Advanced Eclipse RCP

Kai Tödter, Siemens Corporate Technology
Benjamin Pasero, IBM Rational

Download the Tutorial Material from
http://max-server.myftp.org/mp3m/download/mp3m-downloads.html

© Kai Tödter and others, Licensed under Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 Germany License.

Outline

- Demo: MP3 Manager
- A modular component architecture
- Loose coupling of views and editors
- Internationalization
- Adapter factories
- Virtual trees and tables
- Product & feature branding
- Presentation API
- p2, the new provisioning
- Headless build

© Kai Tödter and others, Licensed under Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 Germany License.
MP3 Manager Project

- Open Source
- Licensed under EPL
- Project Goal
  - Provide show cases and best practices for many common use cases in RCP based applications
- Project Homepage
  - http://max-server.myftp.org/trac/mp3m
  - Anonymous svn access
  - Trac wiki and issue tracking

MP3 Manager Demo
The case for extensibility

- Every complicated application has to be open for extension
  - Generally good practice
  - Better integration with other technologies
  - More business opportunities
  - Way to avoid “proprietary closed application”
  - FUD (Fear, Uncertainty & Doubt)

Extensible vs. Extension based

- Two approaches:
  - Extensible applications
    - E.g. Photoshop, MS Office, Mozilla
    - Full size application core with extension interface
  - Extension based platforms
    - i.e. Emacs, Auto CAD, Eclipse
    - Minimalistic runtime, that includes extension mechanism
      - High level language
      - Extension points mechanism
Extension Based Platforms

- Application core acts as a container for extensions
- All functionality is implemented inside extension modules
  - In case of Eclipse those are Plug-ins (Bundles)
- Advantages
  - More open and transparent
    - Core functionality developers and those who extend applications share same programming approach
  - Easy to replace functionality

Implementing Extensions

- Two ways for Eclipse based applications:
  - Extension registry
  - OSGi Services (whiteboard pattern)
- First one is standard in case of Eclipse
- What to choose depends on actual requirements and use cases
Practical Hints / Advice

- Do not hesitate to define own application specific extension points
- Use your own extension points
  - Avoid “backdoors”
- Put some effort into documenting extension points
  - This will help contributors a lot!
- Take care of compatibility
  - Extension point definitions are contracts between you and those who extend. Respect them!

Modular Component Architecture

- OSGi => modules for the Java platform
  - Highly dynamic and flexible
  - Loose coupling of Java modules
- Modular Component Architecture, based on:
  - OSGi Bundles (= Eclipse Plug-ins)
  - Eclipse Features
    - For deployment options
    - For product lines
    - For different customer brandings
    - For different platforms
Open questions:
- What should be the size of a Bundle?
- What functionality should be provided by a Bundle?
- When to separate functionality into different Bundles?
- How to organize Features?

It depends...
- If you don’t have much experience:
  - Start monolithic, then
  - Separate functionality into different Bundles
    - If it is a self-contained block
      - e.g. domain model, Help, Views, Editors
    - If it has the potential of reuse
      - e.g. Update, Views, Editors
    - If it should be updated separately
  - Separate core and UI functionality into different Bundles
How to organize Features?

- These suggestions are not always the best solution, but might help to get started:
  - Plug-ins which are providing the basic functionality of your RCP application should be grouped in their own Feature
  - Plug-ins with additional / optional functionality should be grouped into separate Features
    - E.g. create a separate Help Feature (see bug 202160, resolved in Eclipse 3.4 😊)
  - Create different Features for different product brandings
    - Create the .product configuration in the Feature project

Benefits of Eclipse’s Plug-in Philosophy

- Through its Plug-in architecture RCP lets you:
  - Decompose your code into loosely coupled units
  - Extend (and update) your product incrementally
  - Enforce contracts between groups in your organization
  - Play nicely with components from other vendors
  - Allow even customers to extend your product
MP3Manager Structure Example

- **Feature**: com.siemens.ct.mp3m.feature.base
  - **Plug-in**: com.siemens.ct.mp3
  - **Plug-in**: com.siemens.ct.mp3m.model
  - **Plug-in**: com.siemens.ct.mp3m.ui.views.physical
  - **Plug-in**: com.siemens.ct.mp3m.ui.views.logical
  - **Plug-in**: com.siemens.ct.mp3m.ui.editors.id3.databinding
  - **Plug-in**: de.ueberdosis.mp3info (third party ID3 tag library)

- **Feature**: com.siemens.ct.mp3m.feature.branding.blue
  - **Plug-in**: com.siemens.ct.mp3m.branding.bue

- **Feature**: com.siemens.ct.mp3m.feature.player
  - **Plug-in**: net.javazoom.jlayer (third party MP3 player library)
  - **Plug-in**: com.siemens.ct.mp3m.ui.player

Outline

- Demo: MP3 Manager
- A modular component architecture
- Loose coupling of views and editors
- Internationalization
- Adapter factories
- Virtual trees and tables
- Product & feature branding
- Presentation API
- p2, the new provisioning
- Headless build
Let JFace viewers be SelectionProvider, so other views can deal with selections not knowing the selection origin.

