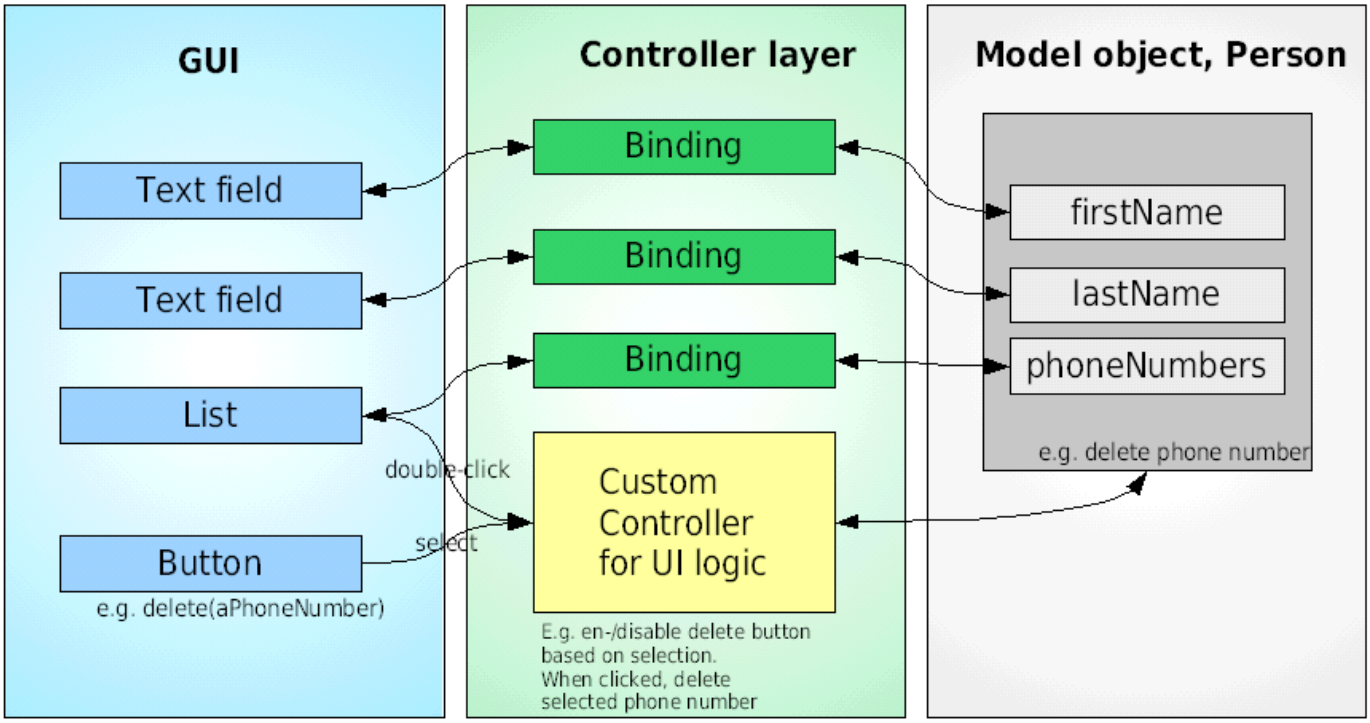
Vision

Get rid of listeners in UI code!

Why?

- Hard to write, hard to maintain.
- For every aspect:
  - Copy initial state into widget.
  - Hook listener (to widget, to model).
  - Write code to sync state incrementally.
  - Validation, conversion typically not separated.
  - Threading.

