Tutorial:
Creating, Packaging, Testing and Deploying Features in Eclipse 3.0

Sandy Minocha – IBM Canada (Vancouver, CA)
Pat McCarthy – IBM USA (Research Triangle Park, NC)
IBM Rational Software

Agenda

- **Getting started**
  - Overall introduction to Features in Eclipse
  - Review of Install basics (extensions, products, Update Manager)
  - Hands on: Define, Build, Deploy, and Service a Feature Set

- **Getting automated**
  - Understanding the build options provided
  - Reviewing the support for full Ant builds provided by PDE Build
  - Hands on: Automating Feature Build and Packaging using Ant, org.eclipse.pde.build, and antRunner

- **Getting branded**
  - Identifying the branding and product launch options in Eclipse
  - Hands on: Defining and Implementing a Branded Eclipse-based Product
Sample Application

Sample Application: Overview

- Sample application displays a list of cities and countries in a view and dialog window.

- Consists of four plugins:
  1. com.ibm.jdg2e.simplemodel - responsible for providing data to the Locations (Basic View) view, JDG2E: JFace Viewer Examples dialog and the Properties view
  2. com.ibm.jdg2e.view.basic - contributes the Locations (Basic View) view
  3. com.ibm.jdg2e.jface.viewers - contributes the JDG2E: JFace Viewer Examples dialog
  4. com.ibm.jdg2e.simplemodel.ui - contributes feature and product branding
Sample Application: Overview

Tutorial Overview

- **Scenario 1: Installing Features**
  1. Create a Feature (Exercise A: Part 1)
  2. Package a Feature to install in an Install site.
     1. using PDE (Exercise A: Part 2)
     2. using AntRunner (Exercise B)
  3. Package a Feature in an Install site.
  4. Deploy a Feature (Exercise A: Part 3)

- **Scenario 2: Servicing Features (Exercise A: Part 4)**
  1. Add Feature branding content
  2. Package a Feature to install in an Update site.
  3. Package a Feature in a Update site.
  4. Deploy a Feature

- **Scenario 3: Product Branding (Exercise C)**
Getting Started:

Sample Application Overview
Using PDE to Build & Package Features
Using Update Manager to Deploy Features

Hands on: Define, Build, Deploy, and Service a Feature Set

Tutorial: Creating, Packaging, Testing and Deploying Features in Eclipse 3.0
Sandy Minocha and Pat McCarthy – IBM Rational Software

Tutorial Overview

- **Scenario 1: Installing Features**
  - com.ibm.jdg2e.simplemodel.ui feature
    1. com.ibm.jdg2e.view.basic plugin
    2. com.ibm.jdg2e.jface.viewers plugin
  - com.ibm.jdg2e.simplemodel feature
    1. com.ibm.jdg2e.simplemodel plugin

- **Scenario 2: Servicing Features**
  - com.ibm.jdg2e.simplemodel.ui feature
    1. com.ibm.jdg2e.view.basic plugin
    2. com.ibm.jdg2e.jface.viewers plugin
    3. com.ibm.jdg2e.simplemodel.ui plugin
  - com.ibm.jdg2e.simplemodel feature
    1. com.ibm.jdg2e.simplemodel plugin
Step 1: Create a Feature

Role of a Feature Developer

Features: Overview

- An installable unit of function
  - Depending on their definitions, installed features can often be directly disabled/enabled
- A packaging construct
  - Plugins add function to Eclipse. Features organize plugins so that they can be installed and managed by the Update Manager
- A logical container for plugins
  - A feature identifies the plugins, at a specific version level, that are part of the feature. This allows the Update Manager to manage plugins by managing the features that contain the plugins.
- Capable of nesting other features
  - This technique is used to manage the source of service for a set of features.
- Installed and managed using the Update Manager
  - Plugins can be copied to the install directory, but they are unmanaged. Features regardless of how they are added to your configuration, are managed by Update Manager.
Features: Tooling – New Feature wizard & editor