Example:

```java
TreeViewer treeViewer = new TreeViewer(parent, SWT.BORDER | SWT.V_SCROLL);
getSite().setSelectionProvider(treeViewer);
```

ViewParts that should react on selections just

- Implement ISelectionListener:

  ```java
  public void selectionChanged(IWorkbenchPart sourcePart, ISelection selection) {
      // we ignore our own selections
      if (sourcePart != this) {
          // do something with the selection
      }
  }
  ```

- Register themselves as selection listener:

  ```java
  getSite().getWorkbenchWindow().getSelectionService().
      addSelectionListener(this);
  ```
If you reuse the org.eclipse.ui.editors extension point, use the “extension” attribute

```
static public IEditorPart[] getMp3Editors() {
    IConfigurationElement[] editors = Platform.getExtensionRegistry()
        .getConfigurationElementsFor("org.eclipse.ui", "editors");
    ArrayList<IEditorPart> editorParts =
        new ArrayList<IEditorPart>();

    for (IConfigurationElement editor : editors) {
        try {
            String extensions = editor.getAttribute("extensions");
            if ("mp3".equals(extensions)) {
                IEditorPart editorPart = (IEditorPart) editor
                    .createExecutableExtension("class");
                // ...
            }
        } catch (CoreException e) {
            ...
        }
    }
}
```
Opening Editor in View

```java
class Mp3DoubleClickListener implements IDoubleClickListener {
    public void doubleClick(DoubleClickEvent event) {
        // …
        if (path != null) {
            PathEditorInput pathEditorInput =
                new PathEditorInput(path);
            String editorId = EditorFactory.getDefaultMp3EditorId();
            try {
                getViewSite().getWorkbenchWindow().getActivePage().
                    openEditor(pathEditorInput, editorId);
            } catch (Exception e) {
                LogUtil.logError("com.siemens.ct.mp3m.ui.views.physical",
                    "cannot open editor with id: " + editorId);
            }
        }
    }
}
```

Outline

- Demo: MP3 Manager
- A modular component architecture
- Loose coupling of views and editors
- Internationalization
- Adapter factories
- Virtual trees and tables
- Product & feature branding
- Presentation API
- p2, the new provisioning
- Headless build
Internationalization

- Language specific strings
- Layout of data, like numbers, dates, etc.
- Colors
- Symbols, pictures, icons

- We focus on language specific strings and images

RCP Internationalization

- Strings in application code
- Strings in plug-in XML contributions
- Strings/images in feature brandings
- Strings/images in product brandings
Eclipse provides two mechanisms for string externalization:

- Standard Java ResourceBundles
- Eclipse way
  - Only present in the wizard if the project build path contains the `org.eclipse.osgi.util.NLS` class
  - Usually available in all plug-ins that have a dependency to `org.eclipse.core.runtime`

---

**Example Java Source**

```java
package com.siemens.ct.test.internationalization;

public class Test {
    public Test() {
        String color = "Color";
        String help = "Help";
    }
}
```
import org.eclipse.osgi.util.NLS;

public class Messages extends NLS {
    private static final String BUNDLE_NAME = "test.internationalization.messages"; //$NON-NLS-1$
    public static String Test_color;
    public static String Test_help;
    static {
        // initialize resource bundle
        NLS.initializeMessages(BUNDLE_NAME, Messages.class);
    }
    private Messages() {}
}

public class Messages {
    private static final String BUNDLE_NAME = "test.internationalization.messages"; //$NON-NLS-1$
    private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle.getBundle(BUNDLE_NAME);
    private Messages() {}
    public static String getString(String key) {
        try {
            return RESOURCE_BUNDLE.getString(key);
        } catch (MissingResourceException e) {
            return '!' + key + '!';
        }
    }
}
After String Externalization

- **Standard way:**

  ```java
  public class Test {
    public Test() {
      String color = Messages.getString("Test.color"); //NON-NLS-1$
      String help = Messages.getString("Test.help"); //NON-NLS-1$
    }
  }
  ```

- **Eclipse way:**

  ```java
  public class Test {
    public Test() {
      String color = Messages.Test_color;
      String help = Messages.Test_help;
    }
  }
  ```

---

Messages.properties

- **Standard way:**

  ```none
  Test.color=Color
  Test.help=Help
  ```

- **Eclipse way:**

  ```none
  Test_color=Color
  Test_help=Help
  ```
Benefits of Eclipse Way

