eclipseCON™ 2008

Equinox Framework: How to get Hooked

Thomas Watson, IBM Lotus

Equinox Project co-lead

Equinox Framework lead developer

Tutorial Agenda

- Equinox Framework Overview
- Equinox Adaptor Hook Architecture
- Hello Adaptor Hook
- More Advanced Adaptor Examples
  - MCache
  - PatchFragments

What is not covered

- General OSGi Bundle Development
- OSGi Services, Declarative Services etc.
- Eclipse RCP, Extension Registry etc.

This tutorial for developers interested in the implementation details of the Equinox Framework.

Requirements

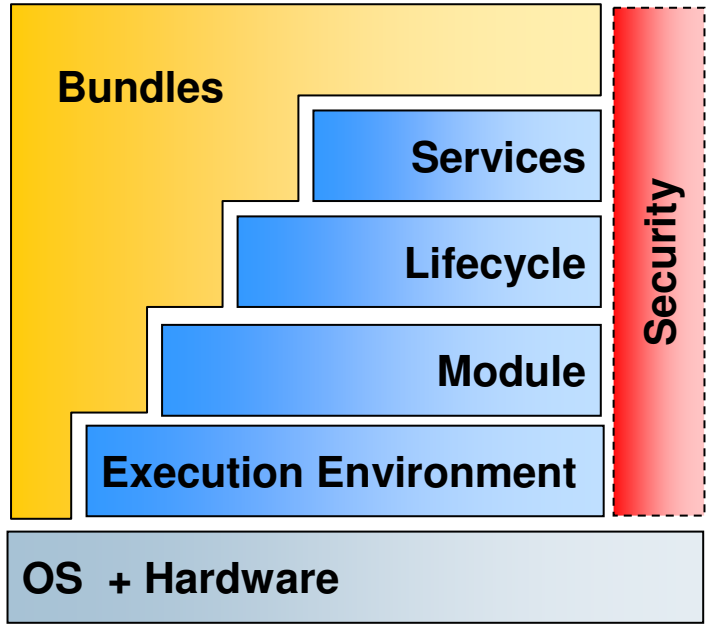
- Eclipse 3.4 M5 SDK installed

  http://download.eclipse.org/eclipse/downloads/drops/S-3.4M5-200802071530/index.php
- J2SE 1.5 SDK installed
- Example projects imported into a clean workspace

Equinox Framework Overview

What is the Equinox Framework?

- An implementation of the OSGi R4.1 Platform Core Specification (the Framework)
  - The Framework reference implementation for the OSGi specification
- The Framework is split up into different layers
  - *Execution Environmemt* – the VM
  - *Module Layer* – Module system for the Java™ Platform
  - *Lifecycle Layer* – Dynamic support
  - *Service Layer* - Service orientated



What is the Equinox Framework?

- Component Orientated
  - Building runtimes requires componentization
  - Modules are packaged as self-describing bundles
  - Strong notion of versions is built into the Framework

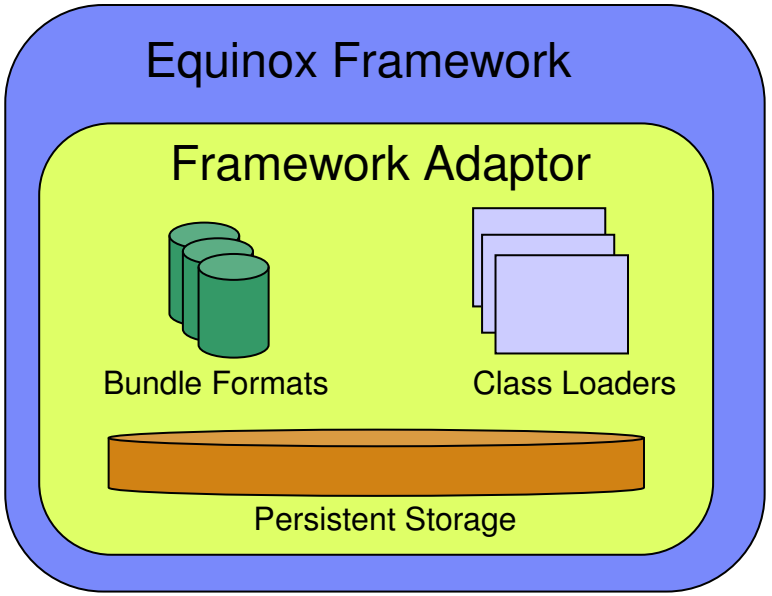
What is the Equinox Framework?

- An Extensible Framework
  - Framework fragments (extensions) provide additional function
  - Hook into the Framework Adaptor architecture

Equinox Adaptor Hook Architecture

Equinox Framework Adaptor

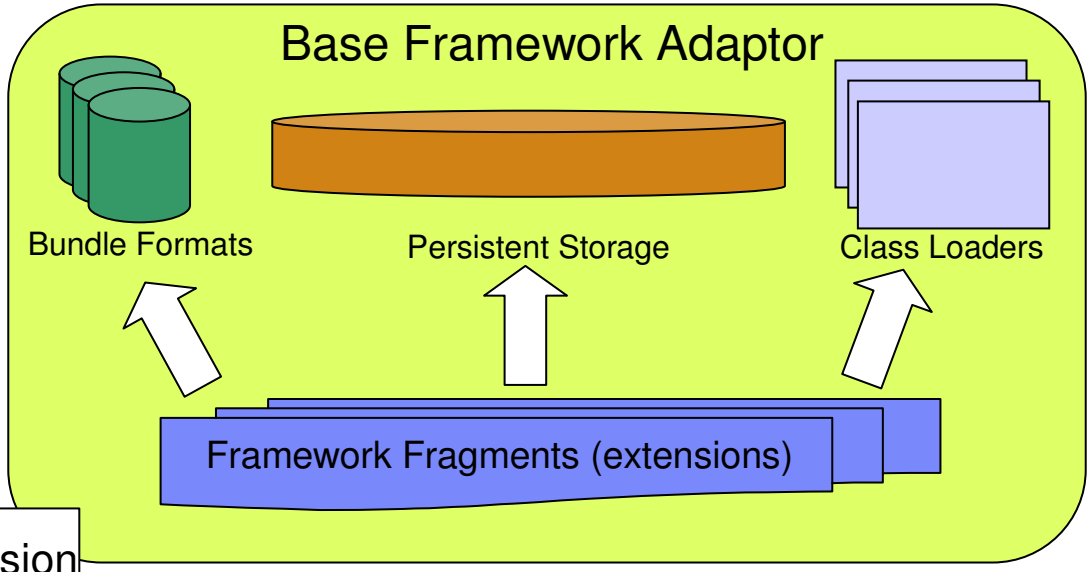
- Provides the implementation *guts* of the Framework
  - Bundle Formats – jar, directory, JXE etc.
  - Bundle Content Access
  - Class Loaders
  - Persistent Storage
- Singleton – Only one can be used
- The complete adaptor API must be implemented
- Difficult to combined adaptor implementations



Hooking into Equinox: The *Hookable* Adaptor

- Base Adaptor provides adaptor hooks
  - Allows extensions to insert additional functionality into the Framework
  - Hook interfaces are available to implement additional functionality
- Fully configurable
  - Additional extensions can be installed using system bundle fragments
  - Can be enabled and disabled using config.ini properties