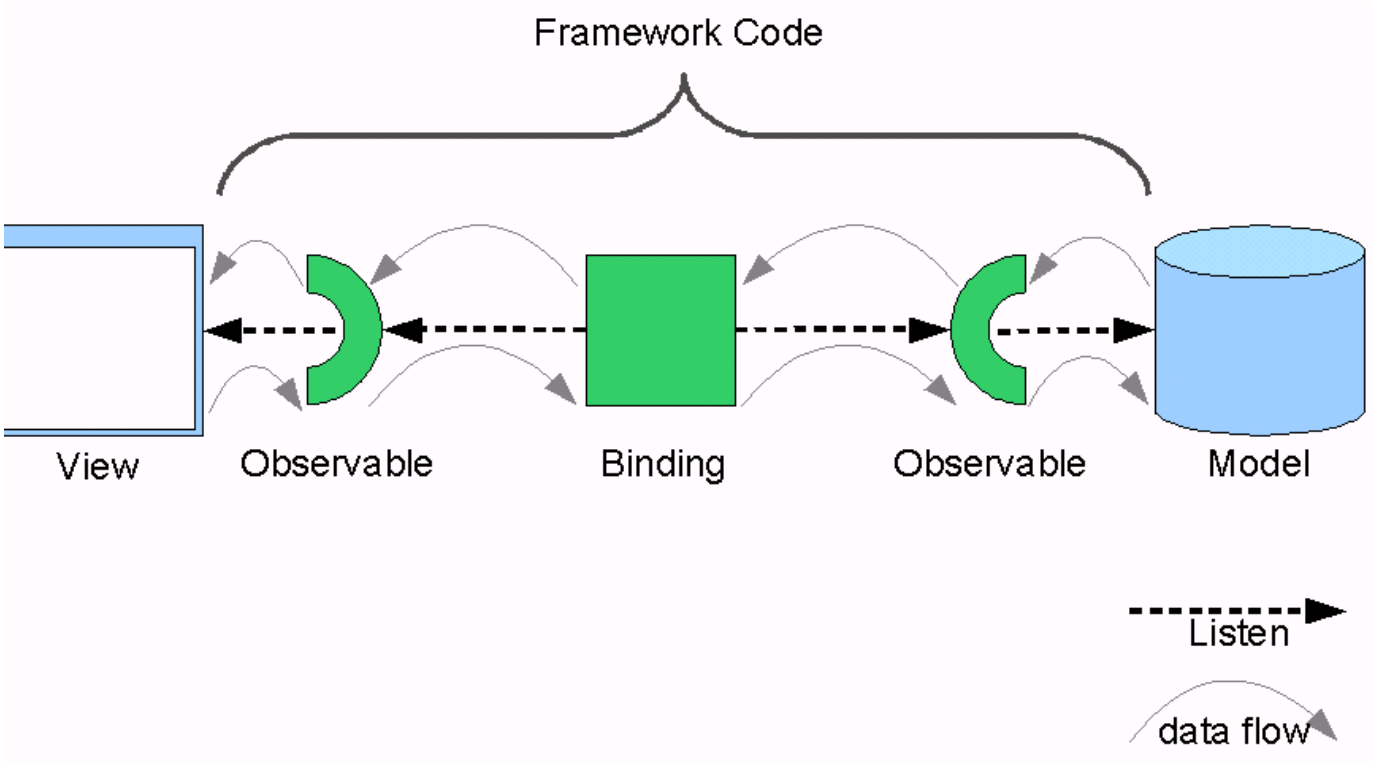
Model View Controller



Concepts

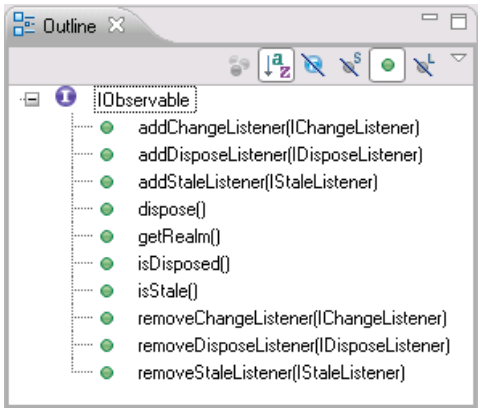
The two main concepts, and *layers* are

- Observables
- Bindings



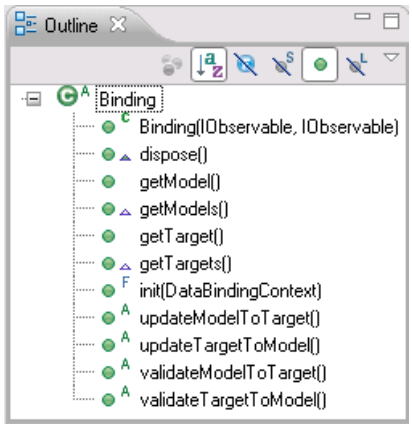
Note: by convention UI always left, model always right, on **diagrams** and in the **API**, see e.g. DataBindingContext, ViewerSupport

