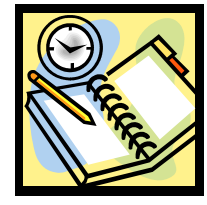
Java Rich Client Platforms: Eclipse RCP compared with NetBeans Platform

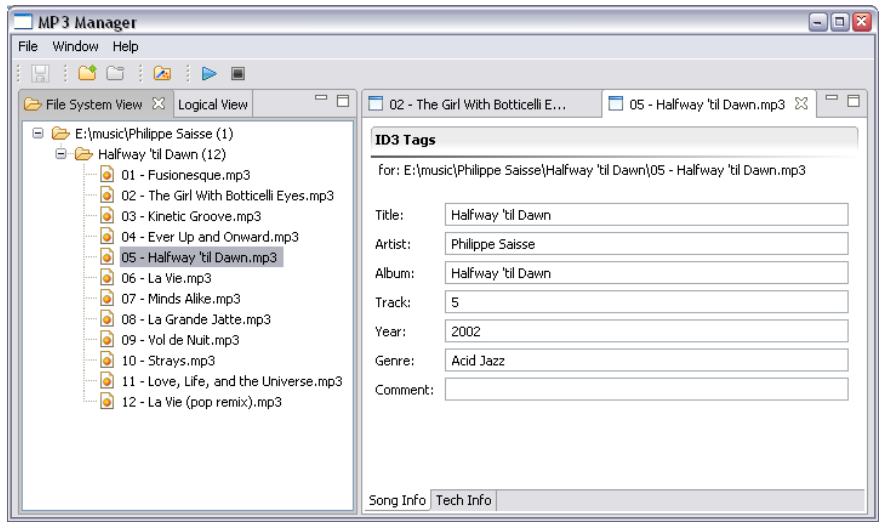
Kai Tödter
Siemens AG
CT SE 2

Outline

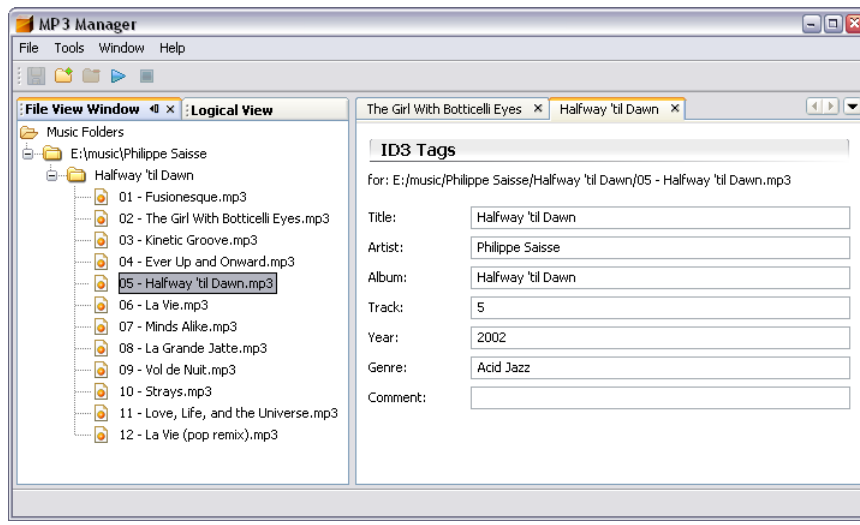
- MP3 Manager: A demo application
- Software Architecture
- Component Model & Module Concept
- UI Toolkits & Customization
- Starting Application Development
- Project Structure
- Actions
- Extension Points & Lookups
- Update Functionality & Help System
- Misc
- Conclusion



MP3 Manager: Eclipse RCP based



MP3 Manager: NetBeans Platform based

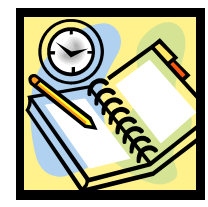


Demo

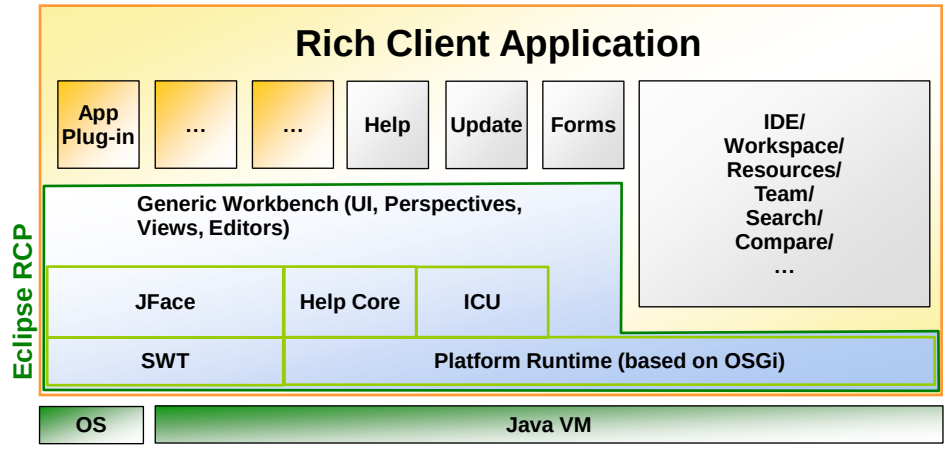
- Showing MP3 Manager on both Platforms

Outline

- MP3 Manager: A demo application
- Software Architecture
- Component Model & Module Concept
- UI Toolkits & Customization
- Starting Application Development
- Project Structure
- Actions
- Extension Points & Lookups
- Update Functionality & Help System
- Misc
- Conclusion