- Faster access and initialization
- Better memory footprint
- Easy detection of
  - Missing or unused keys
  - Typos in keys
- Drawback
  - There are now 2 files to maintain and to keep in sync (messages.properties and the Java file)
- More info at

Strings in XML contributions

- In plugin.xml
  - Use localized strings for every attribute that is presented to the end user
  - Use the notion “%key” as attribute value
    - E.g. name="%FileSystemView.title"
  - Provide `plugin_<locale>.properties` for every locale you want to support
    - E.g. `plugin_de.properties`
  - Use the keys and provide translations
    - E.g. `FileSystemView.title=Datei-System View`
Localized Product/Feature Branding

- ... will be covered later in the Branding part

Fragments for I18N

- You could use a plug-in fragment to separate all localization files from the “English” plug-in
- At runtime, all the files will be merged with the host plug-in
Restart the workbench via `PlatformUI.getWorkbench().restart()`

- No API to specify parameters*, some issues with `EXIT.RELAUNCH` and “eclipse.exitdata” property

Workaround:

- Modify `<product>.ini` file: Add/modify two lines:
  - `-nl`
  - `<locale>`, e.g. de

- Benefit: Makes the language change persistent
- Drawback: Does not work with IDE launcher

*See Bug 222023
Outline

- Demo: MP3 Manager
- A modular component architecture
- Loose coupling of views and editors
- Internationalization
- Adapter factories
- Virtual trees and tables
- Product & feature branding
- Presentation API
- p2, the new provisioning
- Headless build

Adapters
In object-oriented software systems, an adapter simply adapts (converts) an object of type A to another object of relevant type B.

Eclipse provides the interface `IAdaptable` to address the adaption of an object:
```
public interface IAdaptable {
    public Object getAdapter(Class adapter);
}
```

Since model objects should not depend on Eclipse, Adapter-Factories can adapt all objects. They don’t have to implement `IAdaptable`...

How does this work?

---

Every JFace viewer relies on
- A `LabelProvider`
- A `ContentProvider`

Example: Tree
- A class implementing `ITreeContentProvider`
- A class extending `LabelProvider`
private final Object[] EMPTY = new Object[] {};

public Object[] getChildren(Object parent) {
    if (parent instanceof Artist) {
        return ((Artist) parent).getAlbums().toArray();
    } else if (parent instanceof Album) {
        return ((Album) parent).getSongs().toArray();
    }

    // Songs have no children
    return EMPTY;
}

A ContentProvider has to deal with all kinds of
domain objects that built up the tree structure

- An AdapterFactory can be registered with the platform
- The factory provides adapters for a given base class
  - This base class does NOT have to implement IAdaptable
- Often, domain specific classes could be handled by
  IWorkBenchAdapters
- IWorkbenchAdapter is a combination of Label &
  ContentProvider
**AdapterFactory in TreeViewer**

```java
treeViewer = new TreeViewer(parent, SWT.BORDER | SWT.MULTI | SWT.V_SCROLL);

IAAdapterFactory adapterFactory = new AdapterFactory();
Platform.getAdapterManager().registerAdapters(
    adapterFactory, Mp3File.class);

treeViewer.setLabelProvider(
    new WorkbenchLabelProvider());

treeViewer.setContentProvider(
    new BaseWorkbenchContentProvider());
```

**IWorkbenchAdapter Example**

```java
private IWorkbenchAdapter entryAdapter = new IWorkbenchAdapter() {
    public Object getParent(Object o) {
        return ((Mp3File) o).getDirectory();
    }

    public String getLabel(Object o) {
        Mp3File entry = ((Mp3File) o);
        return entry.getName();
    }

    public ImageDescriptor getImageDescriptor(Object object) {
        return AbstractUIPlugin.imageDescriptorFromPlugin(ID,
            IImageKeys.MP3);    
    }

    public Object[] getChildren(Object o) {
        return new Object[0];
    }
};
```
Getting the Adapters

```java
public Object getAdapter(Object adaptableObject, Class adapterType) {
    if (adapterType == IWorkbenchAdapter.class
        && adaptableObject instanceof Mp3Directory)
        return directoryAdapter;
    if (adapterType == IWorkbenchAdapter.class
        && adaptableObject instanceof Mp3File)
        return entryAdapter;
    if (adapterType == IPropertySource.class
        && adaptableObject instanceof Mp3File)
        return new Mp3PropertySource((Mp3File)adaptableObject);
    return null;
}

public Class[] getAdapterList() {
    return new Class[] { IWorkbenchAdapter.class, IPropertySource.class
    };
}
```

Benefits

- Loose coupling of domain objects with UI related objects
- No need to explicitly write ContentProviders and LabelProviders
- Reuse of
  - WorkbenchLabelProvider
  - BaseWorkbenchContentProvider
- AdapterFactory might provide several different adapters like IWorkbenchAdapter or IPropertySource
Lab Tasks

