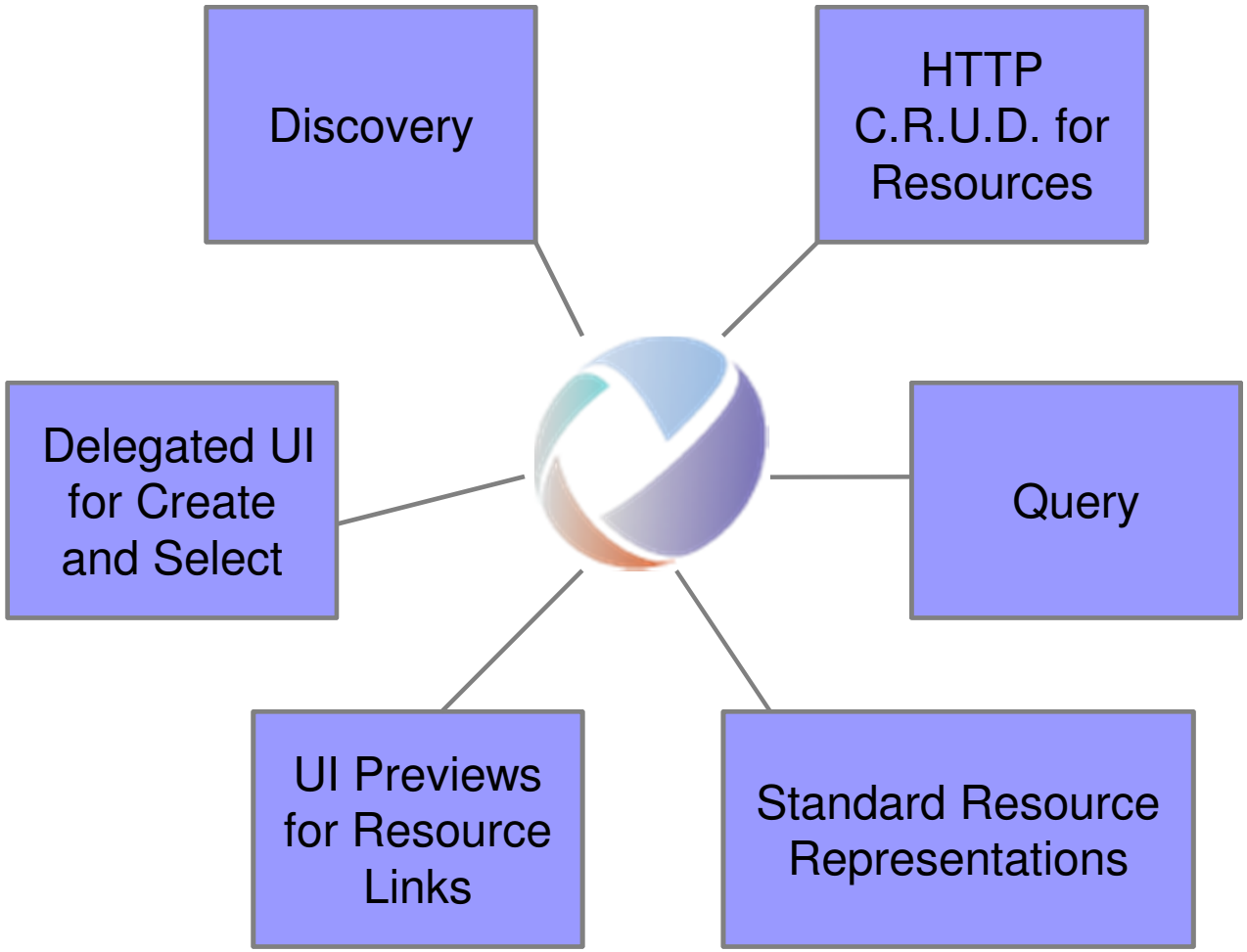
How to implement an OSLC Provider with OSLC4J

