



# Single-Sourcing Techniques for RAP and RCP

Frank Appel  
Technical Lead, RAP  
[fappel@innoopract.com](mailto:fappel@innoopract.com)



# Agenda

Basics

Setup

Dependencies

Extensions

API Differences

Lift Off

API Differences II

Multi-User Environment



# Agenda

## Basics

Setup

Dependencies

Extensions

API Differences

Lift Off

API Differences II

Multi-User Environment

# What is RAP?



- rich internet application runtime platform
- based on the Eclipse programming model
- single sourcing for rich client- and web-applications

# Cobbler, stay with your trade



## Single Sourcing

- common codebase for rich- and web-clients
- reuse of existing RCP code
  - 70% - 90% is possible
  - RAP provides only a subset of RCP
  - applications need to become multi-user enabled

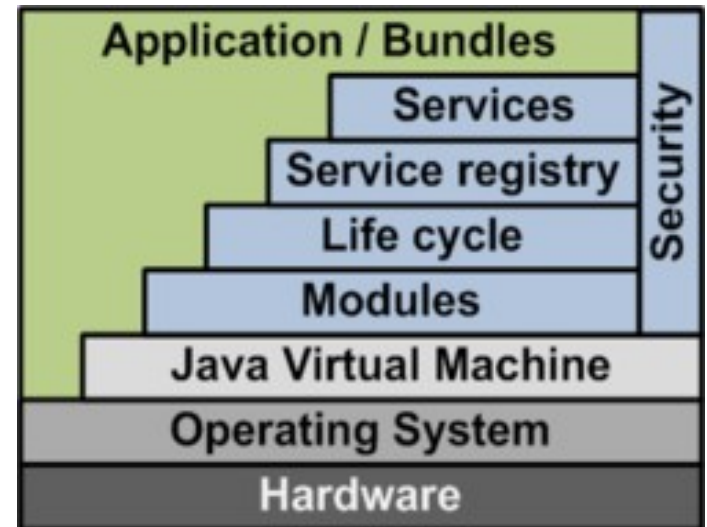
# Groundwork - OSGi



## Plug-ins, Plug-ins, Plug-ins...

OSGi specifies a dynamic component model:

- Module – encapsulation and declaration of dependencies
- Life Cycle – API for life cycle management
- Service Registry – providing functionality to other bundles
- Security layer – limit bundle functionality to pre-defined capabilities

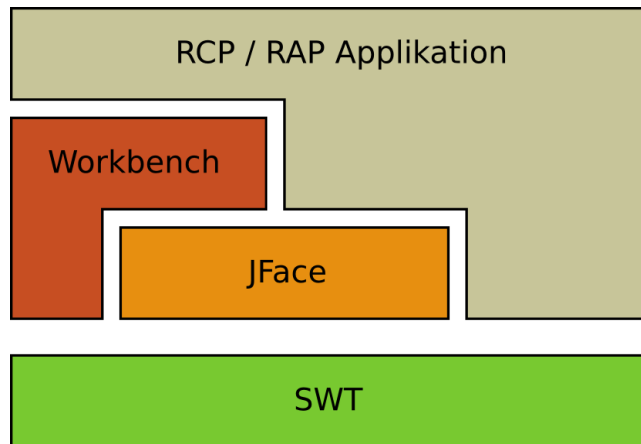


the Eclipse OSGi implementation is provided by the Equinox project

# On the surface



## Contribution to a powerful UI Concept



- **Standard Widget Toolkit (SWT)**  
delivers native widget functionality for the Eclipse platform in an operating system independent manner
- **JFace**  
sits on top of SWT and provides classes for handling common UI programming tasks
- **Workbench**  
is responsible for the presentation and coordination of the user interface

# Best of both worlds



The screenshot displays the RAP Showcase application interface, which is a web-based IDE. The interface is composed of several key components, each highlighted with a colored box and a label:

- Menu Bar:** Located at the top left, containing 'User' and 'Help' menus.
- Tool Bar:** Located below the menu bar, containing icons for file operations (new, open, save, etc.).
- Workbench Window:** The main container for the application, labeled at the top.
- Workbench Parts:** The central area containing the main content, labeled at the top.
- Workbench Page:** The specific page being viewed, labeled at the top.
- Cool Bar:** A vertical sidebar on the left containing a list of tracks and projects.
- Dialog:** A 'Change User Data' dialog box is open, prompting the user to correct input data. It contains fields for Firstname, Lastname, Street, City, Country, Username, Password, and Confirm Password. A message at the bottom states: 'The password and its confirmation are not the same.'
- Map:** A map of Karlsruhe, Deutschland, is displayed in the center.
- Table:** A table titled 'Auftrag/Kosten...' is visible on the right, showing data for 'KT3443233'.
- Form:** A form titled 'Finanzierung' is visible, containing fields for 'Finanzierungskategorie', 'Kategorie', and 'Zu belastende Stelle'.
- Table:** A table titled 'Kosten' is visible on the right, showing data for 'Zugehörigkeit: Alle'.
- Form:** A form titled 'Status (IT-B)' is visible on the right, containing a text area and buttons for 'Signieren' and 'Vor Freigabe prüfen'.
- Table:** A table titled 'Signaturen' is visible on the right, showing data for 'Signiert durch', 'Signiert am', and 'Status'.



# Select a point of view



The screenshot displays the RAP Demo application interface. At the top, a Windows Internet Explorer window shows the 'RAP Demo' page with a menu bar (File, Window, Help) and buttons for 'Open wizard', 'Open new Browser View', 'About', 'Exit', 'Open new editor', 'Save', and 'Save All'. Below this, the application is divided into several views:

- VIEW I**: A tree view showing a hierarchy of nodes including 'Root', 'Locate in browser view', 'EclipseCon location', 'Eclipse Foundation', 'Innooact Inc', 'Parent 2', 'Leaf 4', and 'Child X - filter me!'.
- VIEW II**: A tree view showing a hierarchy of nodes including 'Root', 'Locate in browser view', 'EclipseCon location', 'Eclipse Foundation', 'Innooact Inc', 'Parent 2', 'Leaf 4', and 'Child X - filter me!'.
- VIEW III**: A 'BROWSER VIEW' containing a 3D pie chart titled 'Revenue (in Millions)' and a table titled 'Product Revenue (Best Sellers First)'.

The pie chart shows the following data:

Category	Revenue (in Millions)
Classic Cars	3.85
Motorcycles	1.12
Planes	0.95
Ships	0.66
Trains	0.19
Trucks and Buses	1.80
Vintage Cars	1.02

The table 'Product Revenue (Best Sellers First)' shows the following data:

Product Name	Total Revenue
1992 Ferrari 360 Spider red	\$276,839.98

# Problem Cases



## Differences between RCP and RAP

- RAP runs in a multi-user environment
  - one OSGi instance for all sessions in RAP
  - singletons are shared between sessions
  - no implicit thread to session assignment
  - resources (images, colors und fonts) are shared
- thin-client architecture
  - API limitations (no GC, no MouseMove events)



# Agenda

Basics

**Setup**

Dependencies

Extensions

API Differences

Lift Off

API Differences II

Multi-User Environment

# Hand tools



## Download

- Java Runtime Environment

<http://www.java.com/de/download/>

- Eclipse SDK

<http://www.eclipse.org/downloads/>

- RAP SDK

<http://www.eclipse.org/rap/downloads/>

## Eclipse Distributions z.B.

<http://ondemand.yoxos.com/>

The screenshot shows the 'yoxos on demand' website, which is a 'free eclipse download service'. The main window displays a search for 'rap' in the 'Plug-In Explorer'. The search results show a list of features to be added to the download, including 'Eclipse Development', 'Eclipse Project SDK', 'Java Development', 'Eclipse Java Development Tools', 'Eclipse Plug-in Development Environment', 'Rich Ajax Platform SDK (RAP)', 'Runtime', 'Source Code Management', and 'Sources'. The estimated download size is 189.38 MB. The 'Autoselect' section shows 'Sourcecode and developer resources' selected. The 'Information' tab shows details for the 'Rich Ajax Platform SDK (RAP)' distribution, including its ID, version, provider, and download size.