- Can do many cool things
  - Class sharing
  - Load-time aspect weaving
  - Monitoring
  - Advanced caching
  - Transformation
  - Bundle Formats



config.ini

```
osgi.framework.extensions = my.extension
```


Equinox Adaptor Hook Basics

- System Bundle Fragments
  - Also called Extension Bundles in the OSGi specification
    - Defined by Section 3.15 of the OSGi Specification
  - A system bundle fragment bundle uses the framework as the host
    - Fragment-Host: org.eclipse.osgi
  - Content of the system bundle fragment are available to the Framework implementation

Equinox Adaptor Hook Basics

- Adaptor Hook Implementations
  - Delivered by system bundle fragments
  - Chicken and egg issue
    - Hook implementations provide the *guts* of the Framework
    - Must be discovered before the Framework implementation is loaded
    - Boot strap launcher must be aware of hook implementations
    - Hook Implementations cannot simply be installed like normal bundles. They must be configured into the framework

Equinox Adaptor Hook Basics

- Adaptor Hook Registry
  - Contains all the configured Hook implementations
  - The Hook Registry is populated very early in Framework initialization
  - The Base Adaptor uses the Hook Registry to discover the configured Hooks
- Adaptor Hook interfaces
  - Defines the contract between the Base Adaptor and the Hook implementations
  - Used by the Base Adaptor to call out to Hook implementations

Equinox Adaptor Hook Basics

- Hook Configurator
  - Configures Hook instances into the Hook Registry
  - Specified by the hookconfigurators.properties file
  - Implemented by a system bundle fragment

Hooks Available

- AdaptorHook – hooks into the lifecycle operations of the adaptor
  - Register services
  - Add listeners
- BundleFileFactoryHook - provides support for bundle file formats
  - Jar and Directory bundle formats are supported by default

Hooks Available

- BundleFileWrapperFactoryHook – wraps a bundle file, useful for intercepting access to bundle content
  - Advanced caching
  - Signed bundle support
  - Augment bundle content
- BundleWatcher – useful for tracking bundle lifecycle operations

Hooks Available

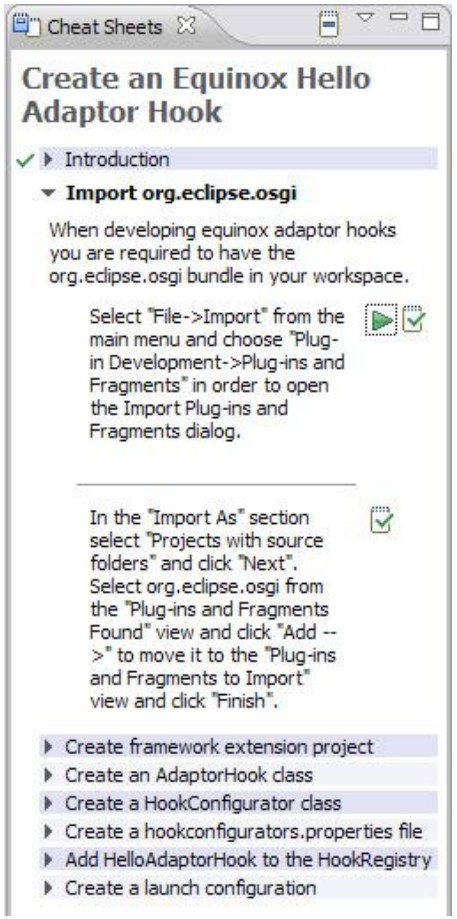
- ClassLoadingHook – add features to the bundle classloader
  - Searching for native code
  - Add classpath entries to a bundle
  - Creating classloaders
  - Byte code weaving
- ClassLoaderStatsHook – record statistics about classloading
  - Insert other operations into class loading
    - Lazy activation on first class load

Hooks Available

- ClassLoaderDelegateHook – Hooks into the OSGi delegation model
  - Insert additional package wire concept
    - Buddy ClassLoading
- StorageHook – saves and loads data for each bundle installed