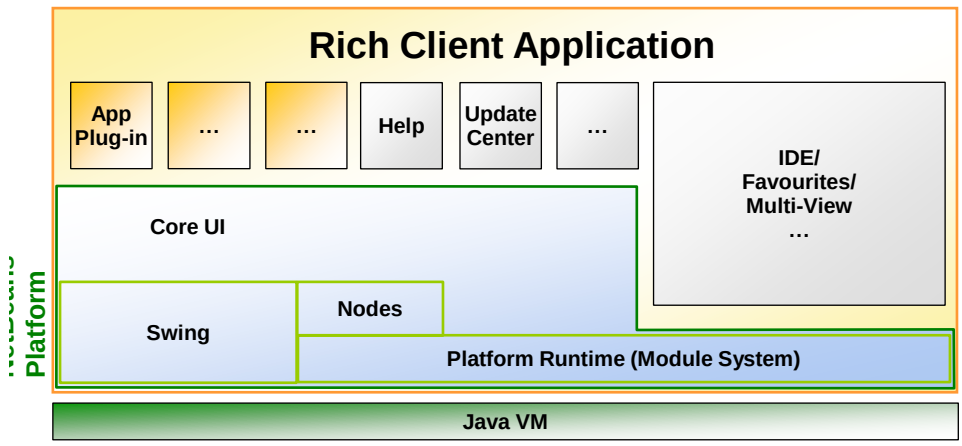


Eclipse RCP based Application



... = Optional reusable component from the Eclipse Platform

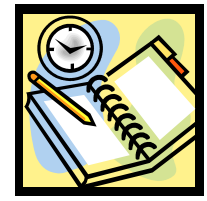
NetBeans Platform based Application



... = Optional reusable component from NetBeans

Outline

- MP3 Manager: A demo application
- Software Architecture
- Component Model & Module Concept
- UI Toolkits & Customization
- Starting Application Development
- Project Structure
- Actions
- Extension Points & Lookups
- Update Functionality & Help System
- Misc
- Conclusion



Eclipse RCP: Based ob OSGi (1)

- Dynamic modules for Java
- Highly adopted standard
- OSGi Bundle is the unit of modularization
  - Eclipse Plug-in == OSGi Bundle
  - Roughly equivalent to a JAR
  - Self-described using MANIFEST.MF metadata

Eclipse RCP: Based ob OSGi (2)

- The OSGi Runtime
  - Manages dependencies and lifecycle of bundles
  - Explicitly supports dynamic scenarios
- Bundles interact through
  - Java package sharing
  - OSGi Service registry
  - Eclipse Extension Registry
- It is possible to run two or more versions of the same bundle in one application

NetBeans Module System (1)

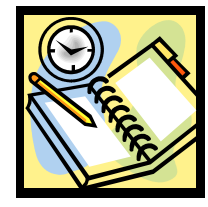
- Dynamic modules for NetBeans Platform
- Proprietary, but
  - Basic idea taken from the Java Extension Mechanism
- NetBeans Module is the unit of modularization
  - Roughly equivalent to a JAR
  - Module attributes in MANIFEST.MF metadata

NetBeans Module System (2)

- The NetBeans Runtime
  - Manages dependencies and lifecycle of NetBeans modules
  - Some support of dynamic scenarios (ModuleInstall)
- NetBeans modules interact through
  - Java package sharing
  - Service registry
  - XML Layer
- It is possible to run two or more versions of the same NetBeans module in one application

Outline

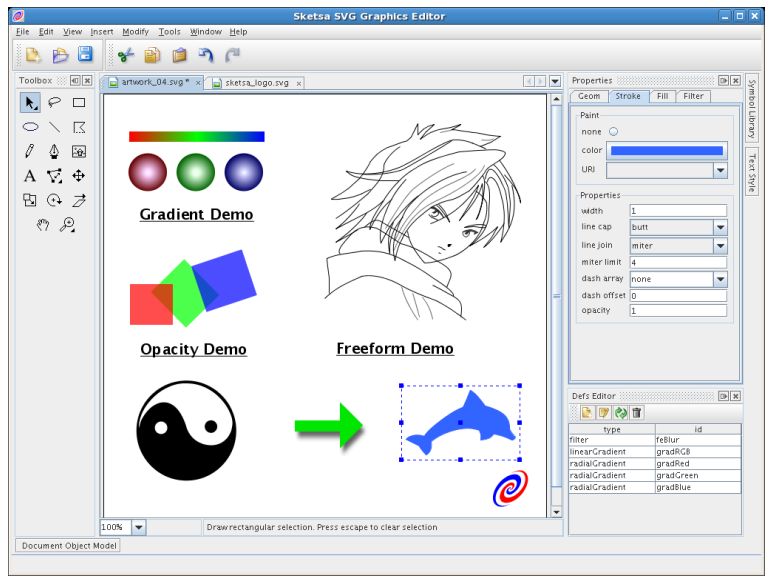
- MP3 Manager: A demo application
- Software Architecture
- Component Model & Module Concept
- UI Toolkits & Customization
- Starting Application Development
- Project Structure
- Actions
- Extension Points & Lookups
- Update Functionality & Help System
- Misc
- Conclusion