Name	Rich Ajax Platform SDK (RAP)	Popularity	★★★★★
ID	org.eclipse.rap.tooling	Provider	Eclipse.org
Version	1.1.1.20080917-1637	Download Size	24.51 MB
		Environment	

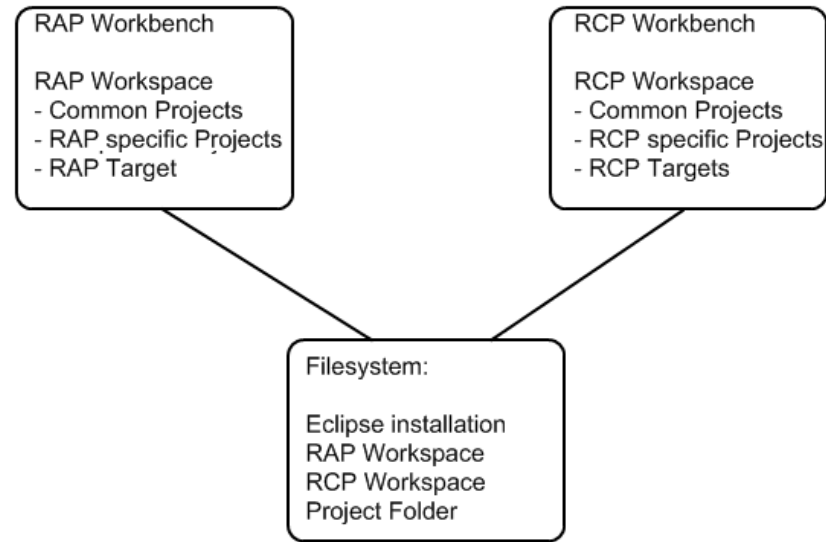
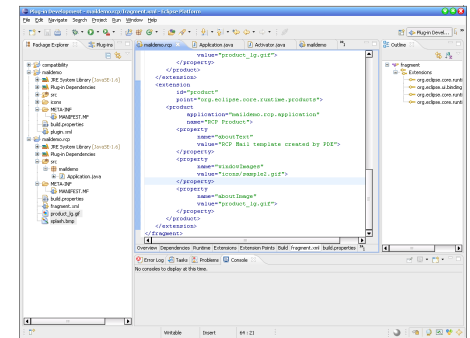
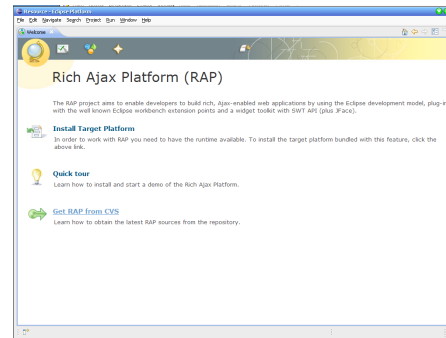
Direct Access Link: <http://ondemand.yoxos.com/geteclipse/start?features=org.eclipse.rap.tooling>

# Workplace



## 2 Workbench Instances

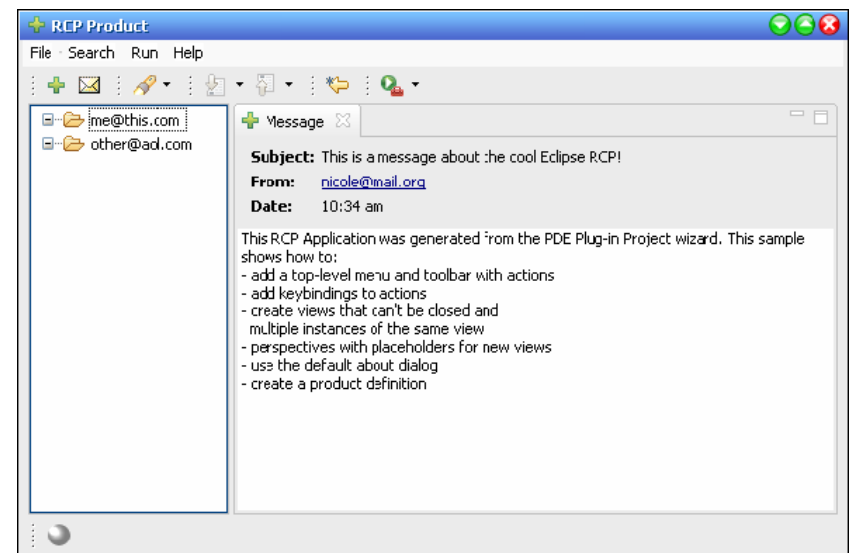
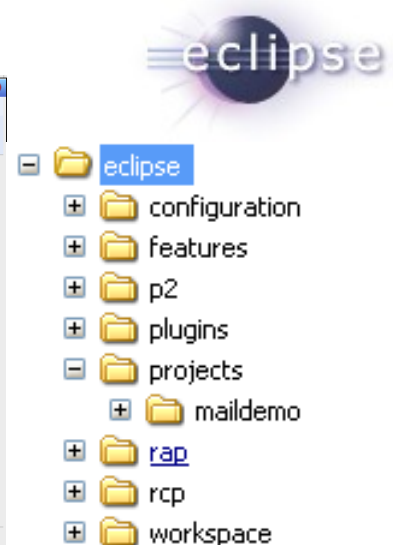
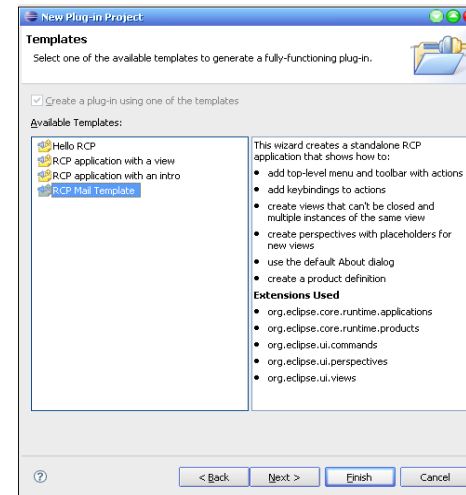
- RAP and RCP need different targets
- switching a target is time-consuming because the complete workspace is recompiled



# The Example Application

## RCP Mail Demo

- created by using the new plug-in project wizard in the RCP workbench
- filed in a common projects folder
- created as Rich Client Application
- runs immediately