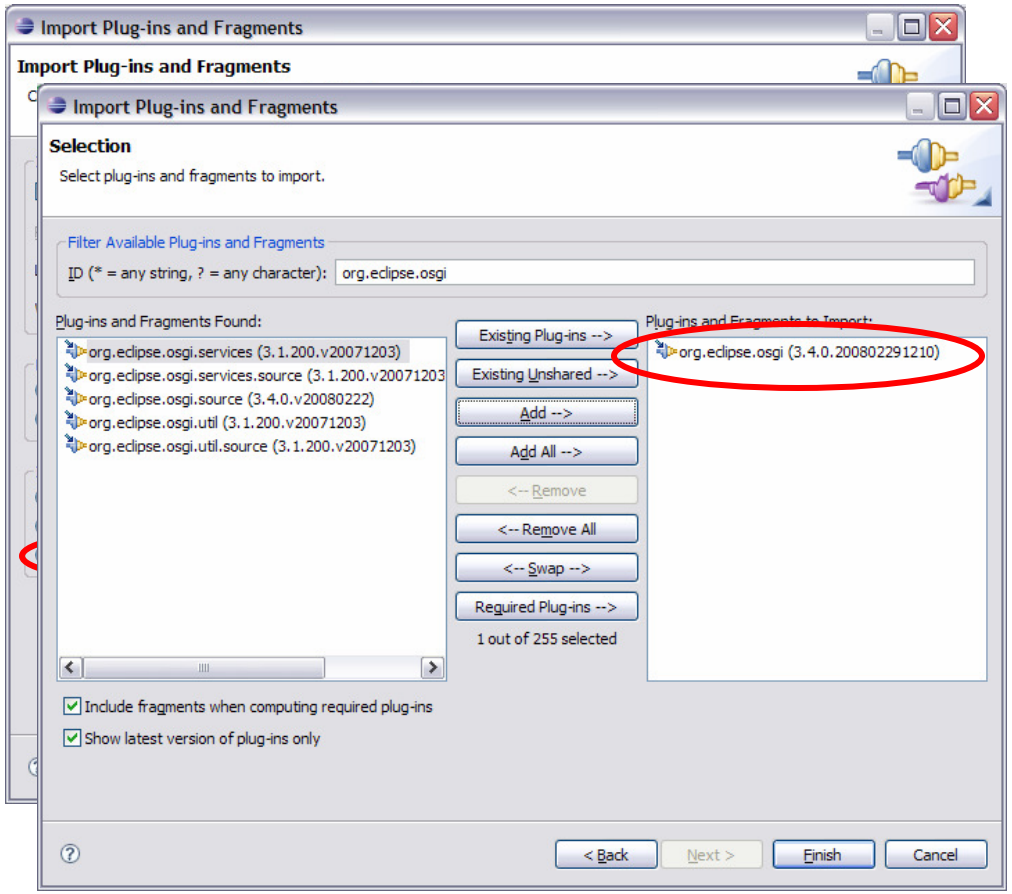
Hello Adaptor Hook

Hello Adaptor Hook Exercise



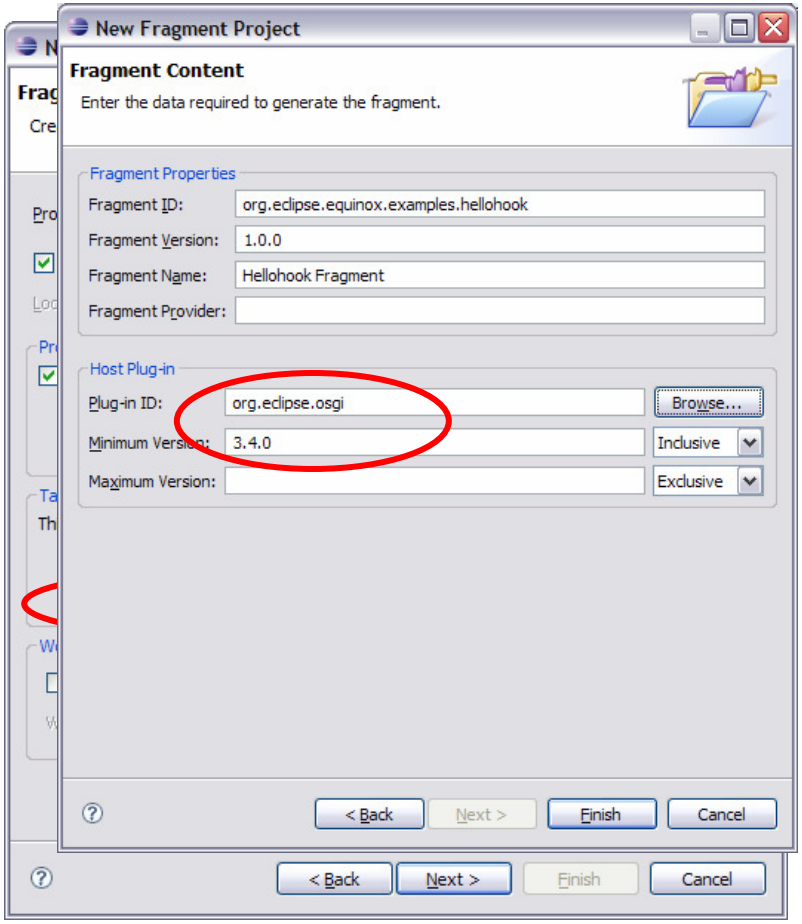
- This exercise is structured as a 7-step cheat sheet
- You use PDE to create a framework extension project
- You create implementations for AdaptorHook and HookConfigurator
- You create a hookconfigurators.properties file
- You use Eclipse Application Launcher to test the hello adaptor hook

Import Equinox Source



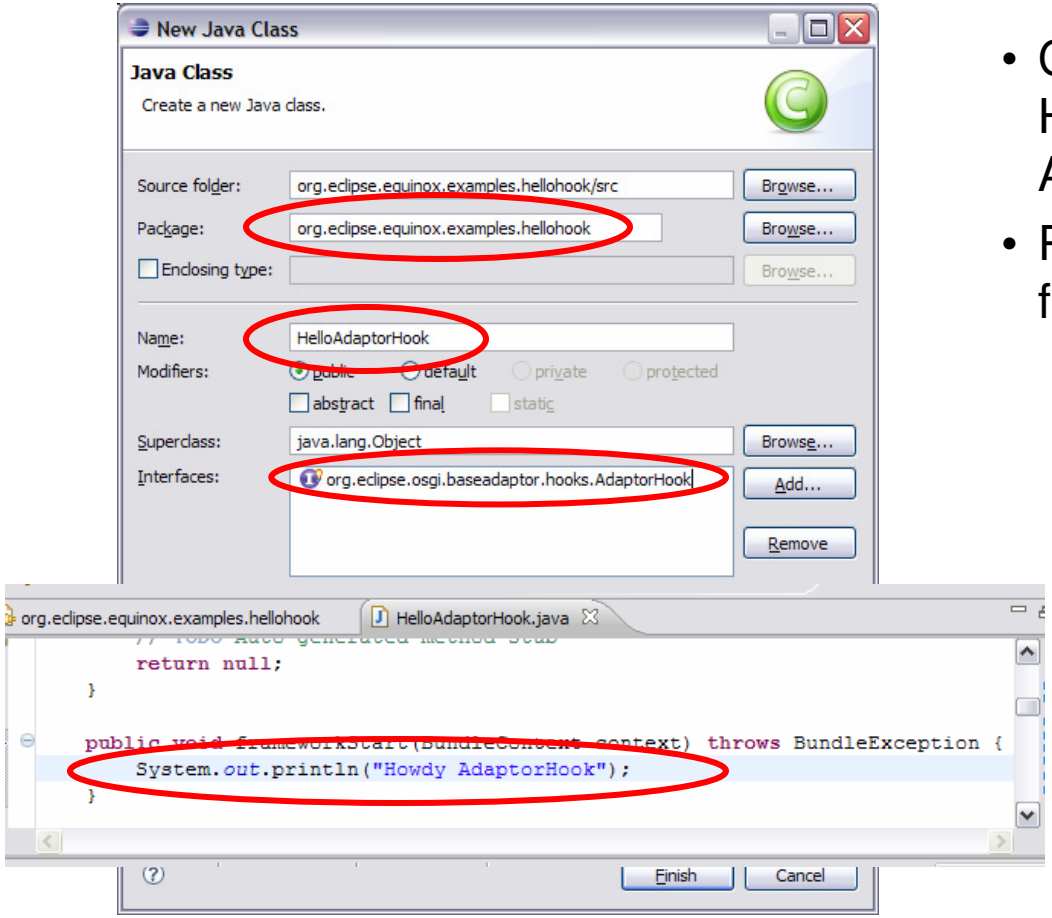
- Open The Import Plug-ins and Fragments Dialog
- Select Projects with source and click Next
- Import org.eclipse.osgi (the Equinox Framework) and click Finish

Create a Framework Extension Project



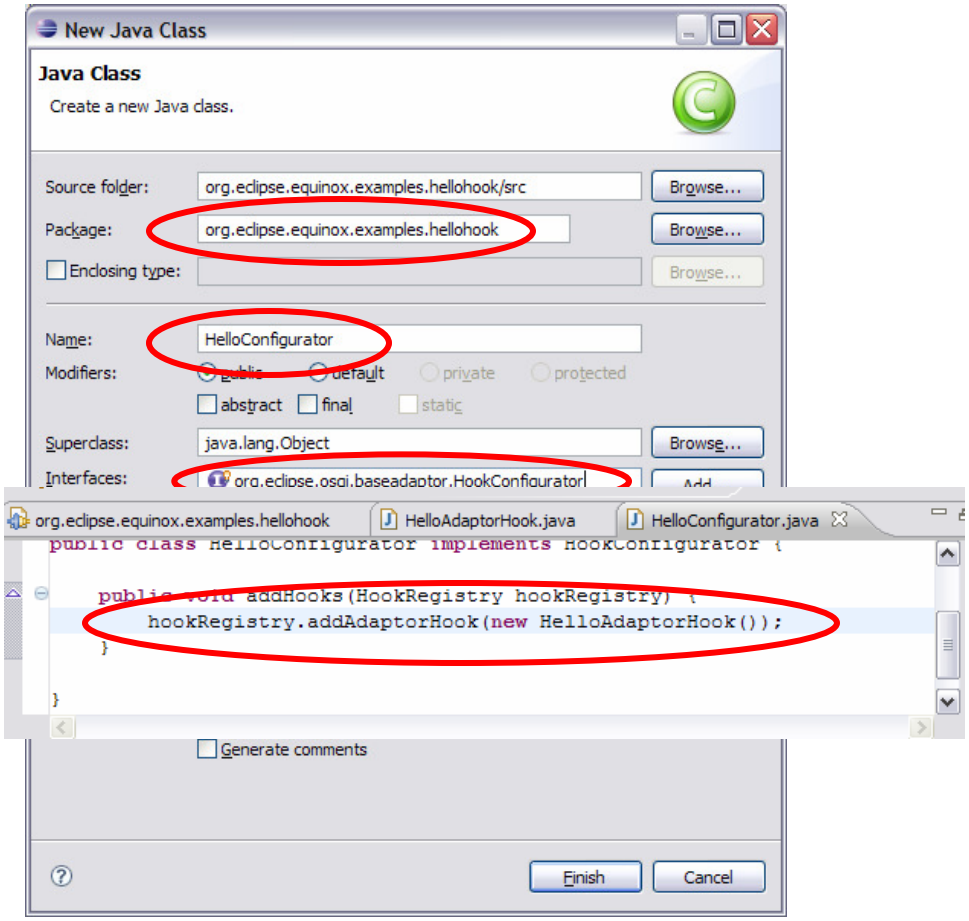
- Create a Fragment Project
- Name the project org.eclipse.equinox.examples.hellohook
- Select an OSGi framework as the Target Platform