IObservable



This interface makes listening to changes uniform

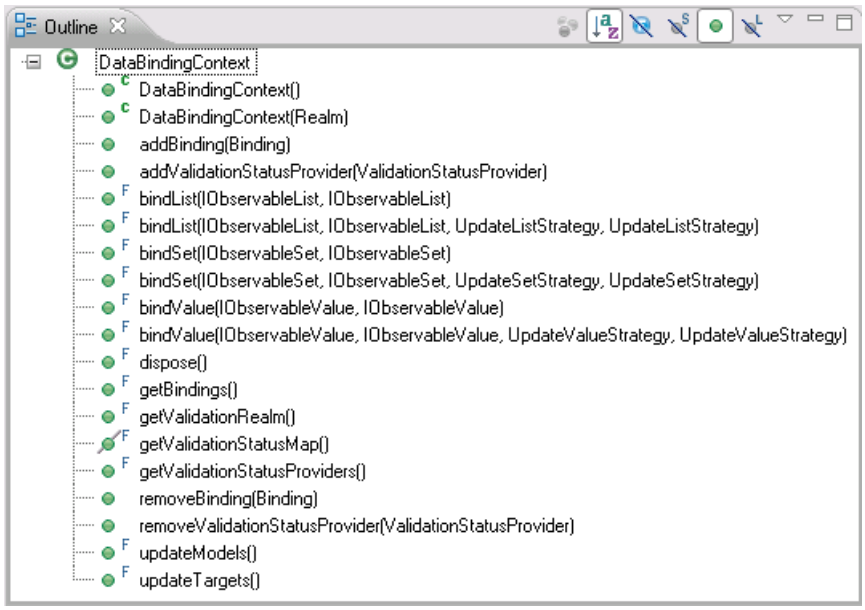
Binding



Represents the binding between two IObservables

Needs to be added to a DataBindingContext

DataBindingContext

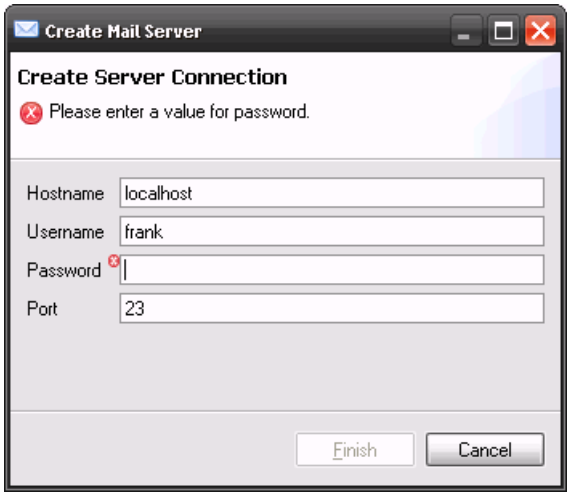


Creation and management of Bindings

Aggregates validation statuses

Goal

We implement a wizard with several text fields



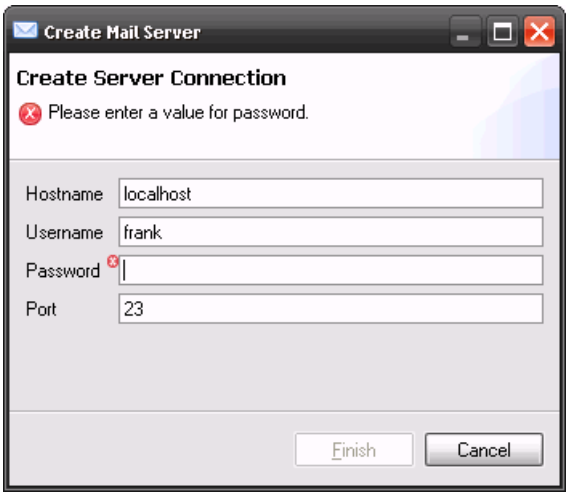
1 minute demo

Plan

We do it in four steps

1. the wizard itself
2. adding data binding
3. adding validation and error messages
4. adding field decorations

We take the first step from the samples and show you the second and third step. We'll take the fourth from the samples again.



The New Server Wizard

We use the MessagePopupAction action and open the wizard from run()

- to create a mailbox
- at first without command (using messagePopup action) and no databinding yet
- only layout and widgets