- Overview page
  - define Update URL

- Information page
  - define Feature description
  - define Copyright Notice
  - define License Agreement

- Content page
  - define included Plugins

- Advanced page
  - define Features to be nested

Features: Tooling – Import Features wizard
Step 2: Package a Feature

Role of a Feature Developer

Packaging Features: Overview

- Packaging of plugins using features is required to install into Eclipse
- Process driven by build.properties, feature.xml and plugin.xml files
- Features and their associated plugins can be packaged in two ways:
  1. Packaging for an Extension site
     - eclipse folder
     - .eclipseextension file
     - features folder
     - <featureId_ver> folder
     - Feature.xml
     - ... 
     - plugins folder
     - <pluginsId_ver> folder
     - plugin.xml
     - <runtime>.jar
     - ... 
     - ...
Packaging Features: Overview

- Features and their associated plugins can be packaged in two ways:
  1. Packaging for an Extension site
  2. Packaging for an update site

  - updateSite folder
    - site.xml file
    - Features folder
      - <featureId_ver>.jar
      - ...
    - Plugins folder
      - <pluginId_ver>.jar
      - ....

Packaging options include:

1. Ant scripts using PDE
   1. Build for an Extension site
   2. Build for an Update site

2. Export Features wizard
   1. Export for an Extension site
   2. Export for an Update site

3. Site editor’s Build All action
   1. Build for an Update site

4. Ant scripts using AntRunner
   1. Build for an Extension site
   2. Build for an Update site
Packaging Features: Tooling – Ant Build dialog

- Multiple targets exist in the generated build.xml file.
- Extension site targets:
  - clean – removes all generated and temporary files
  - refresh – refreshes folder
  - build.jars – builds all the plugins’ runtime jars
  - zip.distribution – creates a ZIP containing all the features and plugins
- Update site targets:
  - clean
  - refresh
  - build.jars
  - build.update.jar – creates individual JAR archives for each feature and plugin

Packaging Features: Tooling – Export Features wizard

- Export Features wizard provides 3 options:
  1. Export as a directory structure
     - builds for Extension site
  2. Export as a single ZIP file
     - builds for Extension site
  3. Export as individual JAR archives
     - builds for Update site
Packaging Features: Tooling – Site editor’s Build All action

Step 3: Package a Feature in an Install site

Role of a Feature Developer
Install Sites: Overview

- Install sites are the basic building block in a configuration.
- An install site is a location on the file system where a features and plugins directory can be found.
- An install site is a single location on the file system but the same location could be included in multiple configurations in multiple Eclipse-based product installations.
- Types of install sites:
  - **Platform base site** – where Eclipse itself is installed. Always exists.
  - **Extension site** - Distinguished from other sites through .eclipseextension file.
  - **Update site** - Distinguished from other sites through site.xml file.
  - **Link site** – a *.link file identifies the location of a valid install site. Link files are stored in the links directory under the <eclipse_install> directory. **Note:** This technique is a carryover from older versions of Eclipse and may not be supported forever.

Install Sites: Overview

- Two options exist for how Features can be installed in Eclipse. Result of these two techniques is same.

1. Install Feature from an Extension site
   - Tool providers package their Features including an .eclipseextension file using InstallShield.
   - .eclipseextension content includes:
     ```
     id=com.ibm.jdg2e.simplemodel.ui
     name=JDG2E Simple Model UI
     version=1.0.0
     ```

2. Install Feature from an Update site
   - Tool providers package their Features including a site.xml file to a HTTP site.
   - site.xml content includes:
     ```
     <site>
     <feature url="features/com.ibm.jdg2e.simplemodel.ui_1.0.0.jar"
     id=com.ibm.jdg2e.simplemodel.ui version="1.0.0">  
     <category name="jdg2e.service"/>  
     </feature>
     <category-def name="jdg2e.service" label="JDG2E Service">
     <description>JDG2E Service Description</description>
     </category-def>
     </site>
     ```
Step 4: Deploy a Feature