Create an AdaptorHook



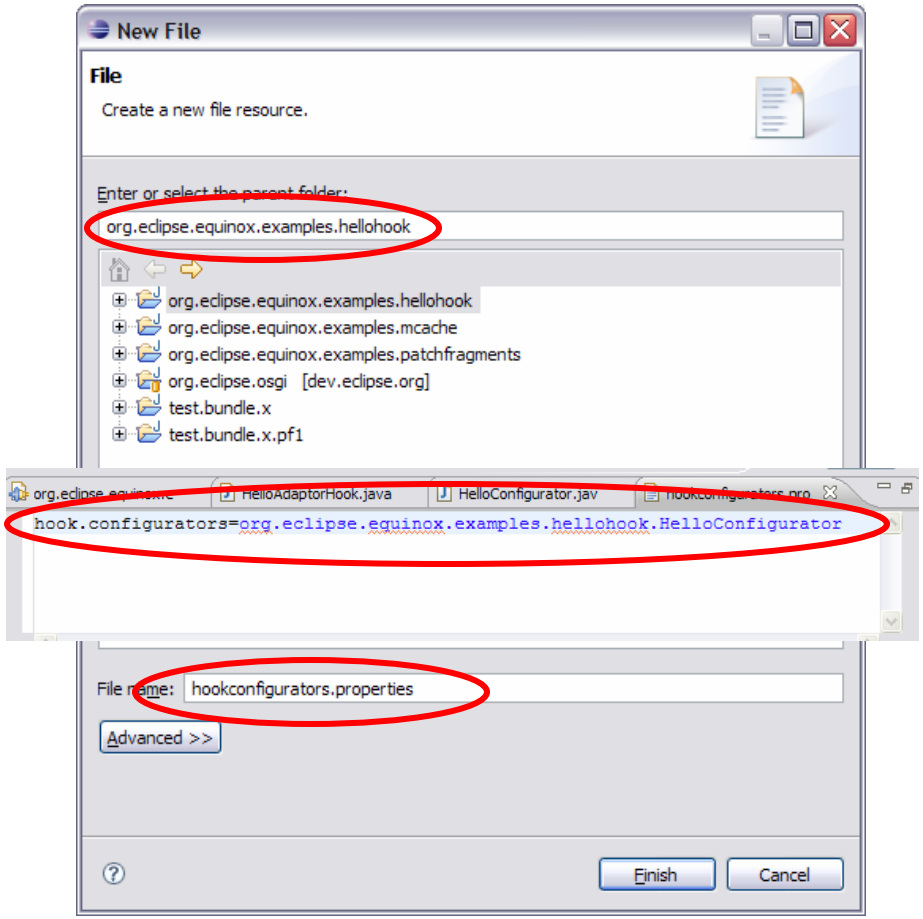
- Create a new class named HelloAdaptorHook that implements AdaptorHook interface
- Print a message from the frameworkStart() method

Create a HookConfigurator



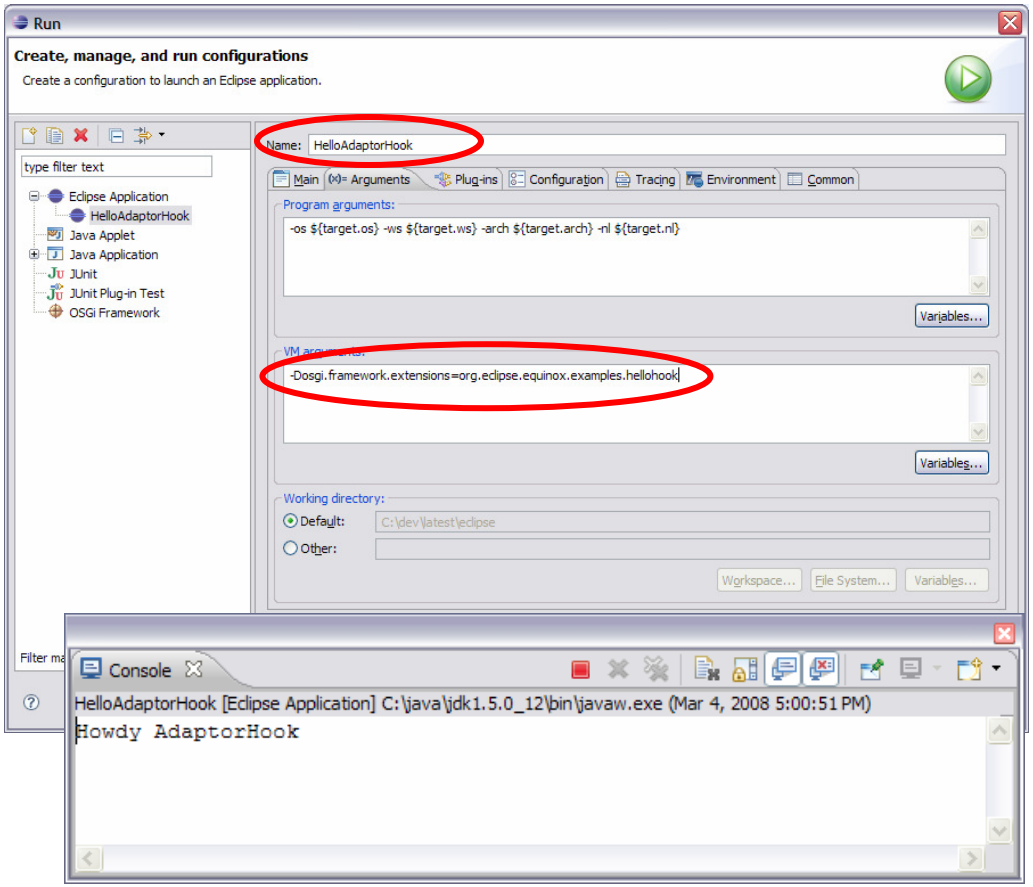
- Create a new class named HelloConfigurator that implements HookConfigurator
- Add HelloAdaptorHook to the HookRegistry

Create a hookconfigurators.properties



- Create a hookconfigurators.properties file
- Add the hook.configurators property with the HelloConfigurator specified