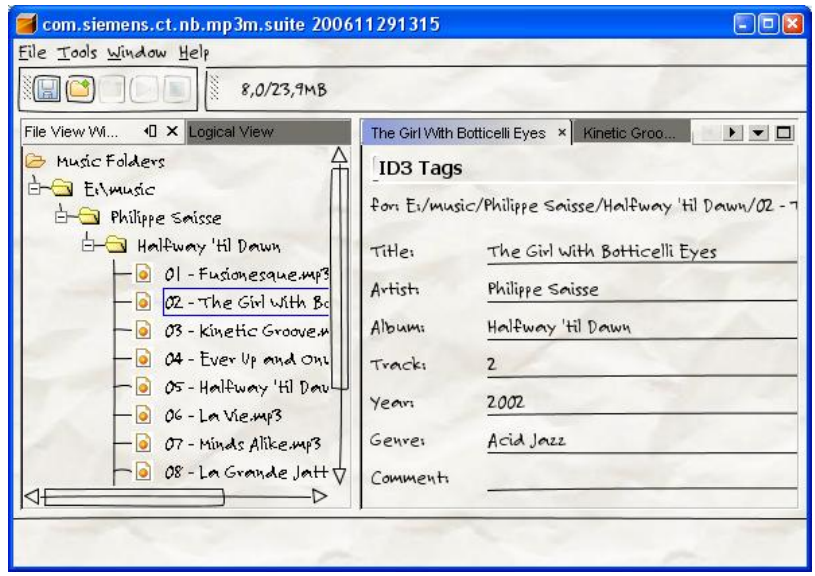
NetBeans UI: Swing

- Very mature UI toolkit
  - But more a widget set than an UI application framework
  - But NetBeans adds framework functionality
- Good performance since Java 1.3
- Java Standard included in the JRE
- Very good free GUI builder Matisse out of the box with the NetBeans IDE
- Native Look & Feels are emulated
  - Since Java 6, native rendering is used if possible
- Very good customizable through the pluggable Look & Feel mechanism

Example of a NetBeans Platform based App



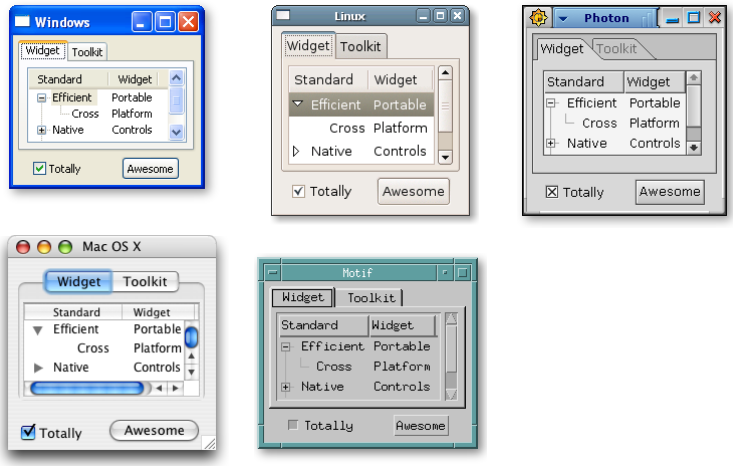
NetBeans MP3M with Napkin Look & Feel



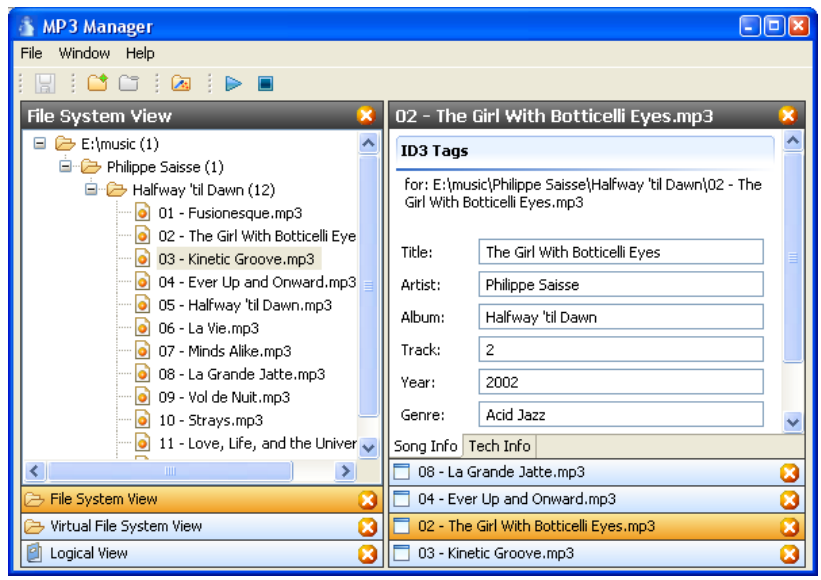
Eclipse UI: SWT/JFace

- Very mature UI toolkit
  - JFace adds UI application framework, with
  - Viewers, Forms, Data binding, Wizards, etc.
- Excellent performance through native widgets
- Some GUI builder as extra components for the Eclipse IDE
  - The very good ones are commercial
- Native Look & Feels
  - Highest OS Look & Feel fidelity
- Partly customizable through Presentation API