Role of an End User

Update Manager: Overview

- Update Manager is an Eclipse component. Divided into two pieces:
  - User interface
  - Eclipse-based headless application - StandaloneUpdate
- Update Manager used to accomplish the following tasks:
  1. Install a Feature. Two options exist:
     1. Access Extension sites locally
     2. Access Update sites to download new Features
  2. Remove a Feature. Two options exist:
     1. Disable/Enable a Feature. Applies to Features contained in both Extension and Update sites.
     2. Uninstall a Feature. Applies to Features contained in Update sites.
  3. Service a Feature
     1. Access Update sites to download and maintain existing Features
  4. Add/Remove Install sites
Update Manager: Configurations

- The Update Manager controls your Eclipse Configuration.
  - Creates initial configuration during startup if missing (i.e., processes content of the features and plugins directories of the install sites)
  - Reads the active configuration
  - Manages changes that occur to the configuration
- A Configuration identifies:
  - what Install sites are accessible
  - the Features that exist in each site
- Configuration information is saved in the platform.xml file found in the configuration/org.eclipse.update directory.
- A Configuration applies to any workspace that might be accessed
- Default configuration is used when Eclipse is launched
  - `<eclipse_install>/eclipse/configuration`
- Alternate configuration can be specified using the `-configuration` startup parameter

Update Manager: Branding

- Four About dialogs provide a snapshot of the entire active configuration.
  1. About Eclipse Platform dialog – exposes product and feature branding content
     - `<plugin>/about.ini`
       - featureImage property
       - aboutText property
     - `<plugin>/about.properties` (optional)
     - `<feature>/license.html`
  2. About Eclipse Platform Features dialog – shows a list of features
     - `<feature>/license.html`
  3. About Eclipse Platform Plugins dialog – opens a list of active plugins
     - `<plugin>/about.html`
  4. Feature Plugins dialog – shows a list of plugins for a specific feature
     - `<plugin>/about.html`
Update Manager: Tools – Product Configuration dialog

- Dialog features:
  1. Install a Feature from an Extension site
  2. Disable/Enable a Feature
  3. Uninstall a Feature that was installed from an Update site.
  4. Service a Feature

Update Manager: Tools – Install/Update dialog

- Dialog features:
  1. Install a Feature from an Update site
  2. Service a Feature
Update Manager: Tools – StandaloneUpdate Application

- StandaloneUpdate application features:
  1. Add Install sites
  2. Remove Install sites
  3. Disable Features
  4. Enable Features

Update Manager: Tools – About dialogs
Getting Automated: Implementing an Automated PDE Build

Hands on: Automating Feature Build and Packaging using Ant, org.eclipse.pde.build, and antRunner

Tutorial: Creating, Packaging, Testing and Deploying Features in Eclipse 3.0
Sandy Minocha and Pat McCarthy – IBM Rational Software

Build Options
Different techniques available to build
Build Processing with the PDE User Interface

- Eclipse PDE user interface can drive build requests
  - Plug-in and Fragment Export wizard
  - Feature Export wizard
  - Editor for site.xml has a "Build All" button

- Cascade processing
  - Build selected plug-ins and/or fragments
  - Build selected features and their plug-ins along with included features
  - Build one site and its features

- Issues
  - Build outputs only in destination zip file or site project structure
  - User interaction required
  - User must understand structure and dependencies

Build Processing with the PDE Generated Ant Scripts

- Eclipse PDE Ant scripts can be used to drive build requests
  - A build.xml script can be generated for each plug-in
  - A build.xml script can be generated for each feature and its referenced plug-ins and/or fragments
  - No cascade to included features for build.xml generate

- Cascade processing
  - Build one plug-in
  - Build one feature and its plug-ins/fragments as well as included features

- Issues
  - Build outputs are in project directory
  - User interaction required
  - User must understand structure and dependencies
Build Processing with Customized Ant Scripts