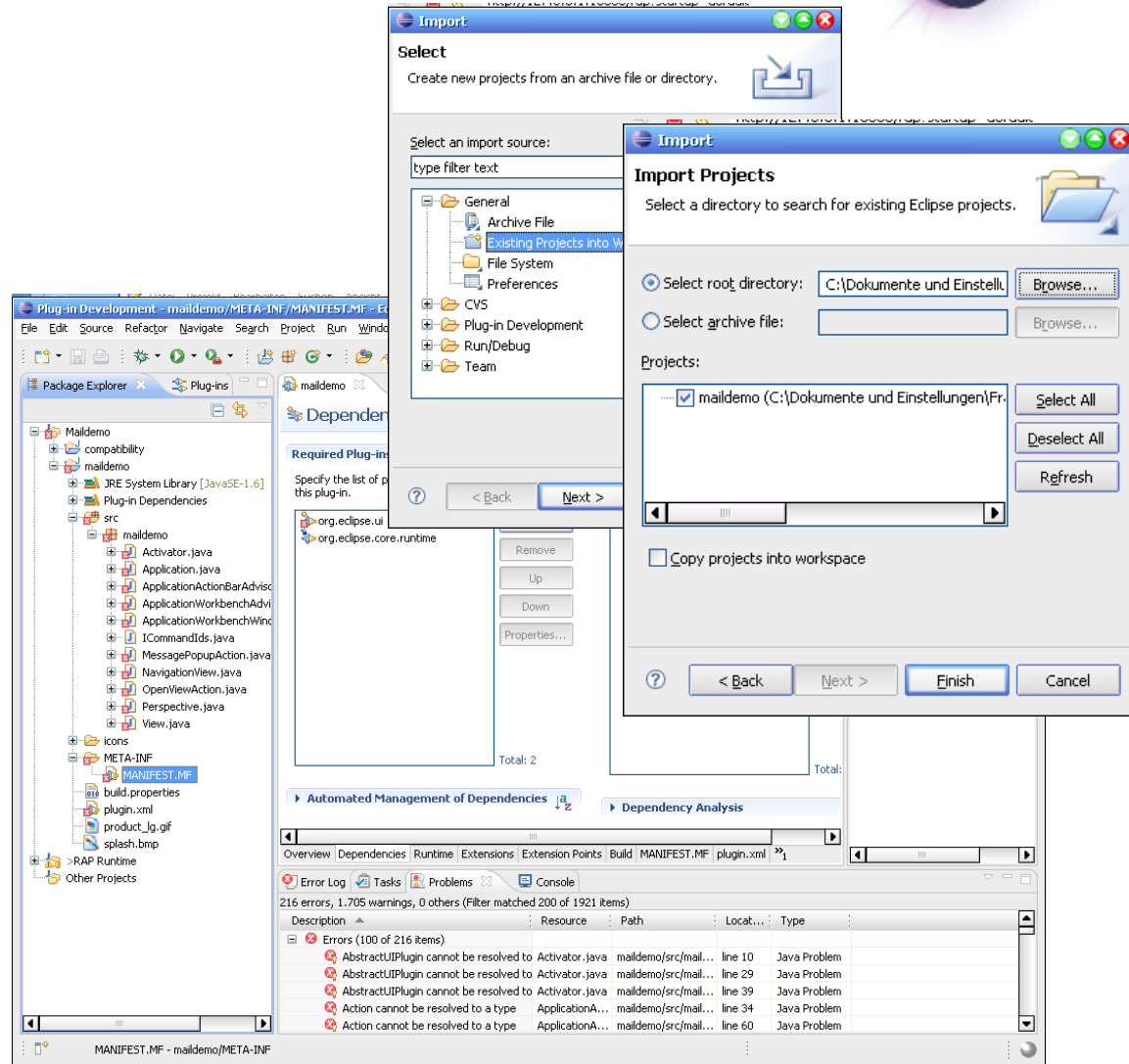


# Import to the RAP Development Environment



## RAP Mail Demo

- import using the import project wizard
- after import 216 error markers
- step by step conversion to support both runtimes





# Agenda

Basics

Setup

**Dependencies**

Extensions

API Differences

Lift Off

API Differences II

Multi-User Environment



# Dependencies



## **The problem of different UI libraries**

possible solutions:

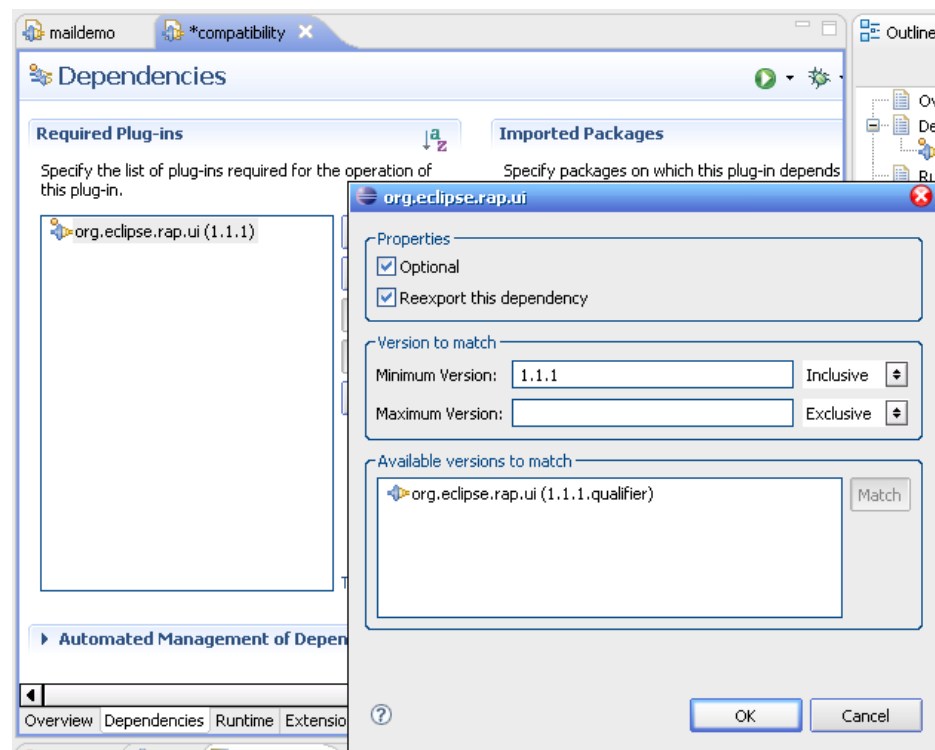
- package import
  - OSGi specification section 3.13.2
  - problems caused by split-packages
- optional dependencies on both library versions
  - warnings caused by missing bundle references
  - UI abstraction layer