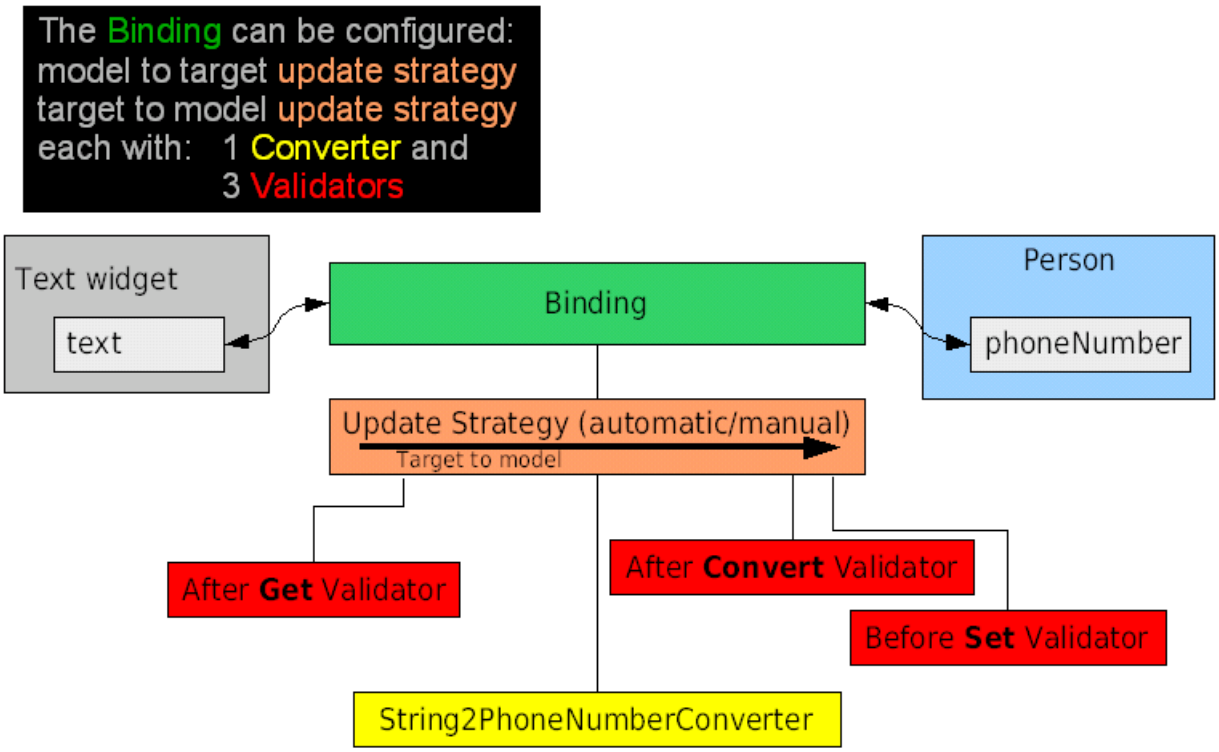
Launch rcpmail-02 to see the wizard

Binding Text Fields

In wizard: binding of text fields to int and string

```
01. DataBindingContext dbc = new DataBindingContext();
02. dbc.bindValue(SWTObservables.observeText(hostnameText, SWT.Modify),
03.                       BeansObservables.observeValue(server, "hostname"));
```

Converters and Validators



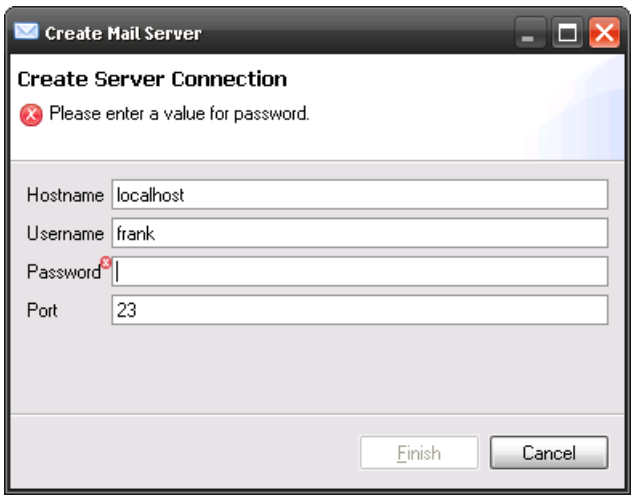

```
01. new UpdateValueStrategy().setBeforeSetValidator(new IValidator() {
02.   public IStatus validate(Object value) {
03.     String s = (String) value;
04.     if (s.contains(" ")) {
05.       return ValidationStatus.error("no spaces please");
06.     }
07.     return ValidationStatus.ok();
08. }
```

Validation

Provided by WizardSupport

Enables Finish button when not errors exist

Control decorations



Used for field validation or other decorations (help available, required field, etc).

Render an image next to a control.

Positioned on the left or right.

Do not take part in layout, no guarantees about clipping. Use margin!

Optional description text when user hovers.

Data binding tree content provider

Goal: For each parent, one IObservableList representing children.

To create these lists lazily, we provide a factory returning lists.

```
01. public IObservable createObservable(Object parent) {
02.     if (parent instanceof Model) {
03.         return BeanProperties.list("servers").observe(parent);
04.     }
05.     if (parent instanceof Server) {
06.         return BeanProperties.list("folders").observe(parent);
07.     }
08.     return null;
09. }
```

Our (invisible) root element is "Model".

BeanProperties.list("servers") is a factory for creating IObservableLists.

Second constructor argument is a "TreeStructureAdvisor".

Is consulted when finding an element in the tree that has not been materialized: notice "polish" when starting app.

Second purpose: optimize for elements without children.

NavigationView

Data binding label provider

Needs to listen to attributes of materialized ("known") elements.

Data binding content provider has getKnownElements().

```
treeViewer.setLabelProvider(new MailLabelProvider(contentProvider
        .getKnownElements()));
```

Label provider needs to know which attributes to listen to.

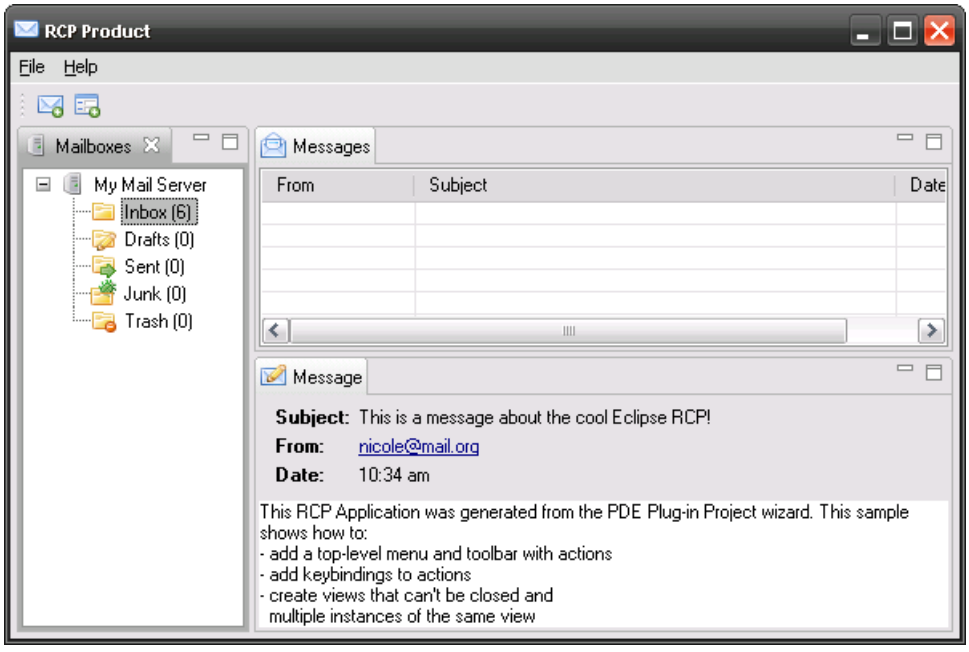
```
MailLabelProvider(IObservableSet knownElements) {
    super(Properties.observeEach(knownElements, BeanProperties
            .values(new String[] { "name", "hostname", "messages" })));
    initializeImageDescriptors();
}
```