Create a Launch Configuration



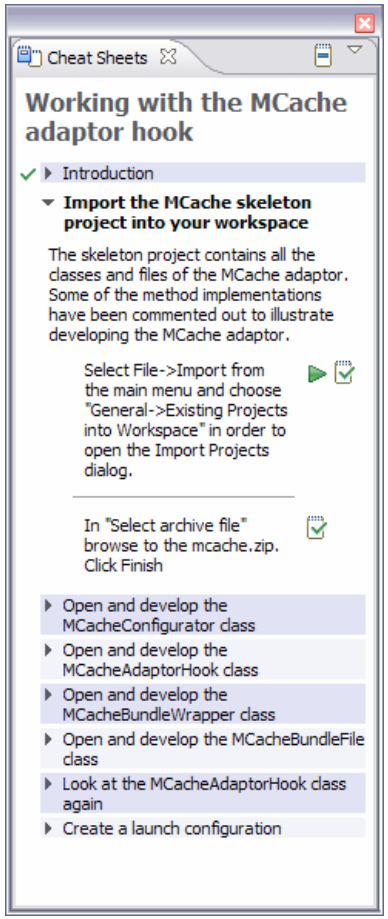
- Create an Eclipse Application launch configuration
- Name it HelloAdaptorHook
- Add the osgi.framework.extensions property with the hellohook specified
- Click on Run to launch Eclipse

MCache Adaptor

MCache Adaptor Overview

- Implements a search miss cache
- Records when a resource is requested from a bundle and the resource is not found
- The next time the resource is requested from the bundle the bundle content is not searched
- Implements the following BaseAdaptor Hooks
  - AdaptorHook
  - BundleFileWrapperFactoryHook
  - BundleFile

MCache Adaptor Exercise



- This exercise is structured as a cheat sheet
- You import the mcach project into your workspace
- You develop a HookConfigurator for the MCache
- You develop an AdaptorHook for the MCache
- You develop a BundleFileWrapperFactoryHook for the MCache
- You develop an implementation of BundleFile for the MCache

MCache HookConfigurator

- The MCacheConfigurator class implements the HookConfigurator
- The MCacheAdaptorHook and MCacheBundleWrapper hooks are added to the HookRegistry

```
    }

    public void addHooks(HookRegistry hookRegistry) {
        // the mcache adaptor hook is used to load the mcache at framework startup
        // and save the mcache at framework shutdown
        MCacheAdaptorHook w2CacheAdaptorHook = new MCacheAdaptorHook();
        hookRegistry.addAdaptorHook(w2CacheAdaptorHook);
        // a bundle file wrapper is needed to intercept bundle entry requests and check the mcache
        hookRegistry.addBundleFileWrapperFactoryHook(new MCacheBundleWrapper(w2CacheAdaptorHook));
    }

}
```


MCache AdaptorHook

- The initialize method is used to load the persistent MCache file
- The frameworkStop method is used to save the MCache file

```
 */
public void frameworkStop(BundleContext context) {
    if (!dirty)
        return; // no need to save
    dirty = false;
    File cacheFile = getCacheFile();
    // simply write a list of paths misses
    PrintWriter writer = null;
    try {
        writer = new PrintWriter(new BufferedWriter(new FileWriter(cacheFile)));
        synchronized (cache) {
            for (Iterator iPaths = cache.iterator(); iPaths.hasNext();)
                writer.println(iPaths.next());
        }
        writer.close();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (writer != null)
            writer.close();
    }
```


MCache BundleFileWrapperFactory

- A BundleFileWrapperFactory decides if a BundleFile should be wrapped
- For MCache every BundleFile is wrapped with an MCacheBundleFile except the system.bundle and content which is not File based

```
    public BundleFile wrapBundleFile(BundleFile bundleFile, Object content, BaseData data, boolean base) throws IOException {
        if (data.getBundleID() == 0)
            // it is usually a bad idea to modify the behavior of the system.bundle file.
            return null;
        if (!(content instanceof File))
            return null; // we only wrapper File content
        // The canonical file path of the content is used to create a unique index into the MCache
        String path = ((File) content).getCanonicalPath();
        // Create an MCacheBundleFile, the content path and bundle id are used to create a unique index
        MCacheBundleFile result = new MCacheBundleFile(bundleFile, data.getBundleID(), path.hashCode(), cacheAdaptorHook);
        return result;
    }

}
```


MCache BundleFile

- The MCacheBundleFile wraps another BundleFile to intercept resource requests and record when a resource is not found
- The methods getEntry, getEntryPaths, and getFile all delegate to the MCacheAdaptorHook to check the MCache

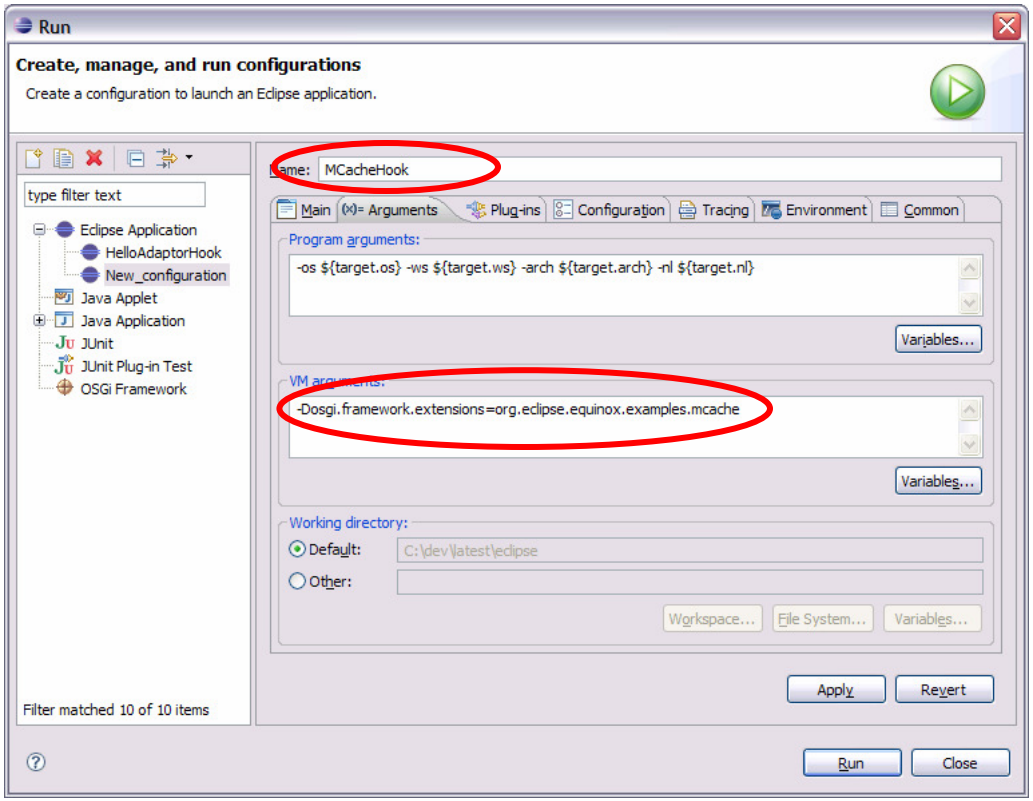
```
 * Checks the MCache for the directory.
 */
public boolean containsDir(String dir) {
    return cacheAdaptorHook.containsDir(dir, this);
}

/**
 * Checks the MCache for the path.
 */
public BundleEntry getEntry(String path) {
    return cacheAdaptorHook.getEntry(path, this);
}

/**
 * Checks the MCache for the path.
 */
public Enumeration getEntryPaths(String path) {
    return cacheAdaptorHook.getEntryPaths(path, this);
}
```