- Custom Ant scripts can be used to drive build requests
  - Allows for total control of the build process
  - Scripts for each project or system can be defined
- PDE provides support for this process
  - Specialized PDE Ant targets can be invoked by custom Ant scripts
    - CVS Fetch
    - Generation of build.xml for features, plug-ins, fragments
  - Templates for custom scripts are provided
  - Eclipse antRunner application can be used to process scripts and automate process
- Feature cascade processing supported
  - Build one feature and any referenced feature can also be built
  - Allows system structure to be defined and feed the build automation process

Implementing an Automated PDE Build

Using the built in support for preparing and packaging plug-ins and features
(the eclipse.org way)
PDE Build Process

- Build the eclipse.org way
  - Prerequisite is a root feature

- Templates provided by PDE build
  - build.xml – base Ant script that governs the full process
  - customTargets.xml – supports customization and extension of processing
  - genericTargets.xml – provides the base implementation

- Templates cover complete build process
  - Checkout from CVS
  - Generate build scripts for features/plug-ins/fragments
  - Process build
  - Assemble build outputs

- Customize as required
  - build.properties values govern processing
  - Limited mandatory customization
  - Unlimited optional processing

Custom Targets Provided by PDE Build

- `eclipse.fetch` – generates a customized Ant script to check out feature/plug-in/fragment projects from CVS
  - Allows for full automation including a pull of the most recent code, or specific project versions (tags) from CVS to feed the build process
  - Mapping between feature/plug-in/fragment projects and their location in CVS is provided as an input along with a specific entry to fetch
  - Fetching a feature will fetch referenced plug-ins/fragments and included features

- `eclipse.buildScript` – generates the feature/plug-in/fragment specific build scripts (build.xml files) required to perform build processing
  - Allows for the automatic update of these custom builds scripts so that they reflect any changes in structure or content (build.properties)
  - Request to generate build scripts for a feature will process referenced plug-ins/fragments and included features
Running PDE Build Processing

- PDE Build processing uses a set of Ant scripts
  - Based on templates
  - Additional scripts are generated at runtime

- Multiple options exist for Ant invocation
  - Eclipse Ant invocation (Run as > Ant Build); (only supports a subset of the build process – no CVS Fetch support)
  - Launching an Eclipse application

- Eclipse antRunner application is part of the Eclipse base
  - Application: org.eclipse.ant.core.antRunner
  - Provided by org.eclipse.ant.core plug-in
  - The antRunner application invokes Ant scripts while also supporting access to the Eclipse-specific Ant targets

Automated Ant Build Process

- **Build.properties**
- **Processing Options**
  - mavenVersionTag
  - buildDirectory
  - buildType
  - build
  - buildLabel
  - timesamp
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir
  - basedir

- **build.xml**
  - Runs Process
  - Build All Elements
  - antBuild
  - antclean
  - antCatalog
  - antJar
  - antPackage
  - antProcess
  - antDebug
  - antClean

- **customTargets.xml**
  - Build Content & Customization
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets
  - customTargets

- **genericTargets.xml**
  - Standard Processing
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets
  - genericTargets

- **Fetch_xxx.xml**
  - Generated and Run
  - Fetch_xxx
  - Fetch_xxx
  - Fetch_xxx
  - Fetch_xxx
  - Fetch_xxx
  - Fetch_xxx
  - Fetch_xxx

- **Assemble_xxx.xml**
  - Generated and Run
  - Assemble_xxx
  - Assemble_xxx
  - Assemble_xxx
  - Assemble_xxx
  - Assemble_xxx
  - Assemble_xxx
  - Assemble_xxx

Build processing cascades through the build scripts to perform CVS checkout through final assembly.
PDE Build Control Options – build.properties

Additional processing can be performed during a PDE build

- Files and directories can be added to zip directory structure using the root control element in the build.properties file
  - root - list the files and folders that must be included in the root of the product. The different values supported are:
    - <folderName> - a relative path to a folder to be copied;
    - file:<fileName> - a relative path to a file to be copied;
    - absolute:<folderName> - an absolute path to a folder to be copied;
    - absolute:file:<fileName> - an absolute path to a file to be copied;
  