Then, just override getText() with computation of label.

```
public String getText(Object element) {
    if (element instanceof Server) {
        return ((Server) element).getHostname();
    }
    if (element instanceof Folder) {
        Folder folder = (Folder) element;
        return folder.getName() + " (" + folder.getMessages().length + ")";
    }
    return null;
}
     <p>Look, Mom, no listeners!</p>
     <p>We will demo later how labels are updated correctly.</p>
```

Goal

We implement a new view with a table of messages



1 minute demo

Plan

What we need to do is

- the view and a TableViewer
- adding data binding to the TableViewer
- adding selection awareness

Table Binding

Similar to Tree Binding

Using `Observable`**`ListTree`**`ContentProvider` and `Observable`**`Map`**`LabelProvider`

Content provider needs an `IObservableFactory`

Label provider needs to know the content provider's known elements

Label provider observer all properties of the known elements

Adding Messages View

Launch rcpmail-07 to see the new view, still with an empty table

It uses a WriteableValue for it's "input", like the MessageTableView

Exercise

Basic: Add validator for port in wizard.

Alternative: Make message subject editable.

Advanced: Add another column to table, e.g. isRead, (using a nested property?)

Keeners: Add messages to tree, under Folders (for fun, not because it makes sense UI wise).

Command Framework

- A command is an abstraction of some semantic behaviour.
- A command is not an implementation of that behaviour. That's a handler.
- A command is not the visual presentation of that behaviour. That's a menu contribution.

```
01. <command categoryId="rcpmail.category"
02.    description="Synchronize with Server"
03.    name="Synchronize" id="rcpmail.syncServer" >
04. </command>
```

Handlers

- A handler implements one behaviour for a command.
- The active handler controls the command's enabled state.
- Handlers most commonly extend AbstractHandler.
- Handlers are provided an application context in their execute(*) method.

```
01. // CreateServerHandler.java
02. public Object execute(ExecutionEvent event) {
03.       WizardDialog dialog = new WizardDialog(HandlerUtil
04.             .getActiveWorkbenchWindow(event).getShell(),
05.             new CreateServerWizard());
06.       dialog.open();
07.       return null;
08. }
```

Menu Contributions

- Menus and Toolbars are important for executing application functionality.
- Menu Contributions create the menu and toolbar structures and insert them into the correct eclipse location.

```
01. <menucontribution locationuri="menu:org.eclipse.ui.main.menu">
02.    <menu id="fileMenu" label="&File">
03.       <command commandid="rcpmail.openNewWindowCommand" icon="icons/silk
    /application_form_add.png" label="New &Window" style="">
04.       </command>
05.       <separator name="rcpmail.separator1" visible="true">
06.       </separator>
07.       <command commandid="rcpmail.command.createServer" icon="icons/silk
    /server_add.png" label="New &Server" style="">
08.       </command>
09.    </menu>
10. </menucontribution>
```

Using Commands

The old version of this RCP app uses ActionFactory to generate Actions to be used in the main menu. In the new version, we'll use commands to create the main menu instead.

First, we'll replace the about action with the equivalent commands.

We will need to place the Open New Window command correctly when we replace the ActionFactory action.