- Create a new project
  com.siemens.ct.mp3m.mytreeview
- Reuse the tree model from the project
  com.siemens.ct.mp3m.model
- Implement a Mp3AdapterFactory with adapters for all tree model elements
- Create a JFace TreeViewer and test both the AdapterFactory approach vs. the standard Label- and ContentProvider mechanism

Optional Tasks

- Create a IPropertySource implementation for Mp3File
  - Hint: Take a look at Mp3PropertySource
  - Add an adapter for IPropertySource and Mp3File to your Mp3AdapterFactory
- Add the standard Properties View to the contacts manager application
  - Hint: Add the project org.eclipse.ui.views to your mp3m.product launch configuration
Outline

- Demo: MP3 Manager
- A modular component architecture
- Loose coupling of views and editors
- Internationalization
- Adapter factories
- Virtual trees and tables
- Product & feature branding
- Presentation API
- p2, the new provisioning
- Headless build

Virtual Trees and Tables (1)

- Challenges in many applications:
  - Huge amount of domain specific data has to be displayed in a tree or table
  - Data for the whole tree or table needs either too much memory or takes too much time to create upfront (or even both)
Virtual Trees and Tables (2)

- Solution:
  - Create model data and tree/table items only when they are really needed (e.g. displayed)
  - Keep only the part of the data in memory that is currently displayed
  - Free model data and tree/table items if they are no longer displayed

A Virtual Table

![Virtual Table Image]
public void createPartControl(Composite parent) {
    TableViewer tableViewer = new TableViewer(parent, SWT.VIRTUAL | SWT.BORDER | SWT.V_SCROLL);

    Table table = tableViewer.getTable();
    // ...
    TableColumn column = new TableColumn(table, SWT.NONE, 0);
    column.setText("No");
    column.setWidth(50);

    tableViewer.setItemCount(100000);
    tableViewer.setContentProvider(new LazyContentProvider());
    tableViewer.setLabelProvider(new TableLabelProvider());

    tableViewer.setUseHashlookup(true);
    tableViewer.setInput(null);
}

class LazyContentProvider implements ILazyContentProvider {
    public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {
        this.viewer = (TableViewer) viewer;
        this.viewer.getTable().addListener(SWT.SetData, new Listener() {
            public void handleEvent(Event event) {
                TableItem item = (TableItem) event.item;
                // compute top and bottom index and clear portions
                // of the table to clean up memory
            }
        });

        public void updateElement(int index) {
            // get mp3Info from domain model
            viewer.replace(new Song(index, mp3Info), index);
        }
    }
}
public void createPartControl(Composite parent) {

    treeViewer = new TreeViewer(parent, SWT.VIRTUAL);
    treeViewer.setLabelProvider(new WorkbenchLabelProvider());
    treeViewer.setContentProvider(
        new TreeContentProvider(treeViewer));
    treeViewer.setUseHashlookup(true);

    Mp3Directory root = new Mp3Directory("root");
    // Some initializations...

    treeViewer.setInput(root);
    treeViewer.setChildCount(root, roots.length);
}
class TreeContentProvider implements ILazyTreeContentProvider {

    public void updateElement(Object parent, int index) {
        Mp3Directory parentDir = (Mp3Directory) parent;
        Mp3File mp3File = parentDir.getMp3Files()[index];

        if (mp3File instanceof Mp3Directory) {
            PrefetchModelJob job = new PrefetchModelJob("Update Model", parentDir, index,
                (Mp3Directory) mp3File);
            job.schedule();
        }

        treeViewer.replace(parent, index, mp3File);
        treeViewer.setChildCount(mp3File, 0);
    }
}

Lab Tasks

- Create a virtual table to display a huge list of mp3 files
  - Hint: replicate the existing mp3s in the table
- Implement a Content Provider that implements ILazyContent-Provider
- Implement the updateElement() method properly
Optional Tasks

- Implement the `handleEvent()` method in your `LazyContent-Provider` to clean up table elements that are no longer needed
  - Hint: Take a look at the class `VirtualTableView` in project `com.siemens.ct.mp3m.ui.views.logical`.

Outline

- Demo: MP3 Manager
- A modular component architecture
- Loose coupling of views and editors
- Internationalization
- Adapter factories
- Virtual trees and tables
- Product & feature branding
- Presentation API
- p2, the new provisioning
- Headless build
What is Product Branding?

- Product branding gives your application a specific high-level visual appearance
- Can be used for
  - Vendor-specific appearance
  - Product families
  - Various different editions of the same software basis

What can be branded in RCP apps?

- Launcher’s icon
- Splash screen with progress bar
- Title bar text
- The image the operating system associates with the product
- About dialog image
- About dialog text
- UI presentation style (see Presentation part)
Example Blue Branding

Example Orange Branding
How to create a Branding?