# Dependencies



## The compatibility plug-in

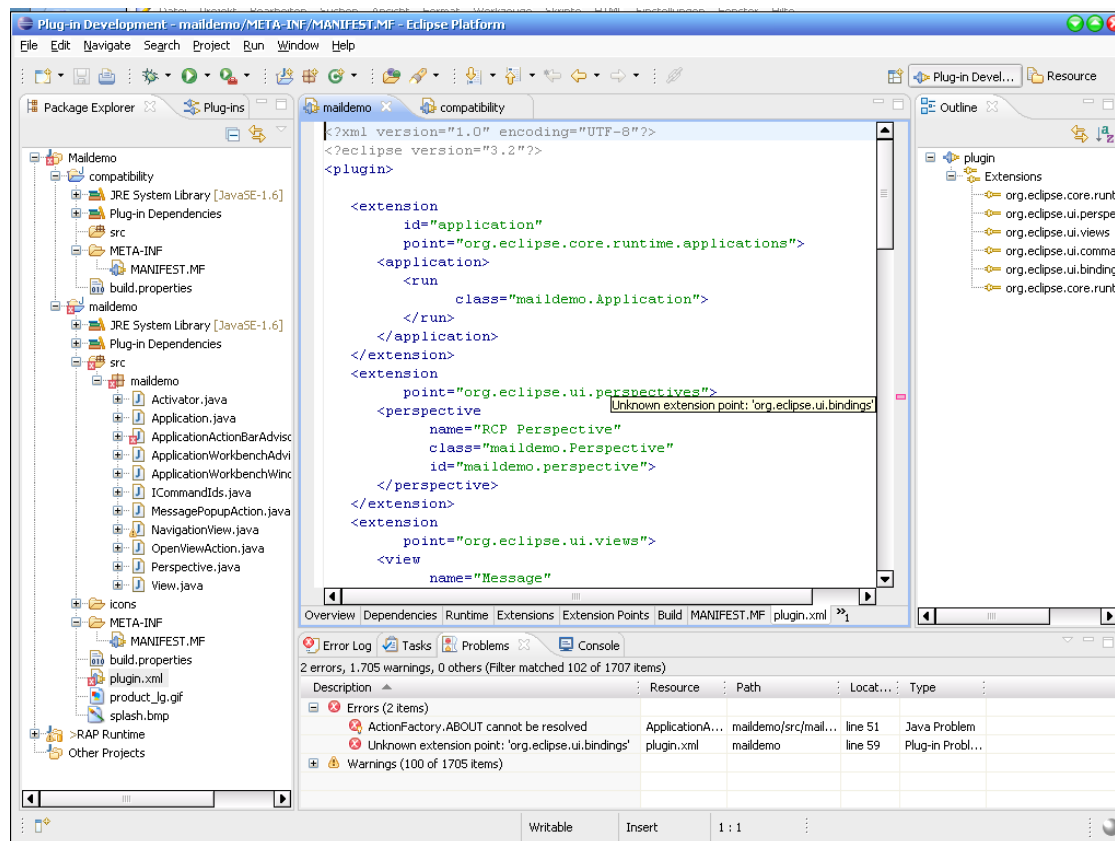
- location in projects folder
- import into both workspaces
- list of required bundles
  - org.eclipse.rap.ui
  - org.eclipse.ui
- properties
  - 'Optional'
  - 'Reexport this dependency'



# Dependencies



## RAP workspace after switching the UI dependencies





# Agenda

Basics

Setup

Dependencies

**Extensions**

API Differences

Lift Off

API Differences II

Multi-User Environment

# Extensions



## **The problem of different Extension-Points**

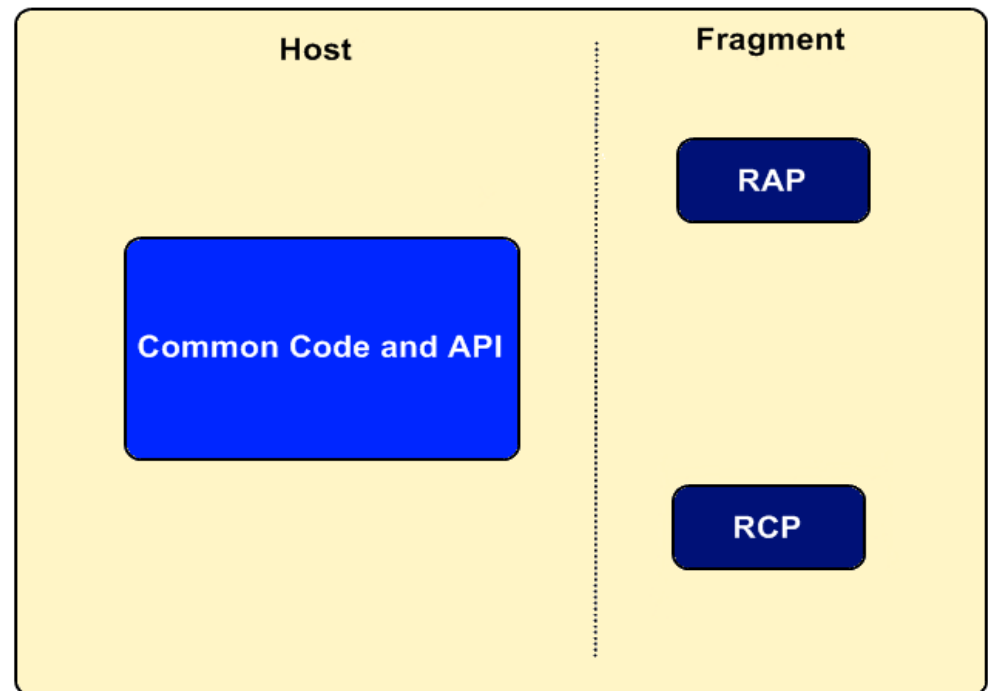
- not all RCP E-Ps are available in RAP
  - e.g. bindings, helpSupport ...
- additional RAP E-Ps for web specific requirements
  - e.g. entrypoint, phaselistener ...

# Extensions



## Fragment Solution

- two fragments per plug-in
  - one for RAP specifics
  - one for RCP specifics
- at runtime, only the plug-in that fits the environment will be installed
- platform specific E-P contributions are moved into the corresponding fragment
- bundle structure stays intact

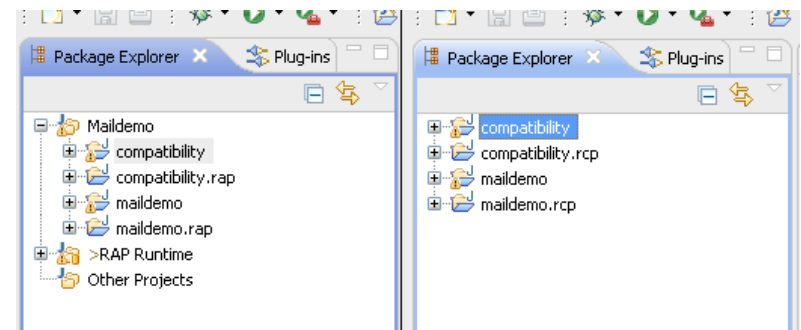
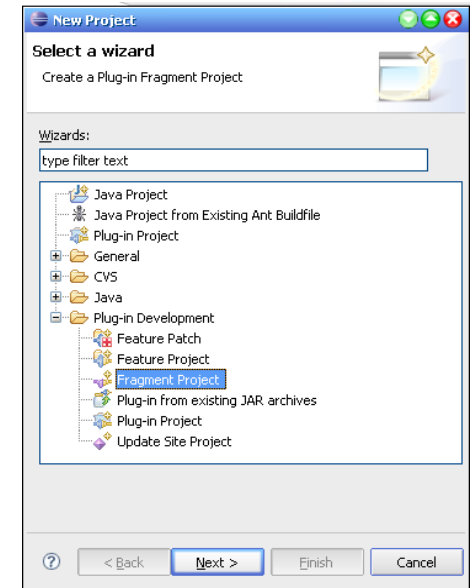


