Developing Eclipse Rich-Client Applications 
Tutorial

Dr. Frank Gerhardt
Gerhardt Informatics Kft.
fg@gerhardtinformatics.com

Michael Scharf
Wind River
eclipsecon@scharf.gr
Versions

Version 1.7, March 2008, EclipseCon 2008 (Eclipse 3.3.2)
Version 1.6, March 2007, EclipseCon 2007 (Eclipse 3.3M5eh)
Version 1.5, March 2006, EclipseCon 2006 (Eclipse 3.2M5a)
Version 1.4, September 2005, /ch/open Workshoptage, Zurich (Eclipse 3.1)
Version 1.3, June 2005, iX-Konferenz, Heidelberg (Eclipse 3.1)
Version 1.2, February 2005: EclipseCon 2005 (Eclipse 3.1M5a)
Version 1.1, January 2005, OOP, Munich (Eclipse 3.1M4)
Version 1.0, June 2004: iX-Konferenz, Heidelberg (Eclipse 3.0RC2)
About the Tutorial

- The tutorial is full of hands-on exercises
  - Not a lecture
- These slides provide an architectural overview at the beginning of the tutorial

- Requirements
  - Notebook
  - Java 1.4 or higher installed
  - Eclipse SDK 3.3.2 installed (slight differences from 3.2.x!)

Eclipse Downloads

- Not to scale

These 3 MB are the RCP

- SWT (2.5MB)
- JDT (18MB)
- Eclipse Platform (28MB)
- Eclipse SDK (103MB)
- PDE
- JDT
- IDE Workspace Misc
- RCP

Java VM
Eclipse Architecture

- Rich Client Application
  - Help (Optional)
  - Update (Optional)
  - Text (Optional)

- Other Tools (CDT etc.)
  - PDE
  - JDT

- Generic Workbench (UI)
  - JFace
  - SWT

- Workspace (Optional)

- Platform Runtime (OSGi)

- Java VM
Platform Runtime (org.eclipse.core.runtime)

- Plug-ins
  - The Eclipse synonym to an OSGi bundle
  - Declares the component model
    - Extensions
    - Extension points

- Platform runtime
  - Contains OSGi and the plug-in runtime
  - Is the runtime environment for bundles / plug-ins
  - Responsible for Jobs and preferences
Platform vs. extensible application

Eclipse is a platform. It has a very small kernel.
Eclipse is an Iceberg

- Separation of declaration and implementation
  - Startup time: $O(#\text{used plug-ins})$, not $O(#\text{installed plug-ins})$
Plug-in Namespaces

- Each plug-in has its own classloader
- Requests are delegated to the responsible classloader
- Therefore: Management of plug-in dependencies are essential
SWT & JFace

- SWT
  - Native GUI-Widgets
  - Not OO (e.g. API-compliance between different OS only due to conventions)

- JFace
  - Abstraction over SWT
    - Viewer
    - Forms-API
    - Wizards / Dialogs / Actions
    - MVC / Command Pattern
Workbench

- Workbench
  - Contributes the empty window
  - Adds support for
    - Menubars
    - Toolbars
    - Perspectives
    - Views / editors
    - Preferences
    - And many more extensionpoints
Key Concepts

- Plug-in / Bundle
  - The basic component in Eclipse with **dependencies** to other plug-ins / bundles.
- Extension Points and Extensions
  - The **declarative wiring** between the Eclipse framework and custom code
- Feature
  - A **group** of plug-ins with a version number
- Product
  - A **deployable application** with branding, defined in terms of plug-ins or features
- Update-site
  - A **directory**, usually on a web server, for the initial download and update of features
Tutorial Overview

- Part I: From Hello, World! to Workbench window
- Part II: From RCP Mail to a product with
  - Update Manger
  - Help System
  - Extension Point
HelloEclipse
main()
rcpsimple