- Create a new product configuration

![Product Configuration Wizard](image1.png)

Separate Branding Plug-ins

- You can create separate branding plug-ins
  - Including product configuration
  - Including all branding resources and information

![Blue Branding Structure](image2.png)  
Blue Branding:
- `com.siemens.ct.mplic.branding.blue`  
- `IRE System Library [d15.6.0.04]`  
- META-INF
  - `about.inf`
  - `about.properties`
  - `build.properties`
  - `icon32x32.gif`
  - `icon32x32.png`
  - `mpl3m.png`
  - `plugin_customization.ini`
  - `plugin.properties`
  - `plugin.xml`
  - `product.lyt.gif`
  - `splash.bmp`

![Orange Branding Structure](image3.png)  
Orange Branding:
- `com.siemens.ct.mplic.branding.orange`  
- `IRE System Library [d15.6.0.04]`  
- META-INF
  - `build.properties`
  - `icon32x32.gif`
  - `icon32x32.png`
  - `mpl3m.png`
  - `plugin_customization.ini`
  - `plugin.properties`
  - `plugin.xml`
  - `product.lyt.gif`
  - `splash.bmp`
Product Branding & Features (1)

Approach 1:
1. Create a feature for each branding
2. Include all plug-ins, that define your product in that feature
3. Place the product configuration in that feature
4. In the product configuration include only the branding feature!

Product Branding & Features (2)

Approach 2:
1. Create a base feature with your application base plug-ins
2. Create a separate feature that contains only the specific branding plug-in
3. Include the application feature in your branding feature
   - Use the “Included Features” tab in the feature.xml editor
4. In the product configuration include only the branding feature!
Internationalized Brandings

- Useful for internationalize product versions
  - Splash screen, images and “about text”
  - Can easily be implemented using plug-in fragments

Localize Splash Screens (1)

- Create a file structure in your localized branding plug-in:
  - nl/<locale>/splash.bmp
- When deploying, use a customized config.ini file, and modify:
  - osgi.splashPath=
    platform:/base/plugins/<original branding plug-in>,
    platform:/base/plugins/<localized branding plug-in>
- Then both plug-ins are in the splash screen search path at startup
Using a custom Splash Handler

- Since Eclipse 3.3 there is a new extension point org.eclipse.ui.splashHandlers
  - Available templates
    - A simulated log-in session
    - An embedded HTML browser
    - A dynamic set of image contributions
- Create a SplashHandler Java class
  - Extend BasicSplashHandler
  - Take a Look at org.eclipse.ui.internal.splash.EclipseSplashHandler

Custom SplashHandler Example

MP3 Manager
An Eclipse RCP Demo Application

Product Version in Custom Splash Handler
Every feature can refer to a branding plug-in
- The feature’s branding data are in the files about.ini and about.properties
- For internationalized feature brandings create plug-in fragments of the branding plug-in
  - Provide the directory structure nl/<locale>
    - E.g. nl/de
  - Provide both about.ini and about.properties for each locale

Example: English Plug-in

- about.ini:
  
  aboutText=%blurb
  featureImage=icon32x32.gif

- about.properties:
  
  blurb=MP3 Manager (English)\n\n\n
  Version: {featureVersion} \n\n\n
  (c) Copyright Siemens AG 2008. All rights reserved.
Example: German Fragment

- **nl/de/about.ini:**
  
  ```
  aboutText=%blurb
  featureImage=icon32x32_de.gif
  ```

- **nl/de/about.properties:**
  
  ```
  blurb=MP3 Manager (Deutsch)
  
  Version: {featureVersion}
  
  (c) Copyright Siemens AG 2008. Alle Rechte vorbehalten.
  ```

Branding Fragment Structure

```com.siemens.ct.mp3m.branding.blue.german
  JRE System Library [jdk1.6.0_04]
  Plug-in Dependencies
  META-INF
  nl
    de
      about.ini
      about.properties
  build.properties
  icon16x16_de.gif
  icon32x32_de.gif
  plugin_de.properties
  product_lg_de.gif
```
German About Dialog

feature brandings in the About dialog

German Feature Branding
Lab Task

- Launch the MP3 manager with
  - Blue branding
  - Orange branding
  - Blue branding in German

Outline

- Demo: MP3 Manager
- A modular component architecture
- Loose coupling of views and editors
- Internationalization
- Adapter factories
- Virtual trees and tables
- Product & feature branding
- Presentation API
- p2, the new provisioning
- Headless build
- Looks great 😊
- But: Looks a bit like the Eclipse IDE

- Looks differently compared to the Eclipse IDE
- Customized for better application usability
What is a Presentation?