# Extensions



## Creation of the Fragment Projects

- using the new project wizard
  - fragment project
  - all fragments are filed in the projects folder
- the following fragments are created
  - compatibility.rap
  - compatibility.rcp
  - maildemo.rap
  - maildemo.rcp
- each workspace contains only the relevant fragments

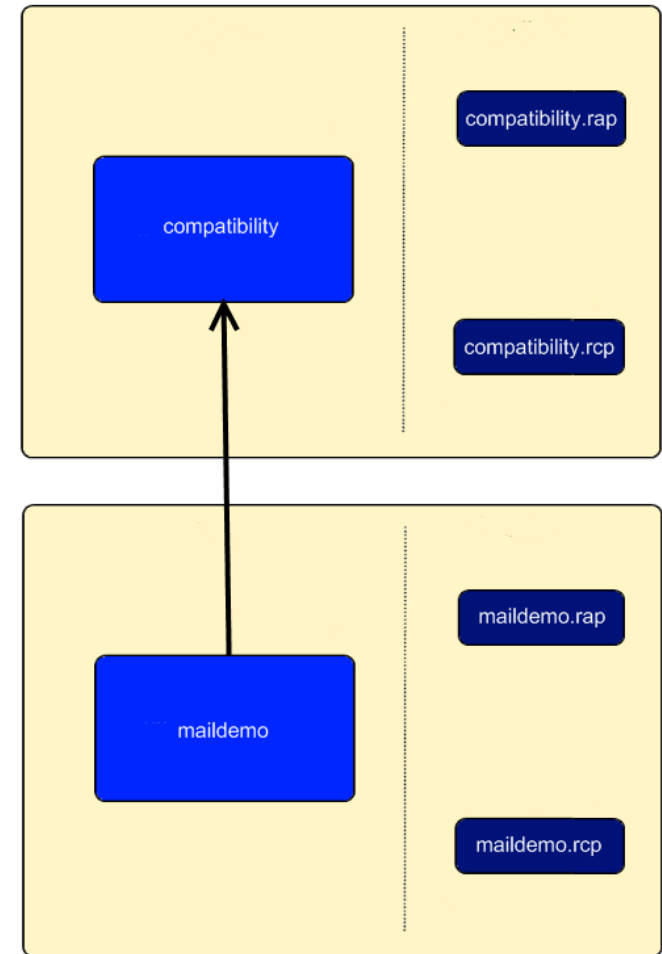


# Extensions



## Creation of the Fragment Projects

- using the new project wizard
  - fragment project
  - all fragments are filed in the projects folder
- the following fragments are created
  - compatibility.rap
  - compatibility.rcp
  - maildemo.rap
  - maildemo.rcp
- each workspace contains only the relevant fragments

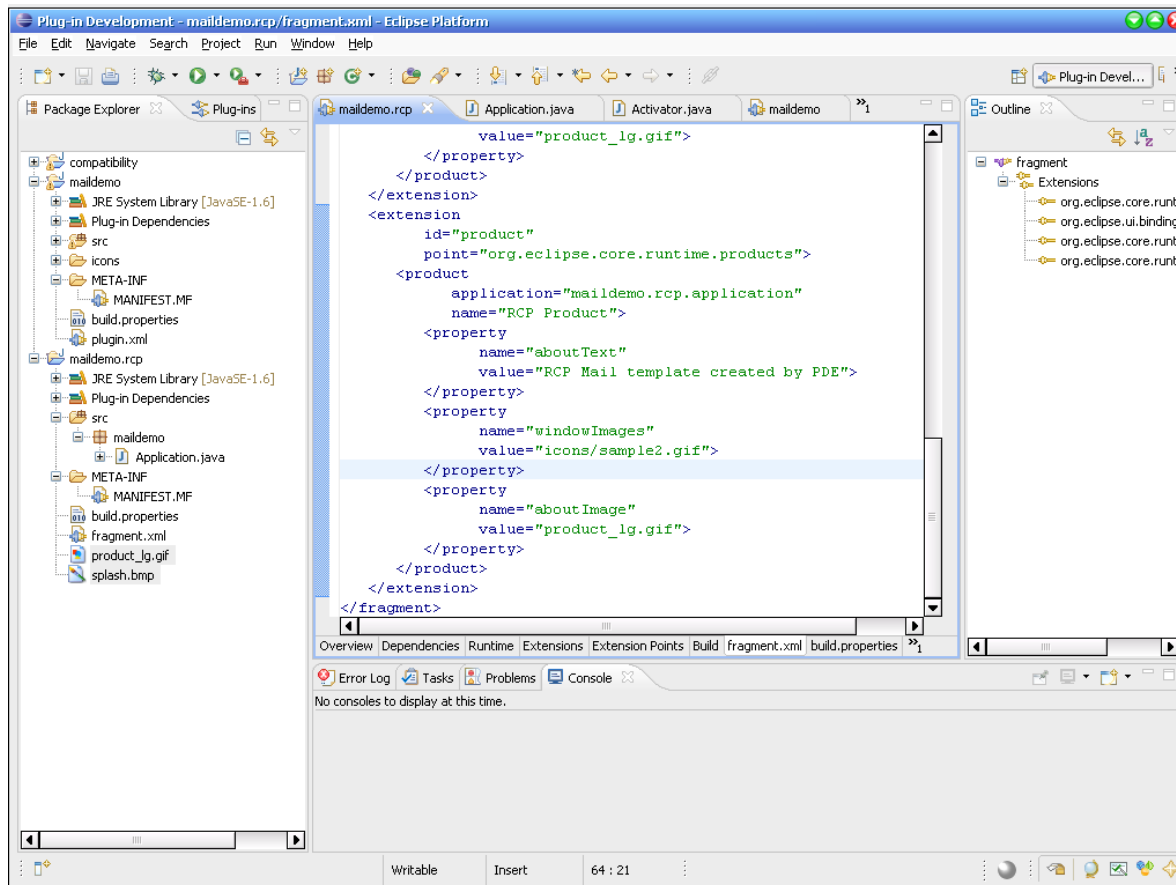




# Extensions



## RCP workspace after moving E-Ps into fragment.xml





# Agenda

Basics

Setup

Dependencies

Extensions

**API Differences**

Lift Off

API Differences II

Multi-User Environment

# API Differences



## The problem of different APIs

- not all RCP APIs are available in RAP
  - e.g. GC, MouseMove Events, FileDialog ...
- additional RAP APIs for web specific requirements
  - e.g. PhaseListener, ISessionStore ...