HelloEclipse

main()
Developing Eclipse Rich-Client Applications  |  © 2008 by Frank Gerhardt and Michael Scharf

```java
rcpsimple

MyApplication
start()

impl
depends
deploy

myapplication

application

IApplication
start()

impl
depends
deploy

org.eclipse.equinox.app
```
MyApplication
start()

myapplication

IApplication
start()

application

depends

impl

debcles

rcpsimple

depends

impl

debcles

org.eclipse.equinox.app
MyApplication
    run()

myapplication

application
    IApplication
        start()
MyApplication
run()

depends
impl
declares
calls

IPlatformRunnable
run()

application

rcpsimple

create
create

MyWorkbenchAdvisor
getInitialPerspectiveId()

MyWorkbenchWindowAdvisor
preWindowOpen()

MyActionBarAdvisor
makeActions()
fillMenuBar()
fillActionBar()

MyPerspective
createInitialLayout()

MyWorkbenchWindowAdvisor
preWindowOpen()

MyActionBarAdvisor
makeActions()
fillMenuBar()
fillActionBar()

create
create

myapplication

impl

new

MyPerspective
createInitialLayout()

impl

depends

calls

impl

depends

MyWorkbenchAdvisor
getInitialPerspectiveId()

MyWorkbenchWindowAdvisor
preWindowOpen()

myapplication

MyPerspective
createInitialLayout()

impl

depends

calls

impl

depends

PlatformUI
createAndRunWorkbench()

org.eclipse.core runtime

org.eclipse.ui

depends
impl
declares
calls

impl
depends
impl
depends
Attention, Attention!

- Watch us pair-programming
- Listen to us
- Ask us questions
- Try to understand what, why, how
- Observe the way we use the IDE

- Do not work in parallel
  - It is not an exercise in synchronous mouse-clicking
  - You will miss many important points
Exercise: Hello, Eclipse – POJO style

- Write a hello world application in
  - a Java project called rcpsimple
  - Package rcpsimple
- Run it

```java
package rcpsimple;

public class HelloEclipse {
    public static void main(String[] args) {
        System.out.println("Hello, Eclipse!");
    }
}
```
rcpsimple

HelloEclipse

main()
HelloEclipse

main()

rcpsimple

MyApplication

start()

impl

depends

declares

myapplication

application

IApplication

start()

depends

declares

org.eclipse.equinox.app

MyApplication

start()
Exercise: Hello, Eclipse – Plug-in style

- Goal: run Hello, Eclipse! as an RCP application
- Convert the project to a plug-in project
  - update classpath
- Create an application extension
  - Which, where?
  - Search for extension point *application*, browse documentation
- Add extension application, accept to add dependency
  - Uncheck „Show only extension points from the required plug-ins“
    - (alternately declare a dependency on o.e.core.runtime first)
- Define the application extension (with children)
  - run, get ClassCastException, -consoleLog, look at the Eclipse code
  - adapt application class, add IplatformRunnable, move method body
  - Rename HelloEclipse to MyApplication
Optional Exercise: Print out command line params

- Look at context
MyApplication

\texttt{start()}

MyWorkbenchAdvisor

\texttt{getInitialPerspectiveId()}

MyWorkbenchWindowAdvisor

\texttt{preWindowOpen()}

MyActionBarAdvisor

\texttt{makeActions()}
\texttt{fillMenuBar()}
\texttt{fillActionBar()}

MyPerspective

\texttt{createInitialLayout()}

MyWorkbenchWindowAdvisor

\texttt{preWindowOpen()}

MyApplication

\texttt{start()}

IApplication

\texttt{start()}

PlatformUI

\texttt{createAndRunWorkbench()}

rcpsimple

org.eclipse.equinox.app

org.eclipse.ui

myapplication

depends

impl
calls

depends

impl

depends

impl

depends

impl

depends

impl

depends

impl

new

create

create

create
Exercise: Open a Workbench window

- **Goal:** create and run the workbench
  - Clue: PlatformUI.createAndRunWorkbench(display, advisor);
- **Want to search in Eclipse jar files?**
  - Add all plug-ins to Java Search
- **Where is PlatformUI?**
  - Add all plug-ins to Java search, use Open Type dialog
- **Add dependency**
- **Use Quick Fixes to flesh out the method backwards**
- **Which display?**
  - Clue: get display from PlatformUI
- **Which advisor to use?**
  - Roll your own
- **Run it**
  - Update launch config
Optional Exercise: Customize window