- Usually RCP apps contain views and editors
- These views and editors are called parts
- The presentation customizes the layout and Look&Feel of areas containing one or more parts
  - Drawback: Not the whole application's look & feel can be customized with the Presentations API
  - No Look & Feel skinning like in Swing
  - Presentation can provide custom widgets and behavior

Presentation responsibilities

- Control layout and visibility of
  - Parts
  - Menus & Toolbars
  - Drag&Drop regions

- Create the Look & Feel for part stacks
  - Tabs
  - Title
  - Buttons (Close, Maximize, Minimize)
  - Borders
Presentation Activation (Eclipse IDE)

- Go to “General/Appearance” in the Preferences
- Choose a presentation (e.g. “2.1 Style”)
- Restart Eclipse

Activation with Preference

- Create .ini file with content:
  `org.eclipse.ui/presentationFactoryId=<ID>`
- ID is the presentation id, e.g.:
  `org.eclipse.ui.internal.r21presentationFactory`
- Specify program arguments:
  `-plugincustomization <presentation.ini file>`
- Or create default .ini file:
  `plugin_customization.ini`
  - Advantage: Will be detected by the launcher automatically
A Minimal Presentation

- A presentation that only displays a part
  - No Borders, Tabs, Menus
  - Only the top part of the stack is shown

Creating a “Minimal Presentation”

- Create a presentation factory
- Extend org.eclipse.ui.presentationFactories
- Provide class, id and name of your presentation

```xml
<extension
  point= "org.eclipse.ui.presentationFactories" >
  <factory
class="presentation.MinimalPresentationFactory"
id="presentation.MinimalPresentationFactory"
name="Minimal Presentation"/>
</extension>
```
public abstract class AbstractPresentationFactory {

    public abstract StackPresentation createEditorPresentation(
        Composite parent, IStackPresentationSite site);

    public abstract StackPresentation createViewPresentation(
        Composite parent, IStackPresentationSite site);

    public abstract StackPresentation createStandaloneViewPresentation(
        Composite parent, IStackPresentationSite site,
        boolean showTitle);

    // ...
}

public class MinimalPresentationFactory
    extends AbstractPresentationFactory {

    public StackPresentation createEditorPresentation(
        Composite parent, IStackPresentationSite site) {
        return new MinimalPresentation(parent, site);
    }

    public StackPresentation createViewPresentation(
        Composite parent, IStackPresentationSite site) {
        return new MinimalPresentation(parent, site);
    }

    public StackPresentation createStandaloneViewPresentation(
        Composite parent, IStackPresentationSite site,
        boolean showTitle) {
        return new MinimalPresentation(parent, site);
    }
}
Widget Issues

- Widget hierarchy
  - The parts are not children of the presentation!
  - Parts and part toolbars are parented by the workbench
    ➡ Allows moving parts between stacks

- A presentation should not use the part’s control
  - It should use instead: IPresentablePart.setBounds() and IPresentablePart.setVisible()

Presentation Examples

- For details, checkout:
  - Eclipse CVS repository
  - Host: dev.eclipse.org
  - CVS-Root: /cvsroot/eclipse
  - Server: pserver
  - Project: org.eclipse.ui.examples.presentation
  - User: anonymous
  - eclipsecon2005-presentationsAPI.ppt slides are included 😊
Creating a real Presentation

- Useful for:
  - Corporate design or Look&Feel
  - Product branding & product families
  - Application usability

- Think of
  - Drawing borders, visible focus
  - Buttons (Close, Minimize, Maximize)
  - Tab Look & Feel
  - Menus (System, View and Part)

MP3 Manager Presentation (1)

Design Goals:
- Title area with no icons, but gradient fill
- Image-based close button for closable parts
- Button-like tabs, with whole part width
  - Different gradient fills for selections
  - Roll-over effect
  - Better usability for MP3 Manager application
- Since we have a title area, tabs should only be visible if there’s more than one tab
Start the MP3 Manager with
- Default presentation
  - Hint: presentation id = org.eclipse.ui.presentations.default
- MP3M presentation
- Eclipse 3.0 presentation
  - Hint: presentation id = org.eclipse.ui.presentations.r30
Outline

- Demo: MP3 Manager
- A modular component architecture
- Loose coupling of views and editors
- Internationalization
- Adapter factories
- Virtual trees and tables
- Product & feature branding
- Presentation API
- p2, the new provisioning
- Headless build

New Eclipse Provisioning: p2

- P2 is the new Eclipse provisioning system
- Introduced with version 3.4
- Replaced the old update manager
- Fixes many of the update manager’s flaws
- Has many new features (see next slides)
**P2 Features (1)**

- Cleaner end-user workflows
- Faster downloads through multi-threading
- Installers can be run as a regular Java application or using Java Web Start
- Can manage complete installation (.exe, .ini, etc.)
- Can manage and update an Eclipse/RCP instance without running it