# API Differences



## **The problem of different APIs**

solution:

- abstract type in host-bundle, that encapsulates the problem
- type implementation in the fragment, that platform dependent, solves the problem
- loading the platform specific implementation at runtime by means of reflection

# API Differences



## Example 'About' Action

the last compile error in the RAP workspace is caused by the missing about action of the ActionFactory.

encapsulate the problem by using an ActionFactoryFacade type:

```
exitAction = ActionFactory.QUIT.create(window);  
register(exitAction);
```

```
aboutAction = ActionFactoryFacade.createAboutAction(window);  
register(aboutAction);
```

# API Differences



## Example 'About' Action II

implementation of ActionFactoryFacade:

```
public abstract class ActionFactoryFacade {  
    private final static ActionFactoryFacade IMPL;  
    static {  
        IMPL = (ActionFactoryFacade) ImplementationLoader.newInstance(ActionFactoryFacade.class);  
    }  
  
    public static IWorkbenchAction createAboutAction( final IWorkbenchWindow window ) {  
        return IMPL.createAboutActionInternal( window );  
    }  
  
    abstract IWorkbenchAction createAboutActionInternal(IWorkbenchWindow window);  
}
```

# API Differences



## Example 'About' Action III

implementation of ImplementationLoader:

```
public class ImplementationLoader {  
  
    public static Object newInstance(final Class type) {  
        String name = type.getName();  
        Object result = null;  
        try {  
            result = type.getClassLoader().loadClass(name + "Impl").newInstance();  
        } catch (Throwable throwable) {  
            String txt = "Could not load implementation for {0}";  
            String msg = MessageFormat.format(txt, new Object[] { name });  
            throw new RuntimeException(msg, throwable);  
        }  
        return result;  
    }  
}
```

# API Differences



## Example 'About' Action IV

platform specific implementations:

### RAP

```
public class ActionFactoryFacadeImpl extends ActionFactoryFacade {  
  
    private class AboutAction extends Action implements IWorkbenchAction {  
        private IWorkbenchWindow window;  
        public AboutAction(IWorkbenchWindow window) {  
            this.window = window;  
            setId( "about" );  
            setText("About RAP MailDemo");  
            setToolTipText("About RAP MailDemo");  
        }  
        public void dispose() {  
            window = null;  
        }  
        public void run() {  
            String title = "About Message";  
            String msg = "This is the about Message of the RAP Mail Demo";  
            MessageDialog.openInformation(window.getShell(), title, msg );  
        }  
    }  
  
    IWorkbenchAction createAboutActionInternal(IWorkbenchWindow window) {  
        return new AboutAction(window);  
    }  
}
```

### RCP

```
public class ActionFactoryFacadeImpl extends ActionFactoryFacade {  
  
    IWorkbenchAction createAboutActionInternal(IWorkbenchWindow window) {  
        return ActionFactory.ABOUT.create(window);  
    }  
}
```