  Example: root=root.files.product

- Source features and/or plug-ins can be generated and included in build
  - Creates the feature and plug-ins required and packages the source along with an appropriate org.eclipse.pde.core.source extension definition
  - Additional plug-ins can be included in the source feature (special documentation)

  Example: generate.feature@com.xxx.source=com.xx

Summary

- Automation of build processing is possible
  - Requires a commitment to a full checkout/build lifecycle
  - Eclipse PDE build scripts provide a solid start while allowing for customization

- Customization of build processing is possible
  - Allows for system-specific control
  - Creation of specific directory structures and content outside the standard features and plugins directory tree

- Leaning how to use the PDE build process is the first step
  - Structured exercise to implement an automated PDE build is provided
  - CVS checkout task is emulated; take home instructions for full implementation are included
Getting Branded:

Branding: Product vs. Feature
Eclipse Product Definitions
Branding in an RCP Application
Product Editor (new in 3.1)

Hands on: Defining and Implementing a Branded Eclipse-based Product

Tutorial: Creating, Packaging, Testing and Deploying Features in Eclipse 3.0
Sandy Minocha and Pat McCarthy – IBM Rational Software

Branding: Product vs. Feature

Branding Layers in Eclipse - What can be branded and where it shows in an Eclipse Workbench Application
Branding Support

- Branding information available for
  - Product launch point
  - Features installed by a product or extension

- Product launch point defines
  - Workbench title
  - About Product dialog content
  - Splash image displayed at startup (part of configuration detail)

- Features identified by:
  - Graphic representation on About [product] dialog
  - Additional content on About [product] Features dialog

Product Information

- Product definition determines branding content and controls runtime behavior
  - Window image
  - Window Name
  - About product info
  - Initial welcome (introduction) page
  - Default perspective
  - Preference default overrides

- Features can optionally provide
  - Welcome (introduction) page contributions
  - About feature info

Note: Not all features need branded; often just one will do fine
Product Information

Welcome to My Product by CompanyA, Inc. Version 1.9.0

Welcome (Introduction)

Splash screen

About product info

About feature info

Eclipse Product Definitions

Defining and Configuring a Product Extension
Product Extension Defines Primary Branding

- Contribute to org.eclipse.core.runtime.products extension point

```xml
<extension id="product" name="JDG2E Simple Product" point="org.eclipse.core.runtime.products">
  <product description="JDG2E Simple Product: Demonstrates JFace Viewers" application="com.ibm.jdg2e.simplemodel.ui.appl">
    <property name="windowImage" value="icons\jdg2eProd.gif"/>
    <property name="aboutImage" value="icons\jdg2eAbout.gif"/>
    <property name="appName" value="JDG2EApp"/>
    <property name="preferenceCustomization" value="plugin_customization.ini"/>
  </product>
</extension>
```

- application value controls the Eclipse application started for product
- preferenceCustomization identifies the file used to override preference value defaults in any plug-in

Product Branding Visible in UI Display

- Product extension supplies values used at runtime in appropriate areas of the Workbench user interface
- Product also identifies the application that should be started
- Feature branding is optional and is done separately

```xml
<extension id="product" name="JDG2E Simple Product" point="org.eclipse.core.runtime.products">
  <product description="JDG2E Simple Product: Demonstrates JFace Viewers" application="com.ibm.jdg2e.simplemodel.ui.appl">
    <property name="windowImage" value="icons\jdg2eprod.gif"/>
    <property name="aboutImage" value="icons\jdg2eabout.gif"/>
    <property name="appName" value="JDG2EApp"/>
    <property name="preferenceCustomization" value="plugin_customization.ini"/>
  </product>
</extension>
```
### About.ini

- Contains information about a feature.
- "Key" are externalized strings defined in about.properties.

- Property "featureImage" contains path to feature image (32x32).
  `featureImage=icons/jdg2eFeat.gif`

- Property "aboutText" contains text for "About" dialog.
  `aboutText=%blurb`

### Feature Branding Support

Clicking on image opens provider specific dialog of feature details.