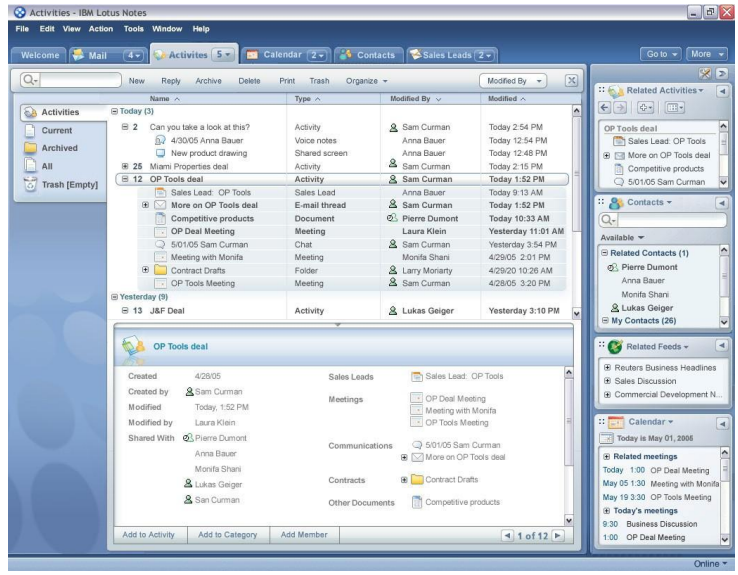
SWT Examples



Eclipse MP3 Manager with customized UI



More SWT Customization: IBM Lotus Notes

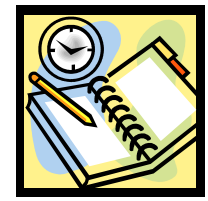


Docking Systems

- A docking system is a windowing system, where the windows can be layouted in several regions
- These regions use tab containers for the containing windows
- The windows can be dragged and dropped into other regions
- The windows can be minimized and maximized
- The windows can be undocked (Stand alone on the OS desktop)
- Both, Eclipse RCP and NetBeans Platform provide excellent docking systems!

Outline

- MP3 Manager: A demo application
- Software Architecture
- Component Model & Module Concept
- UI Toolkits & Customization
- Starting Application Development
- Project Structure
- Actions
- Extension Points & Lookups
- Update Functionality & Help System
- Misc
- Conclusion

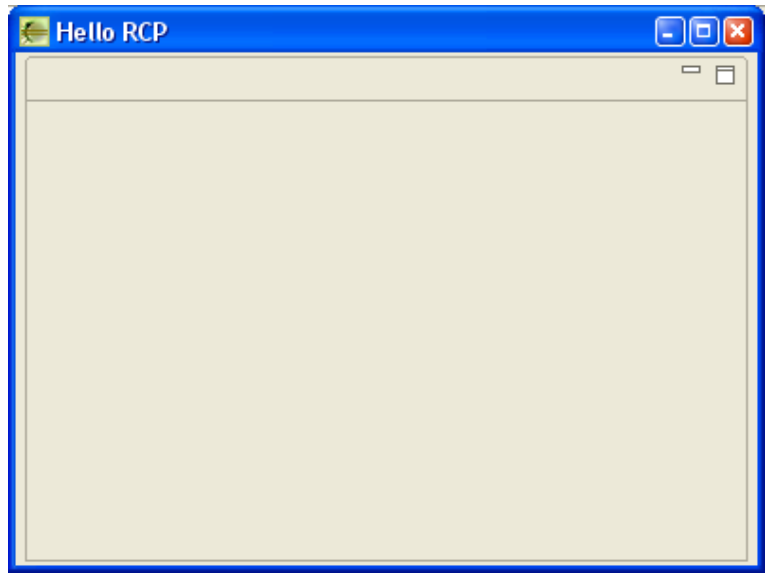


Starting Application Development

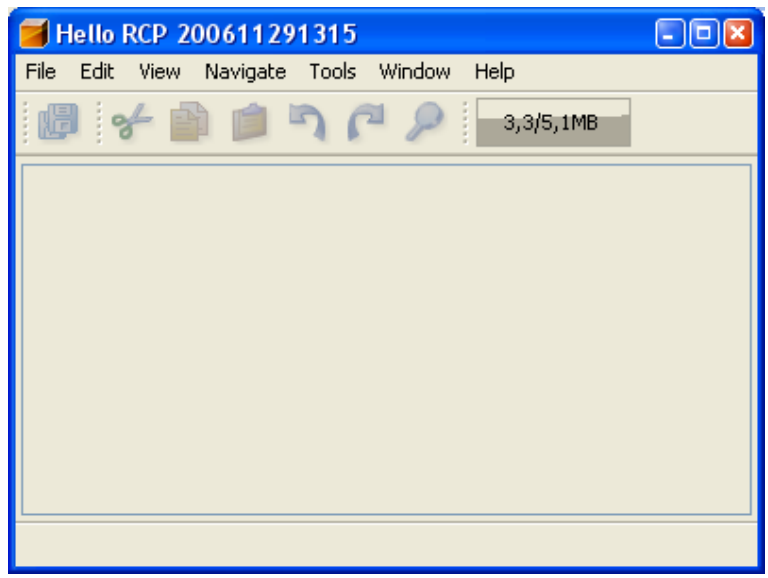
- NetBeans Platform
  - Start with a suite
  - Remove all IDE specific modules
    - Creates Application shown on the next slide
  - Remove all the UI elements you don't want to reuse

- Eclipse RCP
  - Start with a plug-in
  - Choose a RCP template, e.g. Hello World
    - Creates Application shown on the next slide
  - Add new UI contributions

Eclipse RCP Hello World

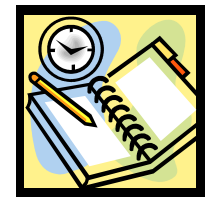


NetBeans Platform Hello World



Outline

- MP3 Manager: A demo application
- Software Architecture
- Component Model & Module Concept
- UI Toolkits & Customization
- Starting Application Development
- Project Structure
- Actions
- Extension Points & Lookups
- Update Functionality & Help System
- Misc
- Conclusion