# Agenda

Basics

Setup

Dependencies

Extensions

API Differences

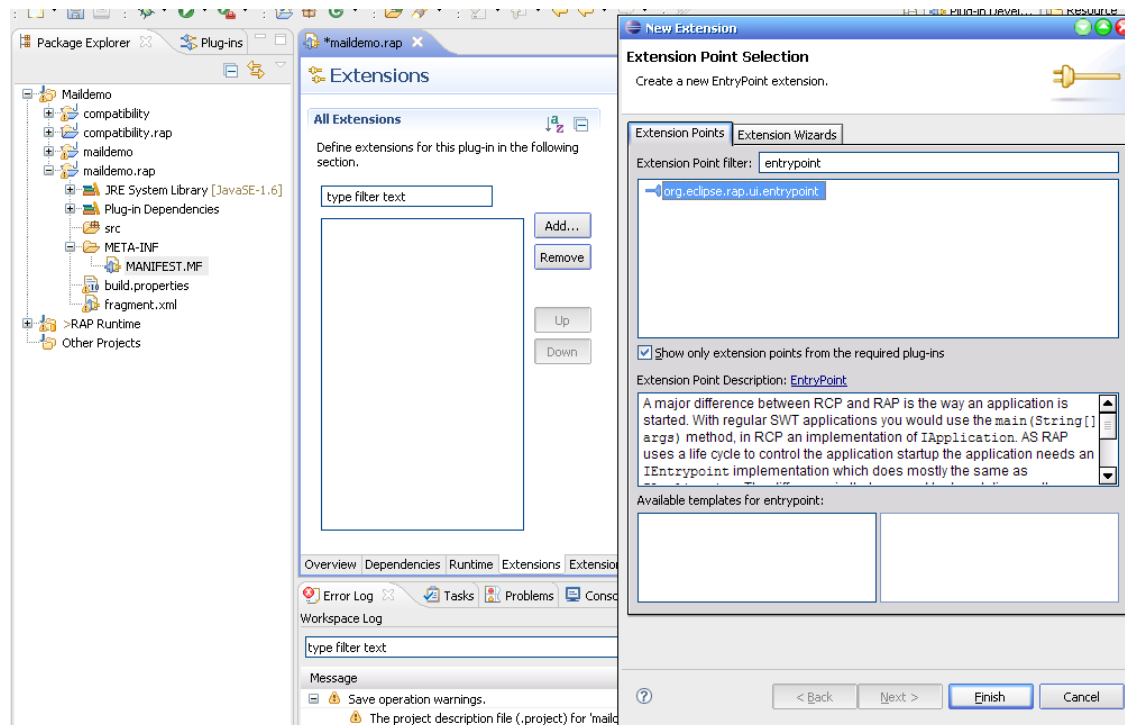
**Lift Off**

API Differences II

Multi-User Environment

# Lift Off

## Start-up help



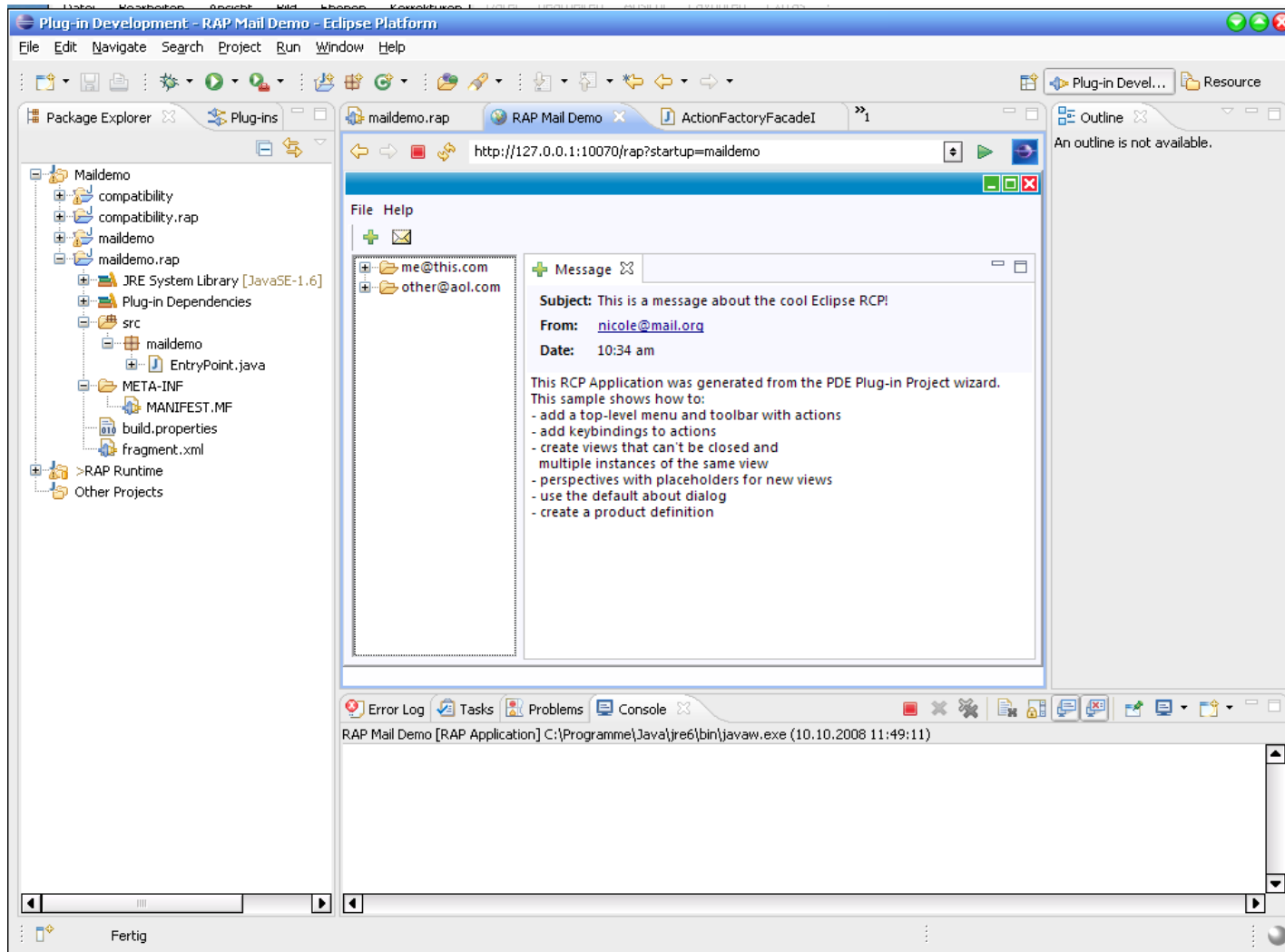
### Extension Element Details

Set the properties of "entrypoint". Required fields are denoted by "\*\*".

id*:	<input type="text" value="maildemo.rap.entrypoint"/>	
class*:	<input type="text" value="maildemo.EntryPoint"/>	<input type="button" value="Browse..."/>
parameter*:	<input type="text" value="maildemo"/>	

```
public class EntryPoint implements IEntryPoint {  
  
    public int createUI() {  
        Display display = PlatformUI.createDisplay();  
        return PlatformUI.createAndRunWorkbench(display, new ApplicationWorkbenchAdvisor());  
    }  
}
```

# Lift Off





# Agenda

Basics

Setup

Dependencies

Extensions

API Differences

Lift Off

**API Differences II**

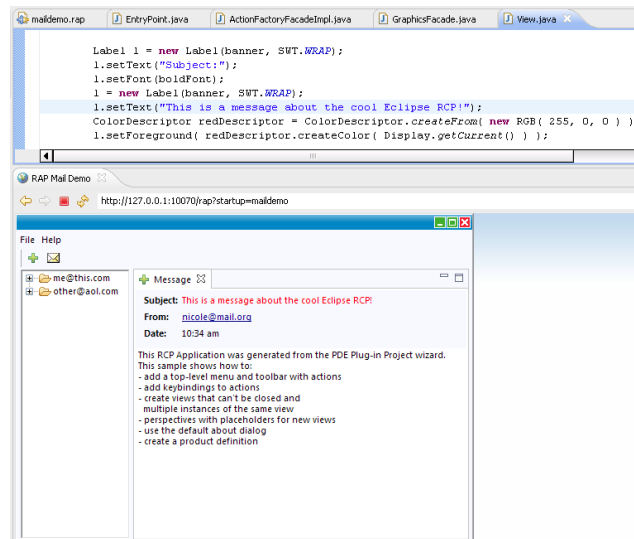
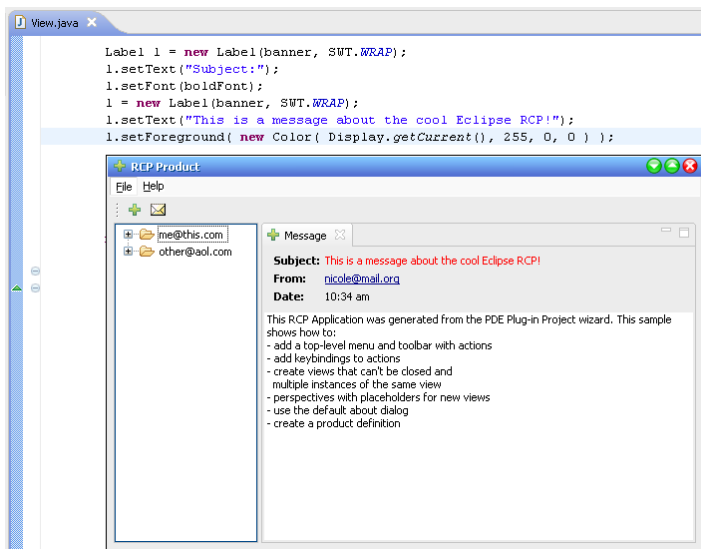
Multi-User Environment

# API Differences



## Resources in RAP are different from RCP

- no public constructor
- no dispose
- shared between sessions
- differences are resolved at JFace layer





# Agenda

Basics

Setup

Dependencies

Extensions

API Differences

Lift Off

API Differences II

**Multi-User Environment**

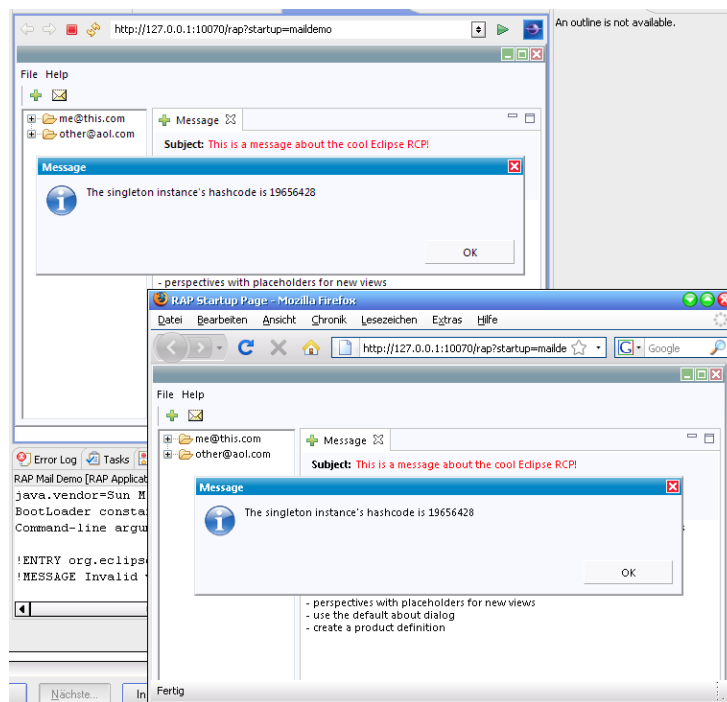
# Multi-User Environment



## Singletons

- classical singletons exist in application scope
- in RAP, most of the time, session scope is desirable

```
public class MySingleton {  
  
    private static MySingleton _instance;  
  
    private MySingleton() {  
        // prevent instance creation  
    }  
  
    public static synchronized MySingleton getInstance() {  
        if( _instance == null ) {  
            _instance = new MySingleton();  
        }  
        return _instance;  
    }  
}  
  
singletonAccess = new Action("Access Singleton") {  
    public String getId() {  
        return "singletonAccess";  
    }  
  
    public void run() {  
        String hashCode = String.valueOf(MySingleton.getInstance().hashCode());  
        String msg = "The singleton instance's hashCode is " + hashCode;  
        MessageDialog.openInformation(window.getShell(), "Message", msg);  
    }  
};  
register(singletonAccess);
```