Checking the MCache

- The methods containsDir, getEntry, getEntryPaths are used to check the MCache in MCacheAdaptorHook
- If the MCache indicates that the path is missing then the methods return without checking the wrapped BundleFile
- Otherwise the wrapped BundleFile is searched
- If the path cannot be found in the wrapped BundleFile then the path is recorded in the MCache

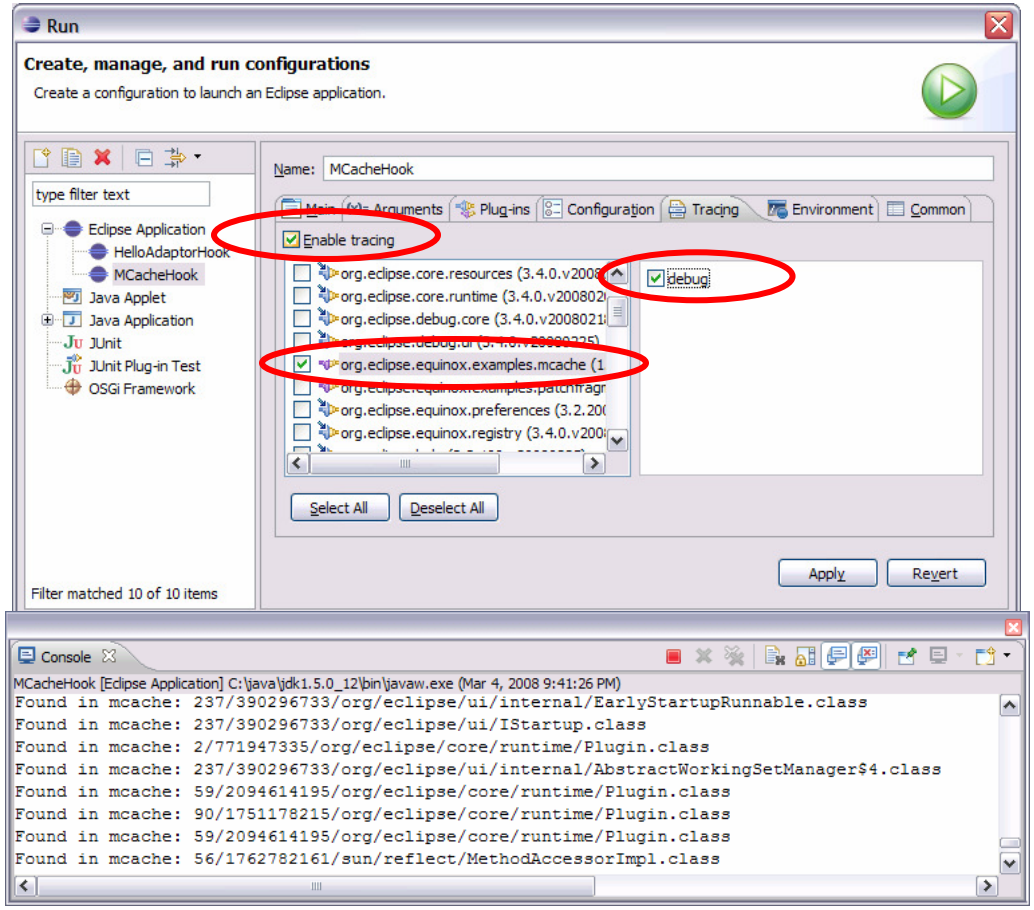
```
 */
BundleEntry getEntry(String path, MCacheBundleFile mCacheBundleFile) {
    String cachePath = checkCachePath(path, mCacheBundleFile);
    if (cachePath == null) // checkPath is null if there was a miss
        return null; // in the mcache; return null
    BundleEntry result = mCacheBundleFile.getWrappedBundleFile().getEntry(path);
    // now check the actual bundle file
    if (result == null) // did not find the path; add to mcache
        addToMCache(cachePath);
    return result;
}
```


Create a MCacheHook Launch Configuration



- Create an Eclipse Application launch configuration
- Name it MCacheHook
- Add the osgi.framework.extensions property with the mcache specified
- Click on Run to launch Eclipse
- Wow not much to see!!

Enable MCache tracing



- Open the MCacheHook launch configuration and go to the tracing tab
- Enable tracing
- Select the mcache bundle and select the debug option
- Click Run to launch again
- Observe the trace messages from the MCache

PatchFragments Adaptor

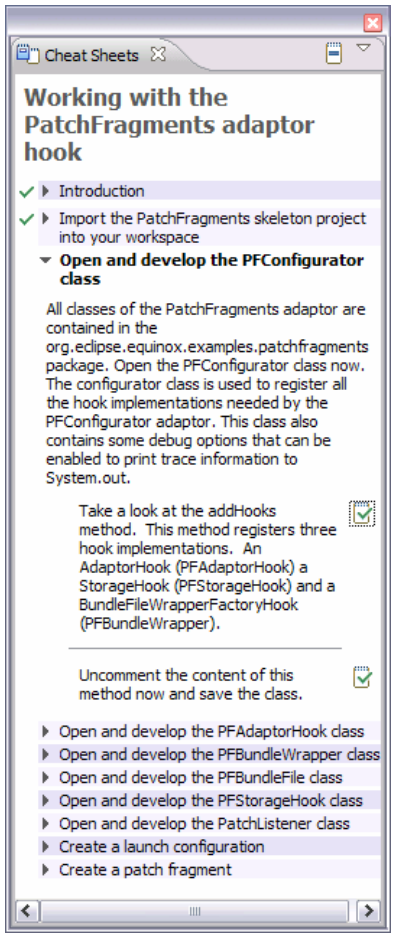
PatchFragments Adaptor Overview

- Implements the patch fragments feature
- Patch fragments are able to insert content before the content of their host bundle
- Allows fragments to be installed to fix bugs in their host bundles
- A new bundle manifest header is introduced to specify a patch fragment

```
Equinox-BundleType: patch.fragment
```

- The following BaseAdaptor hooks are implemented
  - AdaptorHook
  - BundleFileWrapperFactoryHook
  - BundleFile
  - StorageHook

PatchFragments Adaptor Exercise



- This exercise is structured as a cheat sheet
- You import the patchframents project into your workspace
- You develop a HookConfigurator for the PatchFragments adaptor
- You develop an AdaptorHook for the PatchFragments adaptor
- You develop a BundleFileWrapperFactoryHook for the PatchFragments adaptor
- You develop an implementation of BundleFile for the PatchFragments adaptor
- You develop a StorageHook for the PatchFragments adaptor

PatchFragments HookConfigurator

- The PFConfigurator class implements the HookConfigurator
- The PFAdaptorHook, PFBundleWrapper and PFStorageHook hooks are added to the HookRegistry

```
public void addHooks(HookRegistry hookRegistry) {
    // this is where you add the needed hooks

    // an adaptor hook is needed to track PackageAdmin and add a BundleListener
    PFAdaptorHook adaptorHook = new PFAdaptorHook();
    hookRegistry.addAdaptorHook(adaptorHook);
    // a storage adaptor is needed to record the fragments which are patch fragments
    hookRegistry.addStorageHook(new PFStorageHook());
    // a bundle file wrapper is needed to intercept bundle entry requests to allow patched content
    hookRegistry.addBundleFileWrapperFactoryHook(new PFBundleWrapper(adaptorHook));
}
```