Stefan Paschke, ViF

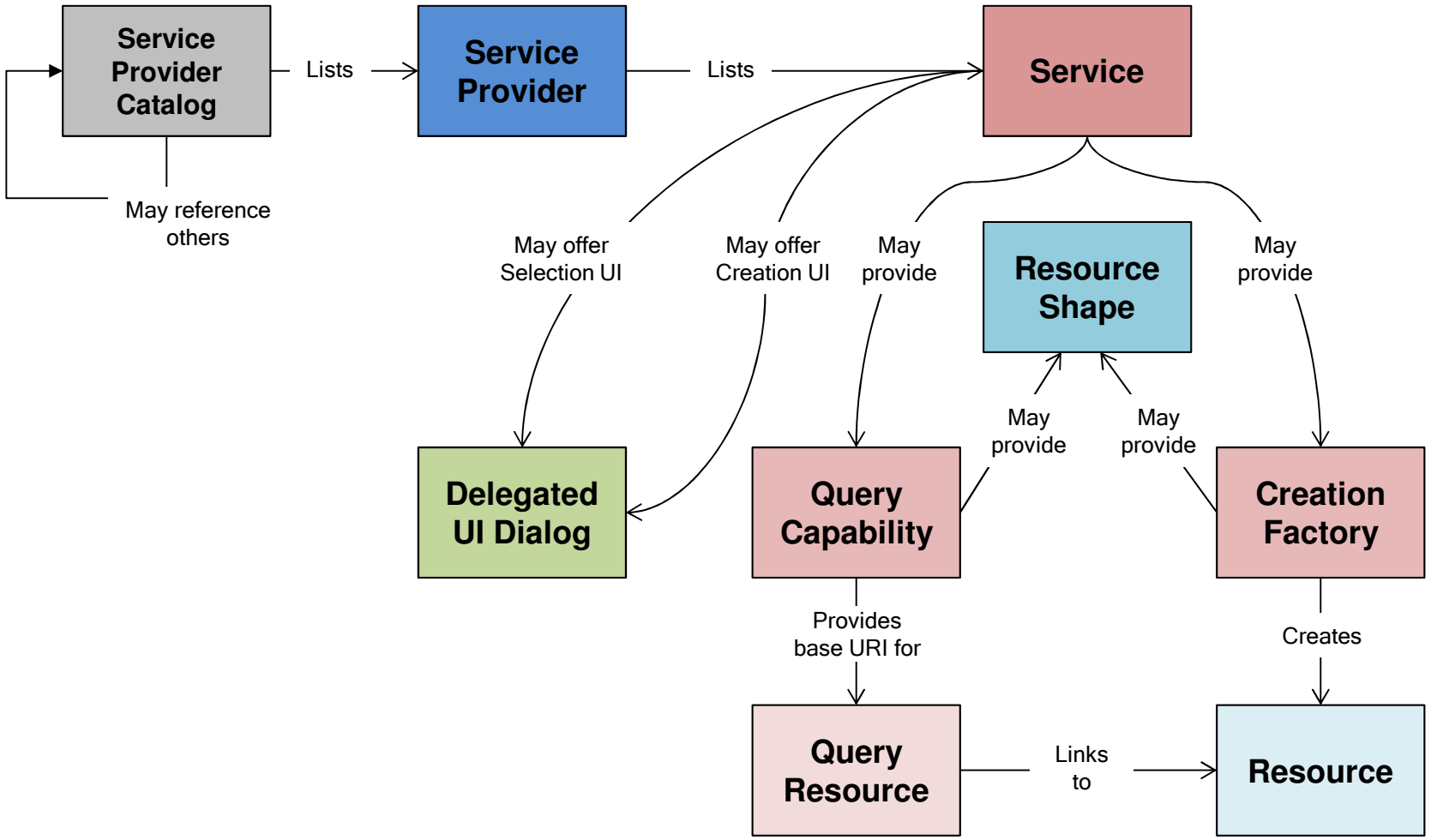
25th April 2013

Table of Contents

OSLC - Architecture

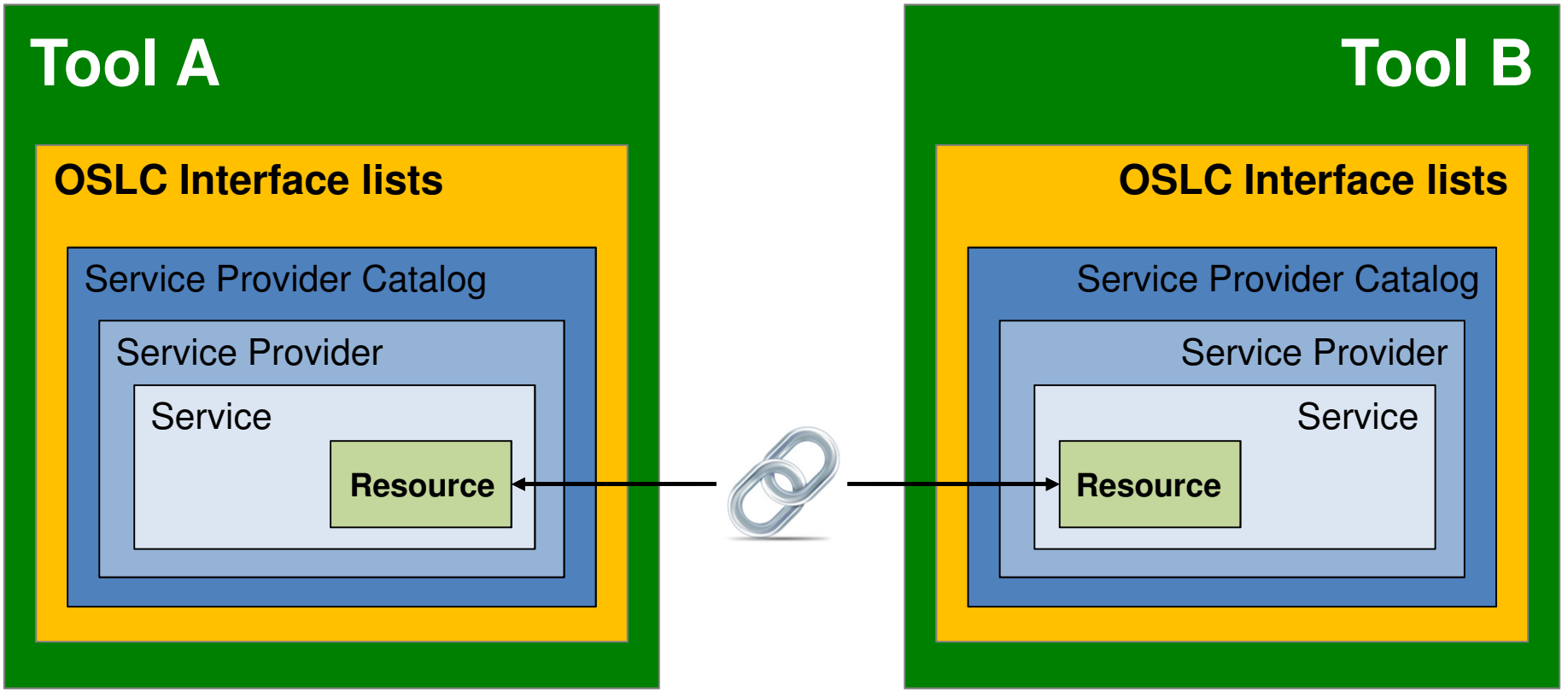


OSLC Service Description Documents

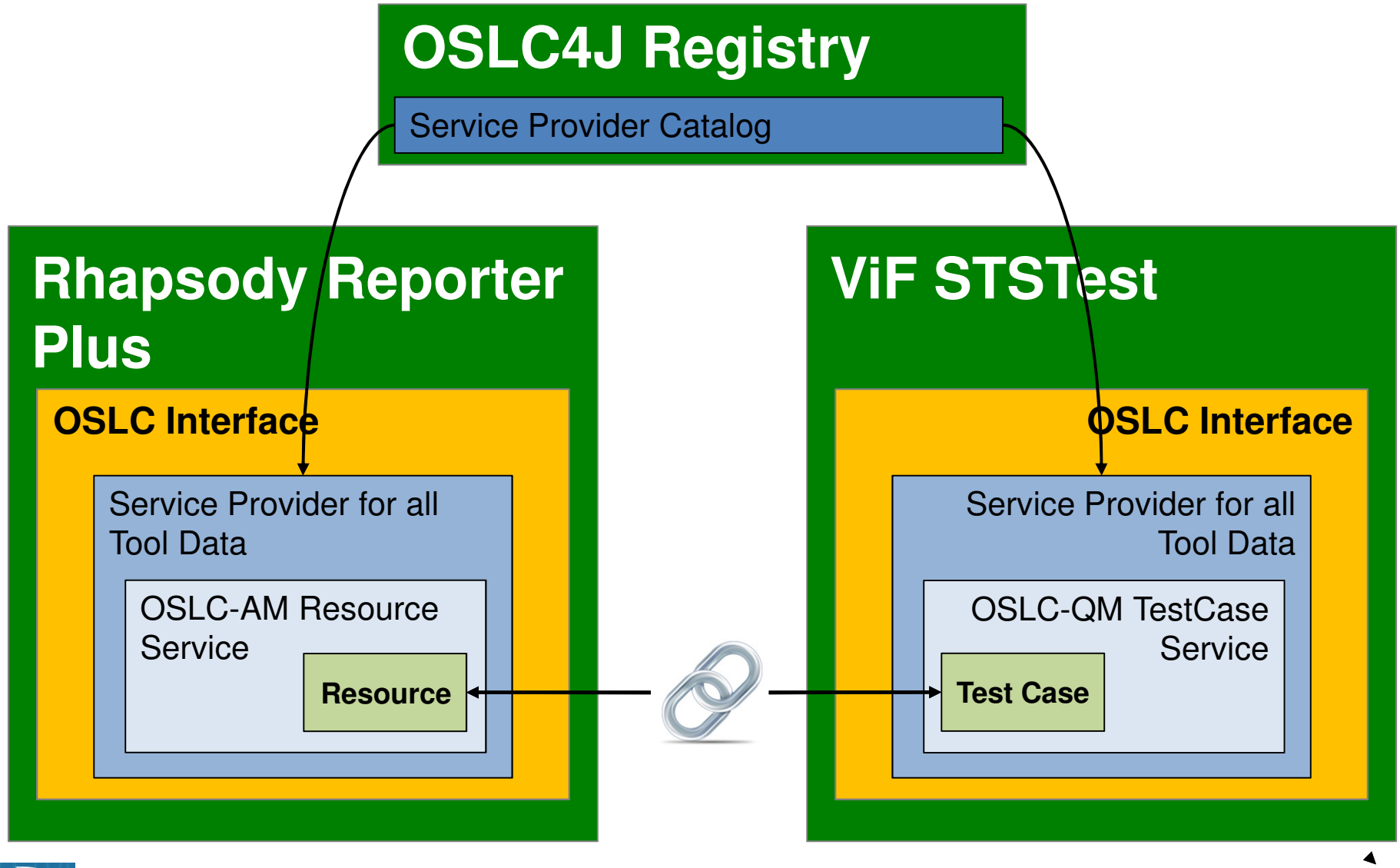