# Multi-User Environment



## Singletons II

implementation in the host bundle:

```
public class MySingleton {  
  
    private final static ISingletonProvider PROVIDER;  
    static {  
        PROVIDER = (ISingletonProvider) ImplementationLoader.newInstance(MySingleton.class);  
    }  
  
    public static MySingleton getInstance() {  
        return (MySingleton) PROVIDER.getInstanceInternal();  
    }  
  
    MySingleton() {  
        // prevent instance creation  
    }  
}
```

```
public interface ISingletonProvider {  
    Object getInstanceInternal();  
}
```

fragment implementation:

### RAP

```
public class MySingletonImpl implements ISingletonProvider {  
  
    public Object getInstanceInternal() {  
        return SessionSingletonBase.getInstance(MySingleton.class);  
    }  
}
```

### RCP

```
public class MySingletonImpl implements ISingletonProvider {  
  
    private static MySingleton instance;  
  
    public synchronized Object getInstanceInternal() {  
        if( instance == null ) {  
            instance = new MySingleton();  
        }  
        return instance;  
    }  
}
```



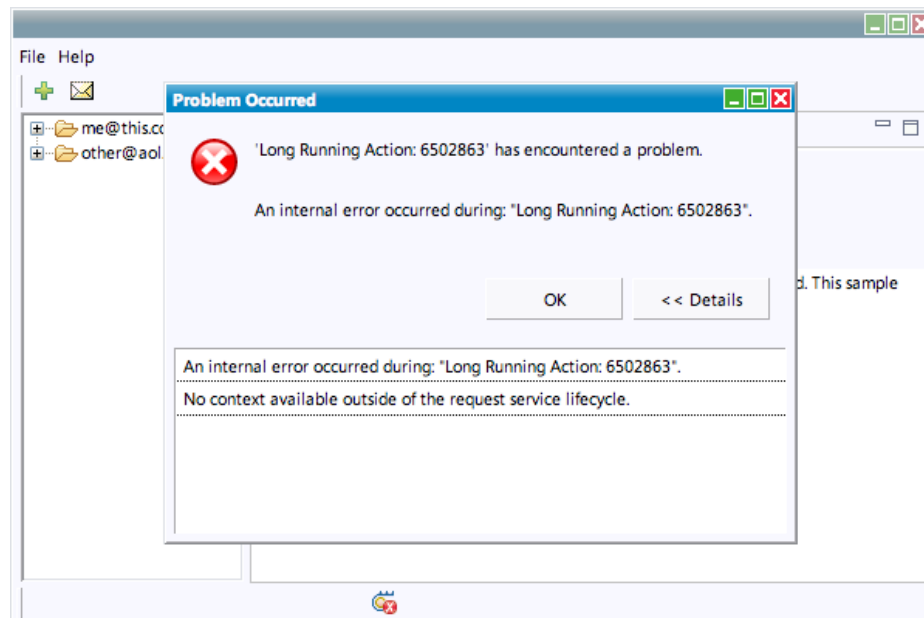
# Multi-User Environment



## Jobs

jobs can't implicitly access session-singletons

```
Job job = new Job("Long Running Action:") {
    protected IStatus run(final IProgressMonitor monitor) {
        IStatus result = Status.OK_STATUS;
        monitor.beginTask("Number counting", TASK_AMOUNT);
        for (int i = 0; i < TASK_AMOUNT; i++) {
            if (monitor.isCanceled()) {
                monitor.done();
                result = Status.CANCEL_STATUS;
            }
            int done = i % TASK_AMOUNT;
            String singleton = String.valueOf(MySingleton.getInstance().hashCode());
            monitor.subTask("work done [" + singleton + "]: (" + done + "%)");
            monitor.worked(1);
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        monitor.done();
        return result;
    }
};
job.setName(job.getName() + " " + job.hashCode());
job.setUser(true);
job.schedule();
```



# Multi-User Environment

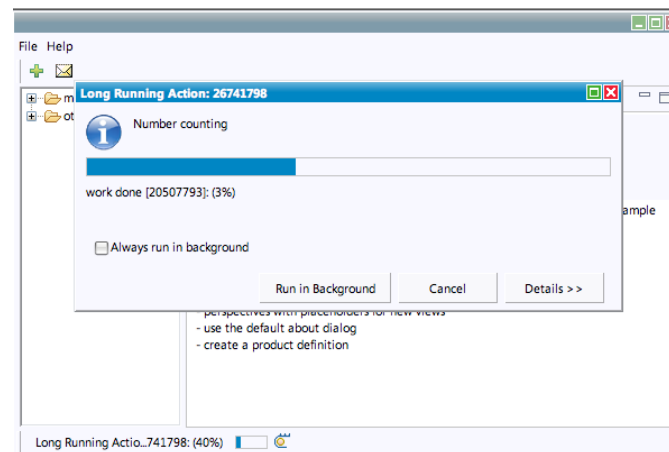


## Jobs II

implementations in the host bundle:

```
JobRunnable runnable = new JobRunnable() {
    public IStatus run(final IProgressMonitor monitor) {
        IStatus result = Status.OK_STATUS;
        monitor.beginTask("Number counting", TASK_AMOUNT);
        for (int i = 0; i < TASK_AMOUNT; i++) {
            if (monitor.isCanceled()) {
                monitor.done();
                result = Status.CANCEL_STATUS;
            }
            int done = i % TASK_AMOUNT;
            String singleton = String.valueOf(MySingleton.getInstance().hashCode());
            monitor.subTask("work done [" + singleton + "]: (" + done + "%)");
            monitor.worked(1);
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        monitor.done();
        return result;
    }
};

Job job = JobFactory.createJob(Display.getCurrent(), "Long Running Action:", runnable);
job.setName(job.getName() + " " + job.hashCode());
job.setUser(true);
job.schedule();
```



```
public abstract class JobFactory {
    private final static JobFactory IMPL;
    static {
        IMPL = (JobFactory) ImplementationLoader.newInstance(JobFactory.class);
    }

    public static Job createJob( final Display display, final String name, final JobRunnable runnable) {
        return IMPL.createJobInternal(display, name, runnable);
    }

    abstract Job createJobInternal( Display display, String name, JobRunnable runnable);
}

public interface JobRunnable {
    IStatus run(IProgressMonitor monitor);
}
```

# Get RAP - <http://eclipse.org/rap>



## Demos

See some demos here

## Downloads

Get the latest RAP release

The RAP project enables developers to build rich, Ajax-enabled Web applications by using the Eclipse development model, plug-ins with the well known Eclipse workbench extension points, JFace, and a widget toolkit with SWT API (using **qooxdoo** for the client-side presentation). The project has graduated from incubation and released its 1.0 release.

[Learn more ...](#)

# More Information



- <http://www.eclipse.org/rap> - RAP project page
- <http://wiki.eclipse.org/RAP> - RAP project wiki
- <http://www.qooxdoo.org> - qooxdoo js library

Questions?

[info@innoopract.com](mailto:info@innoopract.com)