- Change size
- Show status line
- Change window title
Exercise: Add a default perspective

- Return a perspective id
  - run (fails)
- Define an extension myperspective
  - Run (works)
  - Window shows up
Exercise: Create a WorkbenchWindowAdvisor

- Clue: Override createWorkbenchWindowAdvisor
  - Roll your own
  - Run
- Change the window size
  - Clue: preWindowOpen
  - Clue: get the window configurer
- Monkey see, monkey do
  - Have a look what others are doing here, browse Eclipse sources
  - Set size, hide the coolbar
  - Run (smaller window, looks nicer)
Exercise: create an ActionBarAdvisor

- Clue: createActionBarAdvisor
  - Roll your own
  - Run
- Monkey see, monkey do
  - Browse subclasses of ActionBarAdvisor
    - Clue: WorkbenchActionBuilder
- Override in MyActionBarAdvisor
  - MakeActions
    - Create quit action
      - Hint: look for an action factory and how it is used by clients
  - fillMenuBar
    - create File menu
- Run
Title

IWorkbenchWindowConfigurer.setTitle()

Cool bar

IWorkbenchWindowConfigurer.setShowCoolBar()

Note
Not shown here:
IWorkbenchWindowConfigurer
setShowPerspectiveBar()
setShowFastViewBars()
setShowProgressIndicator()
Shell size and style

Menu bar

IWorkbenchWindowConfigurer.setShowMenuBar()

Status bar

IWorkbenchWindowConfigurer.setShowStatusLine()
public HyperbolaWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

    HyperbolaWorkbenchWindowAdvisor(IWorkbenchWindowConfigurer configurator) {
        super(configurator);
    }

    public void preWindowOpen() {
        IWorkbenchWindowConfigurer configurator = getWindowConfigurer();
        configurator.setTitle("Hyperbola");
        configurator.setInitialSize(new Point(275, 475));
        configurator.setShowProgressIndicator(true);
        configurator.setShowPerspectiveBar(false);
    }

    public ActionBarAdvisor createActionBarAdvisor(IActionBarConfigurer configurator) {
        return new HyperbolaActionBarAdvisor(configurator);
    }
}
Tutorial Overview

- Part I: From Hello, World! to Workbench window
- Part II: From RCP Mail to a product with
  - Update Manager
  - Help System
Developing Eclipse Rich-Client Applications  |  © 2008 by Frank Gerhardt and Michael Scharf

Diagram:
- rcpmail
  - InstallWizardAction
  - Update
  - Forms

- Plugin A
- Plugin B
- Plugin C

Declarative Definition (manifest)
Procedural Implementation (Java JAR)

???
rcpmail product

rcpmail-feature

rcpmail

InstallWizardAction

Update

Forms

includes

refers to

rcpmail Update Site

Deploy to c:\temp

rcpmail 1.0.0

Rich Client Platform
Developing Eclipse Rich-Client Applications | © 2008 by Frank Gerhardt and Michael Scharf

Deploy to c:\temp

rcpmail 1.0.0

update

rcpmail Update Site

rcpmail 1.0.1

rcpmail product

rcpmail-feature

rcpmail

InstallWizardAction

Update

Forms

Help...