**P2 Features (2)**

- Automatically picks the best available mirror
- Automatic retry of downloads
- Sharing of plug-ins across multiple eclipse instances (bundle pooling)
- Easy creation of headless and custom update user interfaces
- Validates plug-in inter-dependencies
Common Use Cases:
- Install an RCP application from a p2 repository
- An RCP application uses p2 metadata and artifact repositories to update itself

Recipe for p2-enabling the mail demo:
1. Create the mail demo (project p2-maildemo)
2. Create a product configuration p2-maildemo.product
3. Add 3 plug-ins to both launcher and product configuration (and added required plug-ins)
   - org.eclipse.equinox.p2.exemplarysetup
   - org.eclipse.equinox.p2.ui.sdk
   - org.eclipse.equinox.simpleconfigurator.manipulator
5. To get the final update work in the installed product, it is also necessary to include the following 3 plug-ins with dependencies in the product configuration:
- org.eclipse.ecf.provider.filetransfer
- org.eclipse.equinox.p2.touchpoint.eclipse
- org.eclipse.equinox.p2.touchpoint.natives

6. Export the product and the metadata/artifact repositories to c:/java/RCP/p2-maildemo

7. Use the director app to install the mail demo from the repository, with the following Program arguments:
   - application org.eclipse.equinox.p2.director.app.application
   - metadataRepository file: c:/java/RCP/p2-maildemo/repository
   - artifactRepository file: c:/java/RCP/p2-maildemo/repository
   - installIU p2_maildemo.product
   - version 1.0.0
   - destination c:/java/RCP/p2-maildemo/install
   - profile MaildemoProfile
   - bundlepool c:/java/RCP/p2-maildemo/install
   - profileProperties org.eclipse.update.install.features=true
     - p2.os win32
     - p2.ws win32
     - p2.arch x86
     - roaming
     - consoleLog

   VM arguments:
   - Declipse.p2.data.area=c:/java/RCP/p2-maildemo/install/p2
RCP P2 Self-Update (4)

8. Start the installed mail demo in c:/java/RCP/p2-maildemo/install
9. Select Help/Software Updates…: Shows the P2 UI with installed product in version 1.0.0
10. Now you want to create a new version 1.0.1 of the product and update the installed version 1.0.0:
11. Update main mail demo plug-in to version 1.0.1
12. Update product version to 1.0.1
13. Export the new product version 1.0.1 in the SAME location, to update the metadata/artifact repositories

14. But, the installed app 1.0.0 does not find any updates… So, add the repository c:/java/RCP/p2-maildemo/repository manually as a new site => the new version 1.0.1 is displayed and ready for update.
15. When you want to install the update, the P2 dialog tells you correctly: “RCP Product is already installed, so an update will be performed instead.”
16. And now, when you click finish, the new version will be installed properly!!!
Further Information

- P2 Eclipse Wiki:

- Equinox/p2/Adding Self-Update to an RCP Application:
  [http://wiki.eclipse.org/Equinox/p2/Adding_Self-Update_to_an_RCP_Application](http://wiki.eclipse.org/Equinox/p2/Adding_Self-Update_to_an_RCP_Application)

- Kai Tödter’s blog about p2-enabling of an RCP application:
Lab Task

- Take a look at the mp3m.product in the project com.siemens.ct.mp3m.feature.blue regarding the dependencies
- Deploy the product and create p2 repositories
- Install the MP3 Manager product using the director application
  - Hint: Use the preconfigured launcher “MP3 Manager Director”
- Add a local p2 repository for update

Optional Lab Task

- Implement new functionality
  - Update bundle version
  - Update feature version
  - Update product version
- Re-deploy the product to the same location
- Update your previously installed MP3 Manager
Outline

- Demo: MP3 Manager
- A modular component architecture
- Loose coupling of views and editors
- Internationalization
- Adapter factories
- Virtual trees and tables
- Product & feature branding
- Presentation API
- p2, the new provisioning
- Headless build

Headless RCP build

- PDE build provides the infrastructure for a headless RCP build
- Many templates and scripts of PDE build can be re-used for your own headless RCP build
- Unfortunately, setting up an headless RCP build is not trivial
The build.properties file specifies common properties needed for the build:

- **product**: the location of the product configuration file
- **baseLocation**: the location of an eclipse install containing all the pre-built features and plug-ins that the product requires in features/ and plugins/ subdirectories. The RCP delta pack is mandatory!
- **buildDirectory**: directory the build will take place in
- **configs**: list the configurations for which you want your product to be built
- **archivePrefix**: the name of the directory of your product once installed on disk
The custom targets are hooks that are invoked during the build by the main script. Examples are:

- clean
- prefetch, postfetch
- preGenerate, postGenerate
- preProcess, postProcess
- preAssemble, postAssemble
- prePackage, postPackage
- test

Create a new plug-in `<namespace>.build` for the build configuration files

Copy the files `build.properties` and `customTargets.xml` from `plugins/org.eclipse.pde.build\<version>/templates/headless-build/` into `build/`

Edit `build/build.properties`.