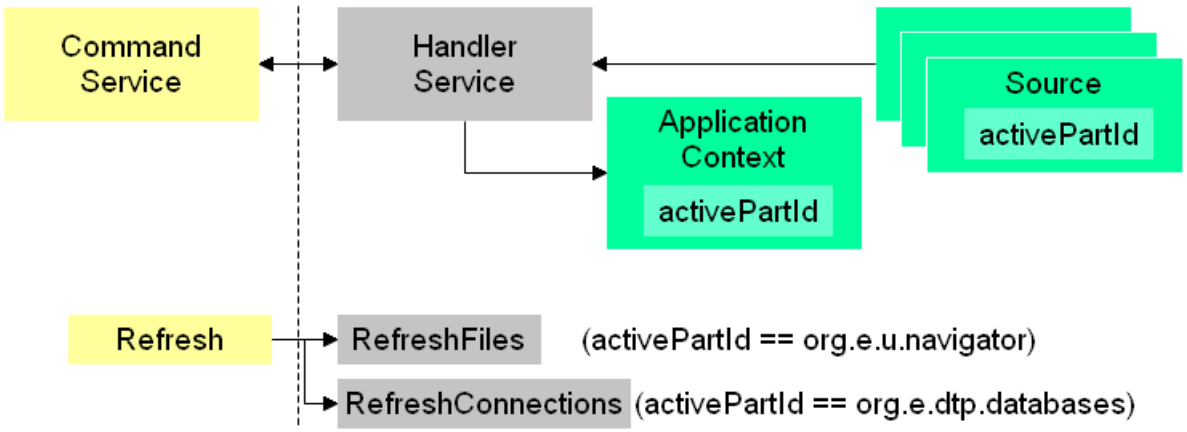
Handlers

- The default handler case is useful, but also uninteresting.
- Multiple handlers can register to handle behaviour for a command.
- At any give time there can be either 0 or 1 active handlers for a command.
- Handlers can be declaratively or programmatically activated.

```
01. // often in createPartControl(Composite)
02. IHandlerService hs = (IHandlerService) getSite().getService(
03.     IHandlerService.class);
04. markAsSpamHandler = new MarkViewAsSpamAndMoveHandler(this);
05. hs.activateHandler(MarkAsSpamAndMoveHandler.MARK_AS_SPAM_COMMAND_ID,
06.     markAsSpamHandler);
```

Handlers - Pick one

Say we want the refresh command to refresh information in the active view:



Handlers

The three most common handler contribution types are:

- A default handler - this is a global handler, usually instantiated once
- Handler for a part - this handler will deal with a specific type of view or editor
- Handler for a selection - this handler is usually keyed off of the selection type

There are a couple of ways to contribute handlers at the different levels:

- Declarative handlers use activeWhen and extract needed state from the application context.
- Programmatic handlers are activated with part creation.

Creating Multiple Handlers

We want to introduce the Mark As Spam command. In the Message View, it should mark the message as spam and move it to the Junk folder. In the Message Table View, it should mark the one or more selected messages as spam and move them to the Junk folder.

If we provide the ability to delete messages it would work in a similar fashion. The message would be moved to the Trash folder.

Evaluating Core Expressions

- The IEvaluationService takes care of handlers activeWhen and enabledWhen, menus visibleWhen, and activities enabledWhen.
- This service is one of the main listeners to source variable changes.
- Expressions that use the changing variable are re-evaluated.
- Most of the variables available from the platform are listed in `org.eclipse.ui.ISources`

```
<enabledWhen>
<with
      variable="selection">
   <iterate
         ifEmpty="false"
         operator="and">
      <instanceof
            value="rcpmail.model.Message">
      </instanceof>
      </iterate>
      </with>
</enabledWhen>
```

Sync With Server Command

The Sync With Server command should be enabled when a single folder is selected. This command is general to the RCP and so can easily be contributed in the plugin.xml.

Active Keybindings

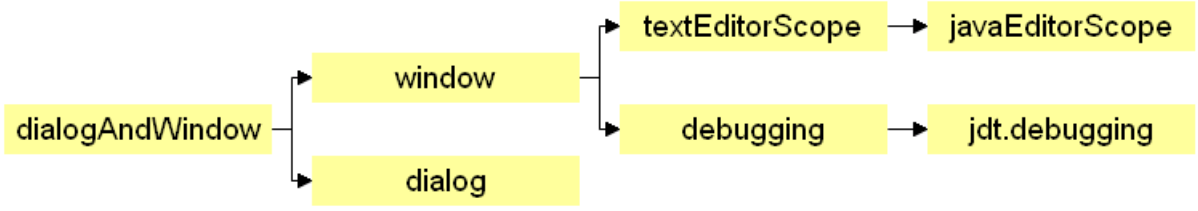
- Different workflows require that some keys operate differently, and in Eclipse that means execute different commands.
- We don't want our keybindings to be overly dynamic, however.
- 2 main ways that keybindings change are scheme and contexts:
  - A scheme is a group of keybindings - very static.
  - A context scopes a keybinding, and contexts change with the program's focus.
- For example, when editing text CTRL+D deletes a line. But when inputting in the Console view, CTRL+D produces the EOF

```

<key
            commandId="rcpmail.syncServer"
            schemeId="rcpmail.key.scheme"
      sequence="CTRL+T">
</key>
```

Active Keybindings - active contexts

- The IBindingService gets the active contexts from the IContextService.
- The IBindingService treats the contexts like an active tree.
- The IBindingService prunes the tree depending on which component has focus.
- This allows the binding service to look up the parameterized command for a key sequence.



Keybindings in the mail App

As with most applications, we want keybindings. There is a choice to make:

- Use the org.eclipse.ui.defaultAcceleratorConfiguration. This makes all of the default

workbench keybindings available.
- Create your own scheme, rcpmail.key.scheme. This gives you control over all of the keybindings in your application.