Eclipse RCP Project Structure

- Plug-ins
  - Provide the functionality
  - Often separation between core and UI plug-ins
- Features
  - Collection of plug-ins that implement the feature's functionality
  - Needed for Update functionality
  - Provide Feature Branding and licensing info
- Product Configuration
  - Can be put in a feature or in a plug-in
  - Contains launching and product branding info

Eclipse MP3Manager Structure Example

- Feature: com.siemens.ct.mp3m.feature.base
  (a container for all the base plug-ins)
  - Plug-in: com.siemens.ct.mp3m
  - 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
  - Plug-in: com.siemens.ct.mp3m.ui.update
  - Plug-in: com.siemens.ct.mp3m.help
  - Plug-in: de.ueberdosis.mp3info (ID3 tag library)
- Feature: com.siemens.ct.mp3m.feature.branding.blue
  (a container only for the branding plug-in)
  - Plug-in: com.siemens.ct.mp3m.branding.bue
- Feature: com.siemens.ct.mp3m.feature.player
  (a container only for the branding plug-in)
  - Plug-in: net.javazoom.jlayer (MP3 player library)
  - Plug-in: com.siemens.ct.mp3m.ui.player

NetBeans Project Structure

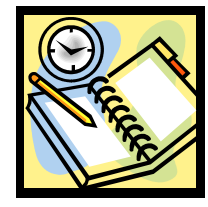
- Modules
  - Provide the functionality
- Suite
  - Collection of modules that implement the application's functionality
  - Provides application branding and licensing info
  - Usually one suite per application

NetBeans MP3Manager Structure Example

- Suite: com.siemens.ct.mp3m.suite
  (a container for all the modules)
  - Module: com.siemens.ct.mp3m
  - Module: com.siemens.ct.mp3m.model
  - Module: com.siemens.ct.mp3m.ui.views.physical
  - Module: com.siemens.ct.mp3m.ui.views.logical
  - Module: com.siemens.ct.mp3m.ui.editors.id3
  - Module: com.siemens.ct.mp3m.ui.update
  - Module: com.siemens.ct.mp3m.help
  - Module: com.siemens.ct.mp3m.ui.player
  - Module: de.ueberdosis.mp3info (ID3 tag library)
  - Module: net.javazoom.jlayer (MP3 player library)

Outline

- MP3 Manager: A demo application
- Software Architecture
- Component Model & Module Concept
- UI Toolkits & Customization
- Starting Application Development
- Project Structure
- Actions
- Extension Points & Lookups
- Update Functionality & Help System
- Misc
- Conclusion



Creating an Action in NetBeans

- Use the “New Action…” wizard
- Fill out the form
- The Action’s Java class template is generated
- The XML layer (layer.xml) for contributing UI to the menu bar and tool bar is created automatically

Action Class in NetBeans (1)

```
public final class AddMusicFolderAction extends
    CallableSystemAction {

   final JFileChooser fc = new JFileChooser();

   public void performAction() {
       // Action’s business logic
   }

   public String getName() {
      return NbBundle.getMessage(AddMusicFolderAction.class,
                                 "CTL_AddMusicFolderAction");
   }
```

Action Class in NetBeans (2)

```
    protected String iconResource() {
        return "com/siemens/ct/nb/mp3m/actions/add_folder.gif";
    }

    public HelpCtx getHelpCtx() {
        return HelpCtx.DEFAULT_HELP;
    }

    protected boolean asynchronous() {
        return false;
    }
}
```

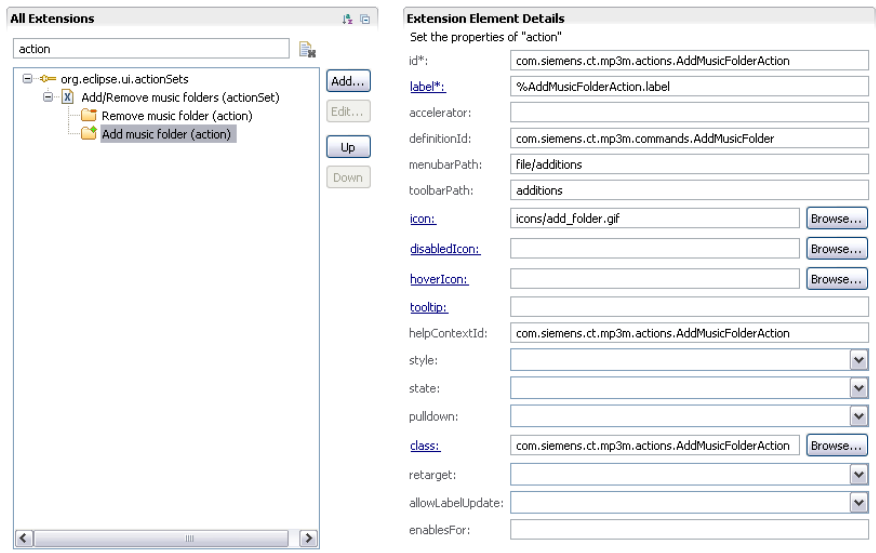

The XML Layer

```
<filesystem>
  <folder name="Actions">
    <folder name="File">
        <file name="com-siemens-ct-nb-mp3m-actions-
                    AddMusicFolderAction.instance"/>
    </folder>
  </folder>
  <folder name="Menu">
    <folder name="File">
      <attr
        name="AddMusicFolderAction.shadow/
              RemoveMusicFolderAction.shadow"
        boolvalue="true"/>
    </folder>
  </folder>/>
  <folder name="Toolbars">
    <folder name="File">
      <file name="AddMusicFolderAction.shadow">
        <attr
          name="originalFile"
          stringvalue="Actions/File/com-siemens-ct-nb-mp3m-
                       actions-AddMusicFolderAction.instance"/>
      </file>
    </folder>
  </folder>
</filesystem>
```

