



Eclipse Plug-ins Development

Radu Farcas – Eclipse Plug-Ins Developer
Catalin Udma – Linux Developer



02-July-2013

Freescale, the Freescale logo, AlliVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, mobileGT, PowerQUICC, Processor Expert, QorIQ, Qorivva, StarCore, Symphony and VortiQa are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Airfast, BeeKit, BeeStack, CoreNet, Flexis, MagniV, MXC, Platform in a Package, QorIQ Qonverge, QUICC Engine, Ready Play, SafeAssure, the SafeAssure logo, SMARTMOS, TurboLink, Vybrid and Xtrinsic are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © 2012 Freescale Semiconductor, Inc.



Plug-ins Development



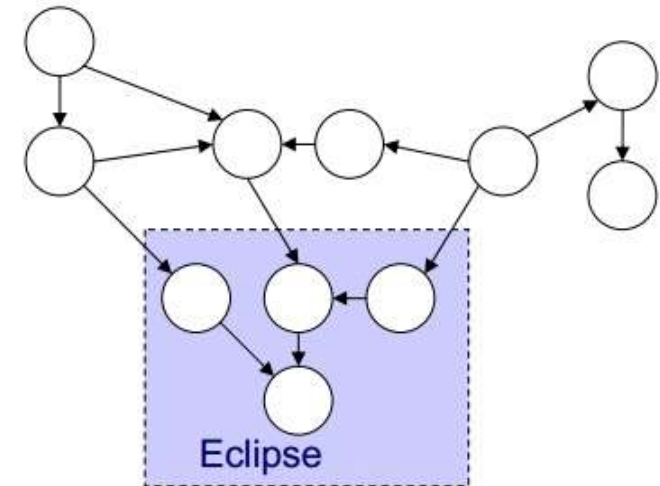
- What's a plug-in?
 - A software component that adds a specific feature to an existing software application
- Standard Java lacks an explicit notion of components
- OSGi Framework
 - Bundle**: a group of Java classes and additional resources equipped with a detailed manifest MANIFEST.MF
 - Life-cycle**: INSTALLED, RESOLVED, STARTING, ACTIVE, STOPPING, or UNINSTALLED
- Equinox
 - Eclipse project that provides a certified implementation of the OSGi
- Eclipse plug-in extension system : possibility to add and customize features
 - All kinds of UI elements
 - Application specific logic



OSGi and Equinox



- **Components == OSGi Bundles == Eclipse Plug-in**
- OSGi support dynamic update and install
- Downstream components can access upstream components through the extension mechanism
 - Downstream component registers (declaratively) an extension point
 - Dependent components register (declaratively) extensions



Eclipse Plug-ins Interactions



Plug-in Development Concepts



- Plug-in
 - Used to group your code into a modular, extendable and sharable unit.
- Feature
 - Used to package a group of plug-ins together into a single installable and updatable unit
- Extensions and Extension Points
 - When a plug-in wants to allow other plug-ins to extend or customize portions of its functionality, it will declare an extension point
 - Plug-ins that want to connect to that extension point must implement that contract in their extension
- Fragment
 - Used to replace or extend the functionality of an existing plug-in
- Target Platform
 - Refers to the plug-ins which your workspace will be built and run against



MANIFEST.MF and plugin.xml



- MANIFEST.MF
 - ID
 - Name
 - Activator class – start() and Stop() methods
 - Execution Environment
 - Runtime dependencies – required plug-ins
 - Exported packages
 - Java classpath
- Plugin.xml
 - Extensions
 - Extension points
 - Build
- Plug-in Registry View



Extension Points



- `<extension`
- `point="org.eclipse.ui.editors">`
- `<editor`
- `name="Sample Multi-page Editor"`
- `extensions="mpe"`
- `icon="icons/sample.gif"`
- `contributorClass="firstplugin.editors.MultiPageEditorContributor"`
- `class="firstplugin.editors.MultiPageEditor"`
- `id="firstplugin.editors.MultiPageEditor">`
- `</editor>`
- `</extension>`



Exercise – Create a multi-page editor



Exercise – Create a pop-up menu action



How Do I Work with Extension Points?



- Find extension points
 - - Documentation
 - - Plugin.xml editor – ‘Add...’
 - - Plug-ins registry editor
 - - Load SDK plug-in as projects with source folders
- Read extension point description
- Look for examples
 - - Existing places where the extension points have been previously used
- Deploying your solution
 - - export wizard via *File* → *Export* → *Plug-in Development* → *Deployable plug-ins and fragments*.