PatchFragments AdaptorHook

- The frameworkStart method uses a ServiceTracker to track the PackageAdmin Service and registers the PatchListener as a BundleListener
- The frameworkStop method closes the ServiceTracker and removes the BundleListener

```
 */
public void frameworkStart(BundleContext context) {
    paTracker = new ServiceTracker(context, PackageAdmin.class.getName(), null);
    paTracker.open();
    patchListener = new PatchListener();
    context.addBundleListener(patchListener);
}

/**
 * Removes the PatchListener and closes the PackageAdmin ServiceTracker.
 */
public void frameworkStop(BundleContext context) {
    context.removeBundleListener(patchListener);
    patchListener = null;
    paTracker.close();
    paTracker = null;
}
```


PatchFragments BundleFileWrapperFactory

- A BundleFileWrapperFactory decides if a BundleFile should be wrapped
- For PatchFragments every BundleFile is wrapped with a PFBundleFile except the system.bundle

```
public class PFBundleWrapper implements BundleFileWrapperFactoryHook {
    private final PFAdaptorHook adaptorHook;

    public PFBundleWrapper(PFAdaptorHook adaptorHook) {
        this.adaptorHook = adaptorHook;
    }

    public BundleFile wrapBundleFile(BundleFile bundleFile, Object content, BaseData data, boolean base) {
        if (data.getBundleID() == 0)
            // it is usually a bad idea to modify the behavior of the system.bundle file.
            return null;
        // at this point we do not know if the BaseData is a host or a fragment;
        // we just create a PFBundleFile for all bundles
        return new PFBundleFile(bundleFile, data, adaptorHook);
    }
```


PatchFragments BundleFile

- The PFBundleFile wraps another BundleFile to intercept resource/class requests
- The methods getEntry, and getFile get a list of patch fragment BundleFiles and delegates to them before delegating to the wrapped BundleFile

```
public BundleEntry getEntry(String path) {
    // see if there are any patches available
    BundleFile[] patchFiles = getPatches();
    if (patchFiles == null) // none available just use the wrapped content
        return wrapped.getEntry(path);
    if ("META-INF/MANIFEST.MF".equals(path)) //$NON-NLS-1$
        return wrapped.getEntry(path); // don't patch manifest
    for (int i = 0; i < patchFiles.length; i++) {
        BundleEntry entry = patchFiles[i].getEntry(path);
        if (entry != null) { // found patched content; return it
            if (PFConfigurator.DEBUG)
                System.out.println("Found patch for \"" + path + "\" in \"" + patchFiles[i] + "\"");
            return entry;
        }
    }
    // no patched content found for the path; use the wrapped content
    return wrapped.getEntry(path);
}
```


Discovering Patch Fragments

- The PFBundleFile object discovers the available patch fragments in the getPatches method
- The code has too many lines to illustrate. Here are the steps it uses:
  - Checks if the bundle is resolved. A bundle cannot have patches fragments attached unless it is resolved
  - Use PackageAdmin to get the list of attached fragments
  - Check each attached fragment for patches. This is done by using the PFStorageHook which is explained later
  - After the patch fragment list is constructed the PatchListener is informed about the patches it must listen to. The PatchListener is explained later
  - Finally we mark the patches as processed so we do not have to discover them again

PatchFragments StorageHook

- StorageHooks are a bit different than other adaptor hooks
  - StorageHooks play two roles
    - A factory for creating StorageHook instances for each bundle installed
    - A bundle instance which contains data about a single bundle
  - StorageHooks implement KeyedElement for quick lookup by the BaseAdaptor
    - By convention the key used by a StorageHook is the implementation class name
- In the PatchFragments adaptor a StorageHook is used to store whether a fragment is a patch fragment
  - The metadata for declaring a patch fragment is the Equinox-BundleType header
  - The PFStorageHook parses this header and stores the patch fragment data

PFragments StorageHook Conventions

- Use the PFStorageHook class name as the key

```
/**
 * The convention is to use the class name as the key for the keyed element
 * for storage hooks
 */
public static final String KEY = PFStorageHook.class.getName();
/**
 * The key hash code is constant
 */
public static final int HASHCODE = KEY.hashCode();
```

```
public int getKeyHashCode() {
    return HASHCODE;
}

public boolean compare(KeyedElement other) {
    return other.getKey() == KEY;
}

public Object getKey() {
    return KEY;
}
```


Create, Save and Load PFStorageHook

- The create and load methods are factory methods which construct new PFStorageHook instances.
- The create method constructs new StorageHook instances for a newly installed bundle
- The load method constructs StorageHook instances for all the installed bundles at framework launch time and loads the patchFragment flag from persistent storage
- The save method persists the patchFragment flag from persistent storage

```
public StorageHook create(BaseData bundledata) {
    return new PFStorageHook();
}
```

```
     */
    public void save(DataOutputStream os) throws IOException {
        os.writeBoolean(patchFragment);
    }
```

```
public StorageHook load(BaseData bundledata, DataInputStream is) throws IOException {
    // This method should always create a new storage hook object to load the data into
    PFStorageHook loadHook = new PFStorageHook();
    loadHook.patchFragment = is.readBoolean();
    return loadHook;
}
```


Check the Equinox-BundleType header

- The initialize method is used to parse bundle manifests for newly installed bundles
- For patch fragments the Equinox-BundleType header is check for the patch.fragment value. This only needs to be done for bundles that have the Fragment-Host header
- A boolean patchFragment flag is set to indicate if the bundle is a patch fragment

```
/
 * Checks the manifest for a patch fragment
 */
public void initialize(Dictionary manifest) {
    // make sure this is a fragment manifest
    if (manifest.get(Constants.FRAGMENT_HOST) == null)
        return; // not a fragment;
    String type = (String) manifest.get(BUNDLE_TYPE_HEADER);
    patchFragment = BUNDLE_TYPE_PATCH.equals(type);
}
```