Creating an Action in Eclipse

- Extend the Extension Point “ActionsSets”
  - In Eclipse 3.3 there is a renaissance of the command framework, using commands, handlers and menu contributions.
- Fill out the form
- The Action’s Java class template is generated by clicking the class attribute
- The XML layer (plugin.xml) for contributing UI to the menu bar and tool bar is created automatically

Creating the ActionSets extension

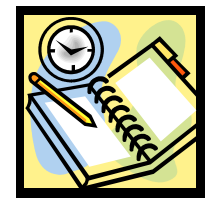


Action Class in Eclipse

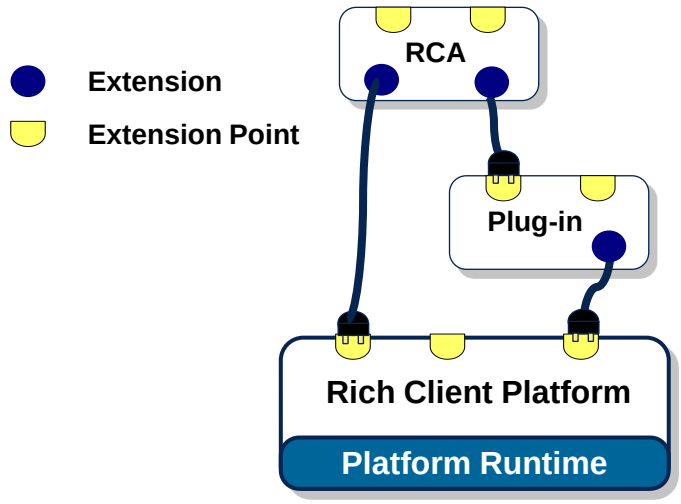
```
public class AddMusicFolderAction implements
                                    IWorkbenchWindowActionDelegate {

   private IWorkbenchWindow window;

   public void run(IAction action) {
      // Action's business logic
   }

   public void dispose() {
      window = null;
   }

   public void init(IWorkbenchWindow window) {
      this.window = window;
   }

   public void selectionChanged(IAction action, ISelection selection) {
      // Selection changes can be handled here
   }
}
```


Outline

- MP3 Manager: A demo application
- Software Architecture
- Component Model & Module Concept
- UI Toolkits & Customization
- Starting Application Development
- Project Structure
- Actions
- Extension Points & Lookups
- Update Functionality & Help System
- Misc
- Conclusion



Eclipse Extensions and Extension Points

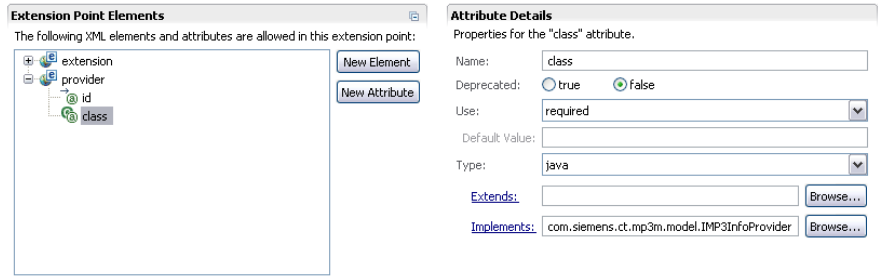


Lazy loading using Eclipse Extension Points

- Create an Interface and expose it to the public
- Create an Extension Point that provides an attribute for the implementing class
- A using plug-in could extent the Extension Point and provide an implementation of the given interface
- The provider of the Extension Point can ckack at runtime, who the extensions are and what to do with them

The Interface and the Extension Point Editor

```
public interface IMP3InfoProvider {
    public IMP3Info getMP3Info();
}
```



The Extensions Check at Runtime

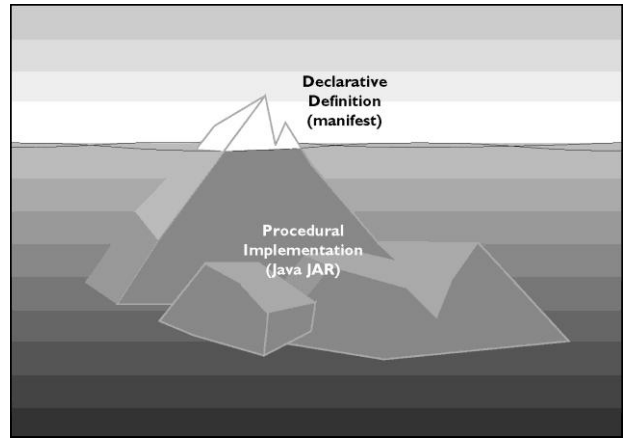
```
IConfigurationElement[] providers = Platform.getExtensionRegistry()
          .getConfigurationElementsFor("com.siemens.ct.mp3m.model",
                                       "mp3info");
for (IConfigurationElement provider : providers) {
   try {
     IMP3InfoProvider provider =
       (IMP3InfoProvider) provider.createExecutableExtension("class");
          // do something useful with the dynamically created class...
   } catch (Throwable e) {
        LogUtil.logError("com.siemens.ct.mp3m.model", e);
   }
}
```