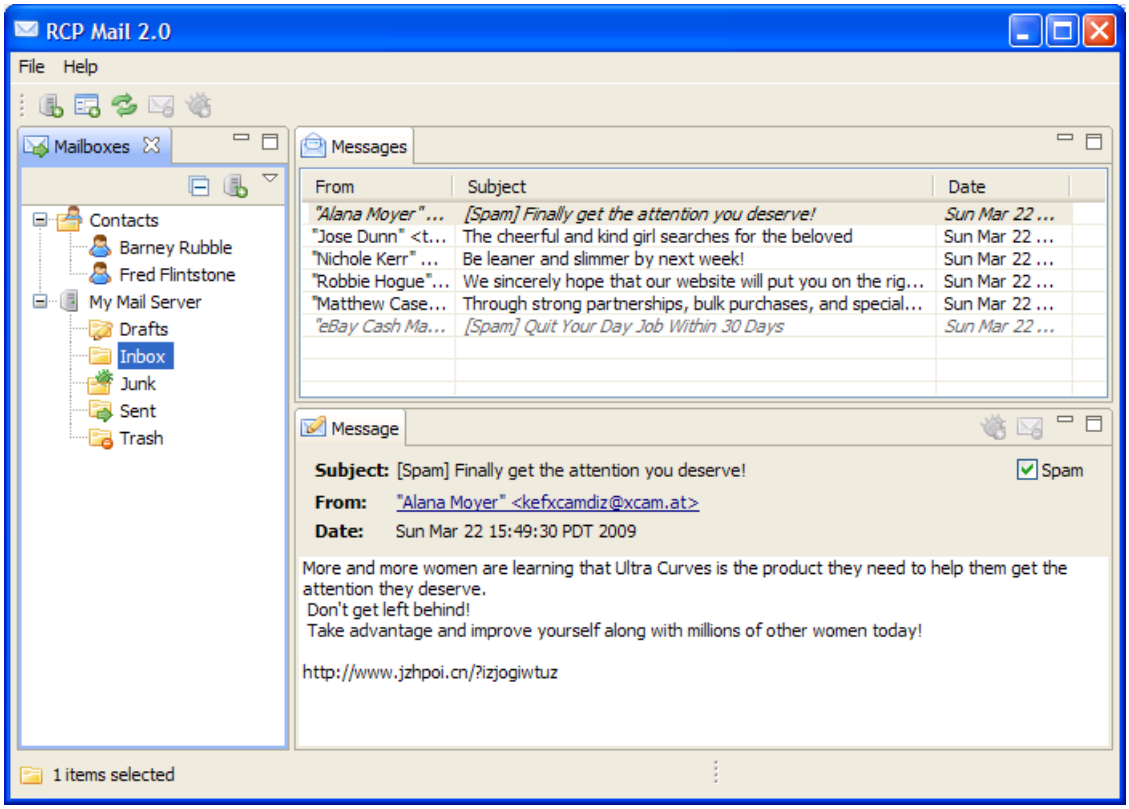
The RCP mail app will use its own scheme to control the keybindings. In this case, it is advisable to copy over the 5 standard keybindings: cut, copy, paste, select all, delete.

Exercise

File/Exit and New Window commands

Common Navigator Framework

Adding Contacts



CNF - Adding Contacts

1. Add contacts to existing RCP mail application.
2. Base plugin unaware and unchanged.
3. Need to hook to the population of the root of the model (the Model object).
4. New Navigator Content Extension is dynamically selected based on the wildcard in the viewer.

CNF - What does it do?

- **Dynamically**

Determined at runtime based on expressions.

- **Hose together different collections of**

  Collections are not related or even known to each other.

- **Content**

  "Content" is content providers, label providers, sorters, filters, drag handler.

CNF - Parts

CommonNavigator - A ViewPart; the Project Explorer is an instance of the CommonNavigator.

CommonViewer - A Viewer.

Navigator Content Extensions - Declare wads of content that can be dynamically enabled.

These content extensions have user visible names and can be dynamically enabled/disabled in the view context menu.

Resource Content Extensions - A set of extensions in `org.eclipse.ui.navigator.resources` that provide the content for resources.

CNF - Implementations

Project Explorer - Main view of the workspace with other content built on resources.

Resource -> JDT, Resource -> CDT, Resource -> JDT -> WTP, etc.

Team Synchronization - Uses the CNF to provide the different types of views, an example of a new of the CNF on an alternate viewer.

RCP Apps that use Resources - Allows RCP model objects to be contributed.

Large RCP Apps - RCP apps that have many sets of content.

CNF - Remaining Steps

Step 12 - Add the CNF to the basic rcpmail application replacing the viewer.

Step 13 - Add the rcpmail.contacts plugin that with the contacts model objects and contact content/label providers.

Step 14 (Exercise) - Add the extension to the rcpmail.contacts plugin hooking it to the CNF.