- product
- archivePrefix
- buildDirectory
- baseLocation
- baseos, basews and basearch
Running the Build (1)

- Precondition for the build: If plug-ins are not fetched from CVS/Subversion, source plug-ins and features must be located in the following structure.

buildDirectory/
  features/
    feature-1/
    feature-2/
    ...
  plugins/
    plugin-1/
    plugin-2/
    ...

Running the Build (2)

- To run the build, execute

  java -jar <eclipse>/plugins/
    org.eclipse.equinox.launcher_<version>.jar
  -application org.eclipse.ant.core.antRunner
  -buildfile <eclipse>/plugins/org.eclipse.pde.build_<version>/scripts/productBuild/productBuild.xml
Add the following properties to the build.properties:

- generate.p2.metadata = true
- p2.metadata.repo=file:${buildDirectory}/repo
- p2.artifact.repo=file:${buildDirectory}/repo
- p2.flavor=tooling
- p2.publish.artifacts=true
- mp3mVersion=3.4.1

Edit/Add the following targets to the customTargets.xml:

- postBuild
- runDirector
<target name="postBuild">
  <antcall target="gatherLogs"/>
  <property file="${buildDirectory}/product.version"/>
  <mkdir dir="${buildDirectory}/result/tmp"/>
  <antcall target="run.director">
    <param name="p2.director.install.path" value="${buildDirectory}/result/tmp/eclipse"/>
    <param name="p2.os" value="win32"/>
    <param name="p2.ws" value="win32"/>
    <param name="p2.arch" value="x86"/>
    <param name="p2.IU" value="com.siemens.ct.mp3m.branding.blue.product"/>
    <param name="p2.version" value="${mp3mVersion}"/>
  </antcall>
  <zip destfile="${buildDirectory}/result/MP3M-p2-RCP-win32-${mp3mVersion}.zip" basedir="${buildDirectory}/result/tmp"/>
  <delete dir="${buildDirectory}/result/tmp"/>
</target>

<target name="run.director">
  <exec executable="${eclipseLocation}/eclipse" failonerror="false" timeout="900000">
    <arg line="-application org.eclipse.equinox.p2.director.app.application"/>
    <arg line="-nosplash"/>
    <arg line="--launcher.suppressErrors"/>
    <arg line="-consoleLog"/>
    <arg line="-flavor ${p2.flavor}"/>
    <arg line="-installIU ${p2.IU}"/>
    <arg line="-version ${p2.version}"/>
    <arg line="-p2.os ${p2.os}"/>
    <arg line="-p2.ws ${p2.ws}"/>
    <arg line="-p2.arch ${p2.arch}"/>
    <arg line="-roaming"/>
    <arg line="-profile MP3MProfile"/>
    <arg line="-metadataRepository ${p2.metadata.repo}"/>
    <arg line="-artifactRepository ${p2.artifact.repo}"/>
    <arg line="-destination ${p2.director.install.path}"/>
    <arg line="-bundlepool ${p2.director.install.path}"/>
    <arg line="-profileProperties org.eclipse.update.install.features=true"/>
    <arg line="-vmargs"/>
    <arg line="-Declipse.p2.data.area=${p2.director.install.path}/p2"/>
  </exec>
</target>

<!-- delete the metadata cache as well as the artifacts for unzipped bundles -->
<delete failonerror="false" includeEmptyDirs="true" dir="${p2.director.install.path}/p2/org.eclipse.equinox.p2.core/cache"/>
</target>
Lab Task

- Install the RCP delta pack to your target platform
- Create a new project
  com.siemens.ct.mp3m.mybuild
- Create copy the files build.properties, customtargets.xml and build.xml from
  com.siemens.ct.mp3m.build
- Adopt build.properties to your environment
- Run the headless build
- Unzip and run the p2-ed MP3 Manager

Further Information

- Andrew Niefer’s blog how to integrate p2 into the build of an RCP application:
  [http://aniefer.blogspot.com/2008/06/example-headless-build-for-rcp-product.html](http://aniefer.blogspot.com/2008/06/example-headless-build-for-rcp-product.html)
Discussion

License

- This work is licensed under the Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 Germany License.

- See [http://creativecommons.org/licenses/by-nc-nd/3.0/de/deed.en_US](http://creativecommons.org/licenses/by-nc-nd/3.0/de/deed.en_US)

- Some slides are based on material of the Eclipse Training Alliance, see [http://www.eclipse-training.net](http://www.eclipse-training.net)