Benefits of the Extension Point Mechanism

- Very good scalability
- Loose coupling of components
  - Using String IDs rather than Java objects
- Very good startup time
  - UI contributions specified in the XML layer are processed at startup
  - Lazy loading of Java classes due to extension check at runtime

The Tip of the Iceberg

- Startup time: O(plug-ins used at startup) rather than O(plug-ins that add UI contributions)



Lazy Loading in NetBeans

- Provide an Interface and expose it to the public
- Create a directory META-INF/services
- Create a file with the fully qualified name of the interface in that directory, e.g. com.siemens.ct.mp3m.model.IMP3InfoProvider
- Put the fully qualified name of the implementing class as content in the file
- The user that checks the global lookup at runtime for implementations of the interface

Using the Global Lookup

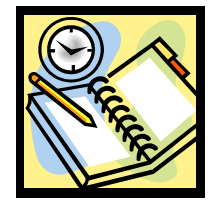
```
IMp3InfoProvider provider =
   (IMP3InfoProvider)Lookup.getDefault().lookup(
         IMP3InfoProvider.class);
```

Benefits of the NetBeans Service Approach

- Very good scalability
- Loose coupling of components
- Easy to use

Outline

- MP3 Manager: A demo application
- Software Architecture
- Component Model & Module Concept
- UI Toolkits & Customization
- Starting Application Development
- Project Structure
- Actions
- Extension Points & Lookups
- Update Functionality & Help System
- Misc
- Conclusion



Integrating Update Functionality

- Both Platforms provide sophisticated Update Mechanisms
  - Eclipse Update using update sites
  - NetBeans Update Center
- Both out of the box update functionalities are often not applicable to domain specific applications
  - Eclipse/NetBeans address experienced software developers

How to create customized update?

- Integrate the out of the box update functionality of the given platform into you application. That lets you test the basic mechanisms and can be done easily with both platforms.
- Then decide, what kind of granularity and complexity you would like to provide in your application.
- Try to reuse some fine granular APIs of the corresponding platform that does the job.

Eclipse Customized Update Action

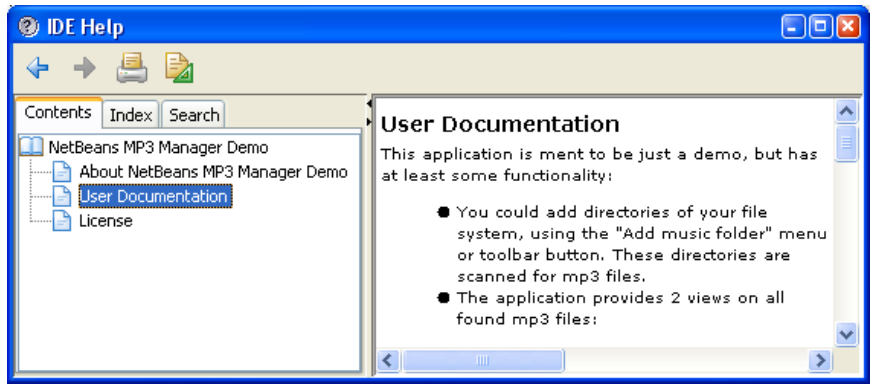
An Update Action that just updates the existing features:

```
protected void run(final IWorkbenchWindow window) {
 BusyIndicator.showWhile(window.getShell().getDisplay(),
    new Runnable() {
     public void run() {
         UpdateJob job = new UpdateJob("Search for updates",
                                                 false, false);
          job.setUser(true);
          job.setPriority(Job.INTERACTIVE);
          UpdateManagerUI.openInstaller(window.getShell(), job);
    }
  });
```

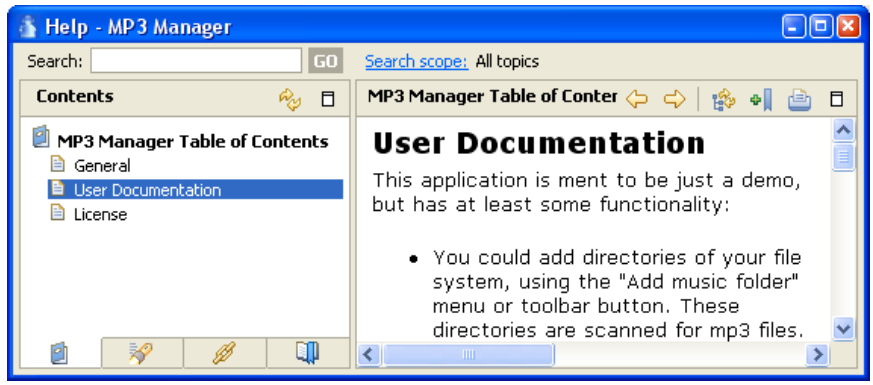

Integrating Help

- NetBeans
  - Using JavaHelp standard
  - Static content only
  - Good & mature help system
  - Help infrastructure is moderate (800 KB)
- Eclipse
  - Using web server and indexing/search engine
    - Currently Tomcat & Lucene, Eclipse 3.3 will use Jetty
  - Static and dynamic content
  - Very good & mature help system
  - Help infrastructure is big (3.7 MB)