Table of contents

Rich Client Platform

includes

deploy

refers to
Rich client application as product - rcpmail

- An Eclipse product is a stand-alone program
- May be packaged and delivered as one or more features
- Defines the application to run
- Products add branding to applications
Exercise: Generate RCP Mail

- Generate application called "rcpmail" using the RCP mail template
- Observe splash screen and about dialog
- Inspect "product" extension
- Inspect runtime configuration
Workbench Structure

Workbench (1)
  Window (0 – N)
    Page (0 – 1)
      Perspective (0 – N), 1 active
      View (0 – N)
      Editor (0 – N)
Add Update Manager

- Update Manager allows to update features in deployed application
- But, Update Manager UI is not part of RCP
- You can reuse all plug-ins from SDK
- You can add external plug-ins as well
- We have to distribute all plug-ins, which we depend on
Exercise: Add Update Manager

- Import all plug-ins as binary plug-ins
- File search for Find and Install in *.properties files
- Find the user the key
  - Be aware of &
- Copy the extension to rcpmail
- Review and adapt the extension
- Copy resources as needed
- Add dependency
- Configure plug-in localization
- Update launch config
- If it does not run: delete the imported plugins)
Features

- Features group plug-ins and features
- Eclipse is packaged as as set of features
- Configuration manager works with features
- Features can be updated – orphaned plug-ins cannot
Exercise: Create rcpmail-feature

- Update URL
  - file:/temp/rcpmail-site/
  - Do put the slash at the end!!!!
- Add all plug-ins required for Update Manager
  - See launch config
Product Export

- Product export requires
  - Underlying plug-ins / features
  - Config file
  - Launcher
- All components need to be packaged
- Eclipse provides product export wizard for this purpose
- Product configuration file contains necessary information
Exercise: Export product using wizard

- Check launch config
- Add *.properties to Build
- Create product configuration
  - Based on launch config
  - Feature-based, add rcpmail feature
- Generate default config.ini
- Export using the wizard
- Run rcpmail product
Deploy to c:\temp

rcpmail 1.0.0

update

rcpmail Update Site

rcpmail 1.0.1

refers to

deploy

includes

Rich Client Platform

rcpmail product

rcpmail-feature

rcpmail

InstallWizardAction

Update

Forms

rcphelp

Table of contents

Help...

update

refers to rcpmail Update Site

deploy

includes

Table of contents
Exercise: Add Help

- Generate Help plug-in
- Add help to action bar
- Update launch configuration
  - o.e.help.*
  - o.a.lucene
  - o.e.tomcat
- Update dependencies
- Add dependencies to rcpmail feature
Add Update Site

- Configure update site in feature descriptor
- Change your plugin
- Update version to 1.0.1 in plug-in and feature
- Create update site
- Build all
- Start RCP application
- Open Update Manager and update features
- Restart the application
Add an Extension Point for Folders

- Create extension point schema in rcpmail
- Make TreeParent and TreeObject top level public classes
- Export rcpmail package from rcpmail plug-in
- Create new rcpcalendar plug-in
  - Add folder extension
  - Implement rcpcalendar.Calendar
  - Call super constructor with name "Calendar" as parameter
  - Add children, if you like
Optional: Deploy again

- Add rcpcalendar to rcpmail feature
- Increment version numbers of rcpmail plug-in and feature
- Add new version of rcpmail to update site
- Update deployed rcpmail using update manager
Summary: Key Concepts

- **Plug-in / Bundle**
  - The basic component in Eclipse with dependencies to other plug-ins / bundles.

- **Extension Points and Extensions**
  - The declarative wiring between the Eclipse framework and custom code

- **Feature**
  - A group of plug-ins with a version number

- **Product**
  - A deployable application with branding, defined in terms of plug-ins or features

- **Update-site**
  - A directory, usually on a web server, for the initial download and update of features
Discussion
Copyright

- © 2007 Frank Gerhardt and Michael Scharf
- Distributed under Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License http://creativecommons.org/licenses/by-nc-sa/3.0/us
- Contains material from „Developing for the R Rapicault, EclipseCon 2005, licenced under I
- Book: Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications by Jeff McAffer and Jean-Michel Lemieux