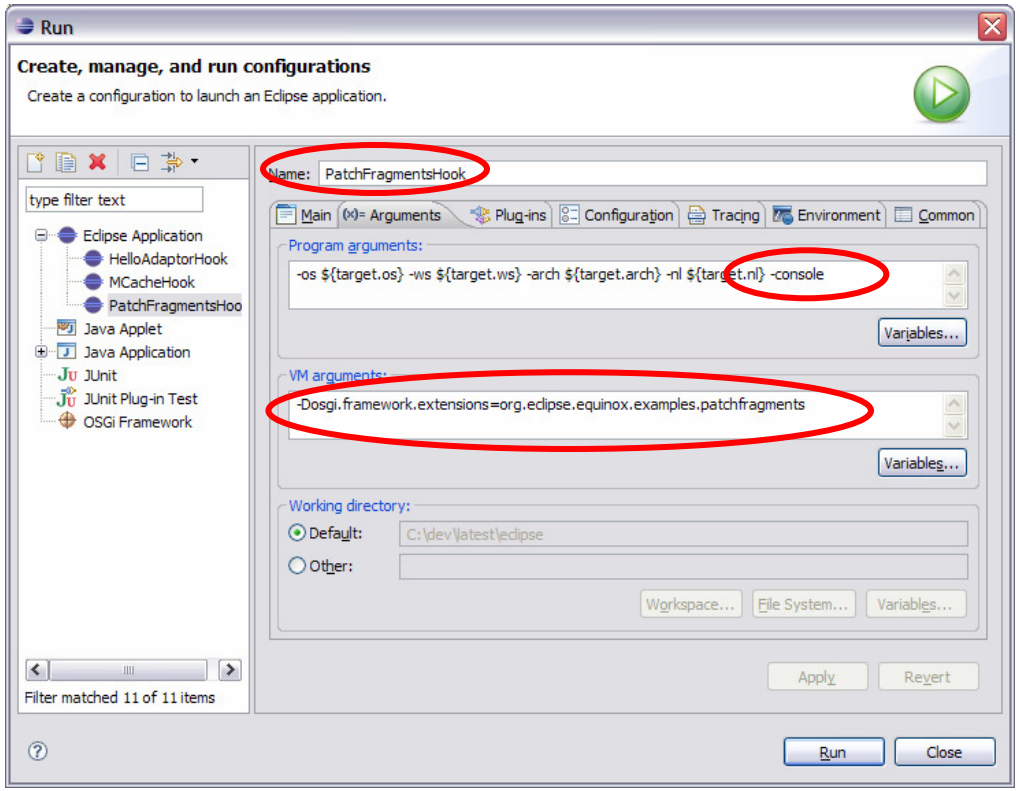

Listening for Patch Fragments

- The bundleChanged method used to listen to BundlEvents
- The PatchListener is only interested in the UNINSTALLED and UNRESOLVED events
- When one of these events occurs the patches are searched to see if the event applies to a PFBundleFile
- The listenToPatches method is used to associate a list of bundles that should be listened to for a PFBundleFile

```
public void bundleChanged(BundleEvent event) {
    if ((event.getType() & (BundleEvent.UNINSTALLED | BundleEvent.UNRESOLVED)) == 0)
        return; // only reset on resolved/unresolved and uninstalled events
    synchronized (patches) {
        for (Iterator entries = patches.entrySet().iterator(); entries.hasNext();) {
            Map.Entry entry = (Entry) entries.next();
            Collection bundles = (Collection) entry.getValue();
            if (bundles.contains(event.getBundle())) {
                PFBundleFile bundleFile = (PFBundleFile) entry.getKey();
                bundleFile.resetPatches();
                entries.remove();
            }
        }
    }
}
```

```
public void listenToPatches(Collection bundles, PFBundleFile patched) {
    synchronized (patches) {
        patches.put(patched, bundles);
    }
}
```


Create a PatchFragmentsHook Launch Configuration



- Create an Eclipse Application launch configuration
- Name it PatchFragmentsHook
- Add the osgi.framework.extensions property with the patchfragments specified and add the –console option
- Click on Run to launch Eclipse
- Wow not much to see!!
- We need a bundle with a bug to patch

Open a bundle project with a bug

- Open the test.bundle.x project
- This project contains a bundle with a bad BundleActivator start method
- Launch the PatchFragmentsAdaptor again. At the console enter:

```
osgi> start test.bundle.x
```

```
 */
public void start(BundleContext context) throws Exception {
    System.out.println("Hello World!!");
    Integer test = null;
    System.out.println(test.toString());
}
```

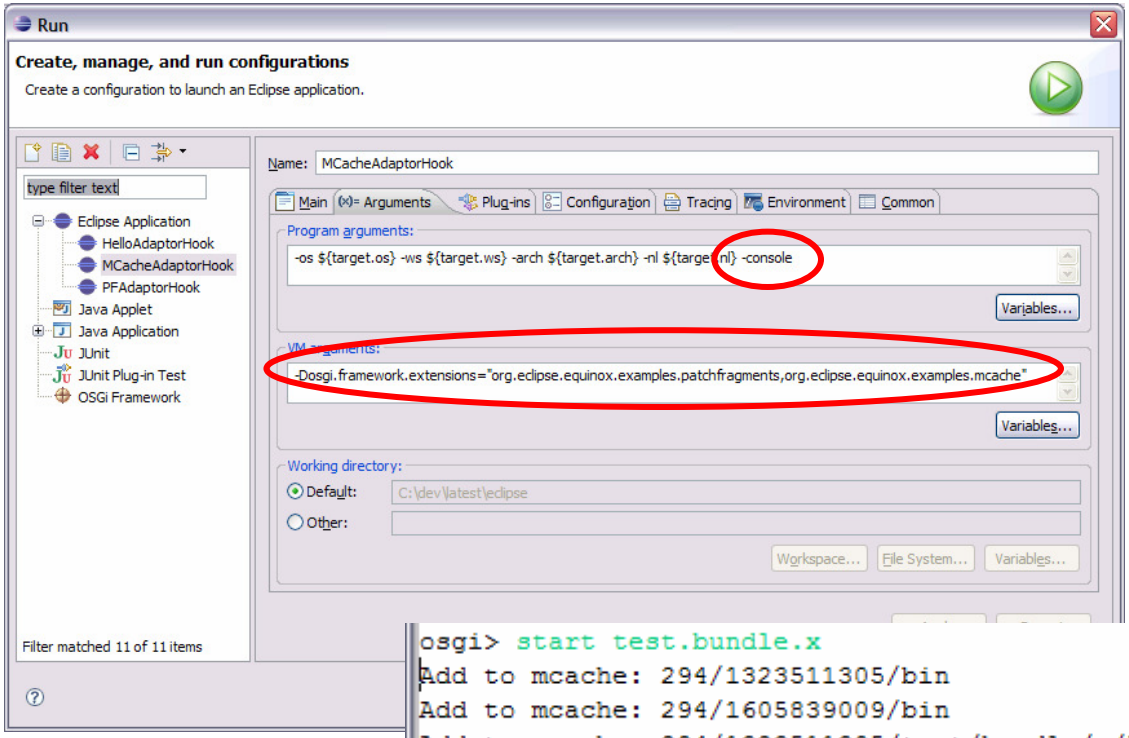
```
osgi> start test.bundle.x
Hello World!!
org.osgi.framework.BundleException: Exception in test.bundle.x.Activator.start() of bundle test.bundle.x.
        at org.eclipse.osgi.framework.internal.core.BundleContextImpl.startActivator(BundleContextImpl.java:1028)
        at org.eclipse.osgi.framework.internal.core.BundleContextImpl.start(BundleContextImpl.java:984)
        at org.eclipse.osgi.framework.internal.core.BundleHost.startWorker(BundleHost.java:346)
        at org.eclipse.osgi.framework.internal.core.AbstractBundle.start(AbstractBundle.java:265)
        at org.eclipse.osgi.framework.internal.core.AbstractBundle.start(AbstractBundle.java:257)
        at org.eclipse.osgi.framework.internal.core.FrameworkCommandProvider.start(FrameworkCommandProvider.java:257)
```


Open a patch fragment project

- Open the test.bundle.x.pf1 project
- This fragment contains a fixed BundleActovator
- Specifies the `Equinox-BundleType: patch.fragment` header
- Launch the PatchFragmentAdaptor again

```
osgi> Hello World!!  This is a patch fragment!!
5000
```


Enable multiple adaptors




```
osgi> start test.bundle.x
Add to mcache: 294/1323511305/bin
Add to mcache: 294/1605839009/bin
Add to mcache: 294/1323511305/test/bundle/x/Activator.class
Add to mcache: 293/1324868391/META-INF/MANIFEST.MF
Hello World!!  This is a patch fragment!!
5000
```

- Open the MCachAdaptor launch configuration
- Add patchfragments to the osgi.framework.extensions property
- Add –console to the program arguments
- Click Run to launch Eclipse

Legal Notices

- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both
- Other company, product, or service names may be trademarks or service marks of others