NetBeans MP3 Manager Help

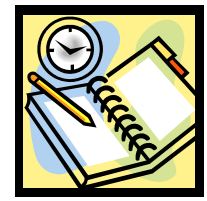


Eclipse MP3 Manager Help



Outline

- MP3 Manager: A demo application
- Software Architecture
- Component Model & Module Concept
- UI Toolkits & Customization
- Starting Application Development
- Project Structure
- Actions
- Extension Points & Lookups
- Update Functionality & Help System
- Misc
- Conclusion



Long Running Operations & User Interaction

- Eclipse:
  - Jobs API
  - Many utilities for asynchronous long running operations in the platform
  - Interactive &Cancelable
- NetBeans:
  - Progress API
  - Interactive & Cancelable

Eclipse Example: IRunnableWithProgress

```
IRunnableWithProgress runnable = new IRunnableWithProgress() {
      public void run(IProgressMonitor progressMonitor)
                        throws InterruptedException {
            int workitemCount = 10;
            progressMonitor.beginTask("Task Name", workItemCount);
            for (int n=0; n<workItemCount; n++) {
                if(!progressMonitor.isCanceled()) {
                    // process a work item
                } else {
                    // do the cancelling cleanup
                    break;
                }
                progressMonitor.worked(1);
             }
             progressMonitor.done();
      }
};
```


Eclipse Example: IRunnableWithProgress (2)

```
try {
    PlatformUI.getWorkbench().getProgressService().
        busyCursorWhile(runnable);
} catch (Exception e) {
    // do something useful
}
```

NetBeans Task Example (1)

```
class  MyJob implements Runnable,Cancellable {
   private boolean isCancelled;

   public boolean cancel() {
      isCancelled = true;
      return true;
   }

   public void run() {
     // do dome initialization
      ProgressHandle handle =
           ProgressHandleFactory.createHandle(
              NbBundle.getMessage(ModelInitializer.class,"Task Name"),
              this);
      int progress = 1;
      handle.start(musicFolders.size());
```


NetBeans Task Example (2)

```
    for (int n=0; n<workItemCount; n++) {
         if(!isCancelled) {
            // process a work item
            handle.progress(progress++);
         }  else {
            // do the cancelling cleanup
            break;
         }
     }
     handle.finish();
   }
}

// Task invocation:
Task task = new Task(new MyJob());
RequestProcessor.getDefault().post(task);
```

Deployment

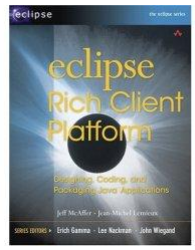
- Both IDEs provide the deployment of the whole application to the local file system
  - NetBeans as ZIP file
  - Eclipse as directory structure, can be zipped
- Java Web Start
  - NetBeans IDE supports direct deployment for Java Web Start
  - Eclipse RCP apps can be made ready for Java Web Start with some manual configuration
- Native installers (e.g. NSIS) can be used in conjunction with both

Licensing

- Eclipse RCP
  - Eclipse Public License (EPL)
    - See http://www.eclipse.org/legal/epl-v10.html
  - ICU4J license
- NetBeans Platform
  - Dual licensed
    - Common Development and Distribution License (CDDL)
    - See http://en.wikipedia.org/wiki/CDDL
    - GPL v2 with Classpath Exception
      - The Classpath exception allows you to link an application available under any license to a library that is part of software licensed under GPL v2, without that application being subject to the GPL's requirement to be itself offered to the public under the GPL.

Eclipse RCP Documentation

- Lots of information at www.eclipse.org
- Good RCP Wiki http://wiki.eclipse.org/index.php/Rich_Client_Platform
- Highly recommended book:

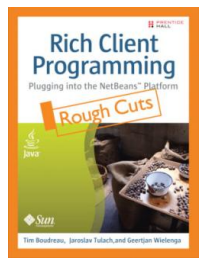


Eclipse Rich Client Platform:
Designing, Coding, and Packaging Java(TM) Applications (The Eclipse Series)

by Jeff McAffer and Jean-Michel Lemieux

NetBeans Documentation

- Lots of information (Articles, Tutorials, etc.) at platform.netbeans.org
- Highly recommended book:



Rich Client Programming: Plugging into the NetBeans Platform

by Tim Boudreau, Jaroslav Tulach, and Geertjan Wielenga

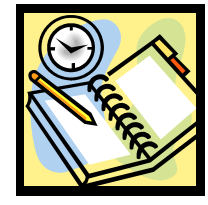
Eclipse RCP MP3 Manager

- Little Eclipse RCP showcase
- http://max-server.myftp.org/trac/mp3m
- Open Source
- Anonymous subversion access
- Trac Wiki and bug-tracking system

- NetBeans Platform MP3 Manager coming soon…

Outline

- MP3 Manager: A demo application
- Software Architecture
- Component Model & Module Concept
- UI Toolkits & Customization
- Starting Application Development
- Project Structure
- Actions
- Extension Points & Lookups
- Update Functionality & Help System
- Misc
- Conclusion



Conclusion

- Both platforms provide
  - Module system with
    - Dependency management
    - Dynamic modules
  - Service infrastructure and lazy loading support
  - Mature UI toolkits with huge widget sets
  - Very good docking system
  - Update support
  - Support for interactive, long running operations
  - Integration of help system
  - And MUCH MORE!

But the Question is…

- Which platform is the better one?
- Which platform should I use?

Recommendation: Get the requirements for YOUR rich client application first! There might be non-functional requirements like scalability, extensibility, reliability, usability and so on as well as functional requirements. After prioritizing the requirements, make your platform choice.

Both platforms Eclipse RCP and NetBeans Platform offer you a lot and help you to build better Java rich client applications!

Discussion