Step 12: TreeViewer -> CNF

Steps to change the TreeViewer to CNF:

1. Open PDE editor for manifest runtime tab, add new dependency: org.eclipse.ui.navigator.

2. Go to the extensions page and select the org.eclipse.ui.views->Mailboxes view.
3. Change the id to rcpmail.NavigatorView and the class to rcpmail.EditedCommonNavigator. We'll come back to it. Change the name to MailboxesCNF.
4. Change the allowMultiple property to false.
5. Add a new extension: org.eclipse.ui.navigator.viewer
6. Rightclick on it and select New->viewer. Change the viewerId to rcpmail.NavigatorView
7. Rightclick on the org.eclipse.ui.navigator.viewer extension and choose New->navigatorContentBinding. Change the viewerId to rcpmail.NavigatorView
8. Rightclick on it and choose New->(includes), rightclick on it, and select New->contentExtension
9. Change the pattern to rcpmail.* (this allows us to be extensible).
10. Add a new extension: org.eclipse.ui.navigator.navigatorContent.
11. Rightclick on it and choose New->navigatorContent
12. Change it's id to rcpmail.MailboxContent. Change it's name to MailboxContent.
13. Change the contentProvider to rcpmail.MailContentProvider
14. Change the labelProvider to rcpmail.NavigatorLabelProvider
15. Set priority to Normal and activeByDefault to True.
16. Rightclick the MailboxContent contentExtension and choose New->(enablement)
17. Rightclick on (enablement) and choose new instanceof.
18. Set the value to rcpmail.model.ModelObject.
19. Open the Perspective.java and create a new field: `private final String NAVIGATOR_ID = "rcpmail.NavigatorView";`
20. Change the line: `layout.addStandaloneView(NavigationView.ID, false, IPageLayout.LEFT, 0.25f, editorArea); to: layout.addStandaloneView(NAVIGATOR_ID, true, IPageLayout.LEFT, 0.25f, editorArea);`
21. Comment out the line: //layout.getViewLayout(NavigationView.ID).setCloseable(false);
22. Copy MailLabelProvider to NavigatorLabelProvider.
23. In NavigatorLabelProvider, change the superclass to LabelProvider;
24. replace the constructor with a no-arg constructor;
25. and remove the reference to folder.getMessages().
26. At the MailboxesCNF view click on the class (new class wizard) to create a new class extending the CommonNavigator
27. Change the superclass to org.eclipse.ui.navigator.CommonNavigator then click Finish.
28. Now we have to override the getInitialInput() method of the CommonNavigator by adding this to our EditedCommonnavigator class:

```
01.     protected IAdaptable getInitialInput() {
02.         Model.getInstance();
03. }
```

29. Now you can run the plugin by rightclicking it and selecting Run as->Eclipse Application. You can see your brand new CommonNavigator view on the left side with the dummy servers listed.

Step 13: Adding rcpmail.contacts plugin

1. Add model objects for Contacts and Contact.
2. Add content provider ContactsContentProvider:

   `getElements()` returns the Contacts header when it sees the Model object

```
01. public Object[] getElements(Object inputElement) {
02.     if (inputElement == Model.getInstance()) {
03.         return new Object[] { Contacts.getInstance() };
04.     }
05.     return null;
06. }
```

`getParent()` returns:

```
01. if (element instanceof Contact)
02.     return Contacts.getInstance();
03. if (element instanceof Contacts)
04.     return Model.getInstance();
05. }
```

3. Add label provider ContactsLabelProvider.

Step 14: Exercise

Hooking rcpmail.contacts plugin to the CNF

Steps to add the navigator content extention to the rcpmail.contacts plugin

1. Add a new extension: org.eclipse.ui.navigator.navigatorContent.
2. Rightclick on it and choose New->navigatorContent
3. Change it's id to rcpmail.ContactsContent. Change it's name to Contacts Content.
4. Change the contentProvider to rcpmail.contacts.ContactsContentProvider
5. Change the labelProvider to rcpmail.contacts.ContactsLabelProvider
6. Set activeByDefault to True.
7. Rightclick the ContactsContent contentExtension and choose New->(enablement)
8. Rightclick on (enablement) and choose new or.
9. Rightclick on (or) and choose new instanceof.
10. Set the value to rcpmail.model.Model.
11. Rightclick on (or) and choose new instanceof.
12. Set the value to rcpmail.contacts.model.Contacts.
13. Rightclick on (or) and choose new instanceof.
14. Set the value to rcpmail.contacts.model.Contact.

Acknowledgements

This presentation uses the Slideous [4] package by Prof. Stefan Gössner, licensed under the GNU LGPL License 2.1 [5].

External Links

This section lists all hyperlinks included in the presentation. When printing HTML, usually only the blue and underlined hyperlinks are shown and the targets of all hyperlinks are "lost". This handout, when printed (only!), includes a number like a footnote (e.g. [123]) after each hyperlink to refer to the following list of targets.

[1] http://creativecommons.org/licenses/by-nc-sa/3.0/us
[2] http://www.eclipsecon.org/2009/sessions?id=641
[3] http://max-server.myftp.org/rcp-mail/download/rcpmail-downloads.html
[4] http://goessner.net/articles/slideous/
[5] http://creativecommons.org/licenses/LGPL/2.1/