### Running the Product

- The product to be started can be identified in the launch dialog during debug or run processing.

- The `eclipse.exe` accepts a `-product` parameter.

- The `config.ini` file can identify the product to start when the `eclipse.exe` is started.
Adding Splash to your Product

- Splash image is not a product attribute
  - Splash is typically a file named splash.bmp
  - Identified in the config.ini associated to the launch/configuration
  - Typically found in a plug-in’s directory

- Splash can be optional
  - Not defined – not shown
  - The –nosplash parameter can be used to prevent display

- Two options for identification
  # The comma-separated list of locations to search for the splash screen file (splash.bmp).
  # For each list element a subdirectory structure based on the pattern nl/-locale- is searched.
  # The system binds to the first matching file. There is no default value.
  osgi.splashPath = platform:/base/plugins/org.eclipse.platform

  # The location of the splash screen file. If this value is set at system startup it is used
  # in favor of the osgi.splashPath searching outlined above. If the value is not set
  # the searching is done and this key is bound to the result of the search.
  osgi.splashLocation =E:/Custom_RCP_Base/eclipse/plugins/com.ibm.jdg2e.simplemodel.ui_3.0.0/splash.bmp

Branding in an RCP Application

Adding and Using About [product] Dialog Support
Branding of Generic Workbench Applications

- IDE-based products must use standard product branding approach
  - The `org.eclipse.ui.ide.workbench` application is used “as-is”
  - Use of features and associated feature branding is expected

- RCP applications must decide if product branding is desired
  - Custom RCP `workbenchAdvisor` can control UI elements (name/image)
    ```java
    IWorkbenchWindowConfigurer configurer = ...;
    configurer.setTitle("Product Name");
    configurer.getShell().setImage(image);
    ```
  - The name and image can be provided by a product definition
  - RCP application must be launched as a product
  - The `Help>About [product]` menu option must be added
    ```java
    protected MenuManager createHelpMenu(IWorkbenchWindow window) {
        MenuManager menu = new MenuManager("&Help", IWorkbenchActionConstants.M_HELP);
        menu.add(new Separator());
        menu.add(new Separator());
        menu.add(ActionFactory.ABOUT.create(window));
        return menu;
    }
    ```

Note: Use of features is optional; if not used the standard `About [product]` dialog does not display feature information or options

Defining Products in Eclipse 3.1

New product editor support
Product Editor to Create Product Extension

- Eclipse 3.1 adds a wizard to create a product definition
  - New file type: name.product
  - Wizard creates file and opens editor
  - Editor makes changes in the plugin.xml (product definitions are an extension)

Summary

- Branding makes the Eclipse platform yours
  - Products branding establishes your identity on the Eclipse platform base
  - Feature branding allows for a component view, and also the ability of other cooperating software to show its identity within your product domain

- Product definitions
  - Definition by extension (with more support in 3.1)
  - Can be tested; Nothing wrong with starting the product as the launch target

- Feature branding is used as required
  - Often found for root or other key features
  - One branded feature can represent a set of plug-ins from multiple features
  - Branding content is provided by control files and content in a plug-in
Summary

- **Features**
  - Used to define structure to a set of plug-ins
  - Management unit for the delivery of new function and service to an existing Eclipse-based application
  - Support provider/feature-level branding for identity in a Workbench comprised of many participants

- **Build processing**
  - Multiple techniques provide flexibility – from a simple export to a fully automated Ant script
  - Eclipse.org process can be emulated by starting with the PDE Build templates and RelEng team guidance
    (http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/pde-build-home/articles/Automated%20Builds/article.html?rev=HEAD&content-type=te)

- **Branding**
  - Multiple levels of branding (feature, product, welcome)
  - Branding provides identity and recognition

---


- **Audience:**
  - Eclipse users
  - Plug-in developers

- **Available!!**
- Listed on amazon.com and other book sites
- ISBN 0321305027
- Publisher: Addison-Wesley
  - 1136 pages
- Make sure you get the 2nd edition
- See www.jd2e.com for a preview of the CD content