- What scenario do you want to support?
- Scenario = Scope + Tools
- Scope
  - What do you want to achieve to what extend?
- Tools
  - What type of artifacts need to be exchanged? http://open-services.net/specifications/
- Interface
  - Connects things with one another.
  - Existing for involved tools? One could say work is done! ;-)
  - OSLC support build in Tool?

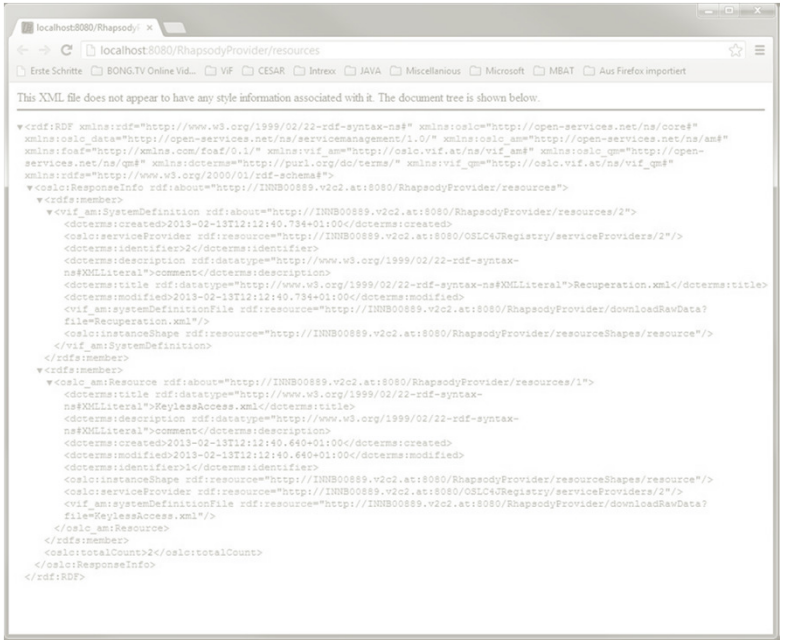
Integration on basis of OSLC



Integration on basis of OSLC Example

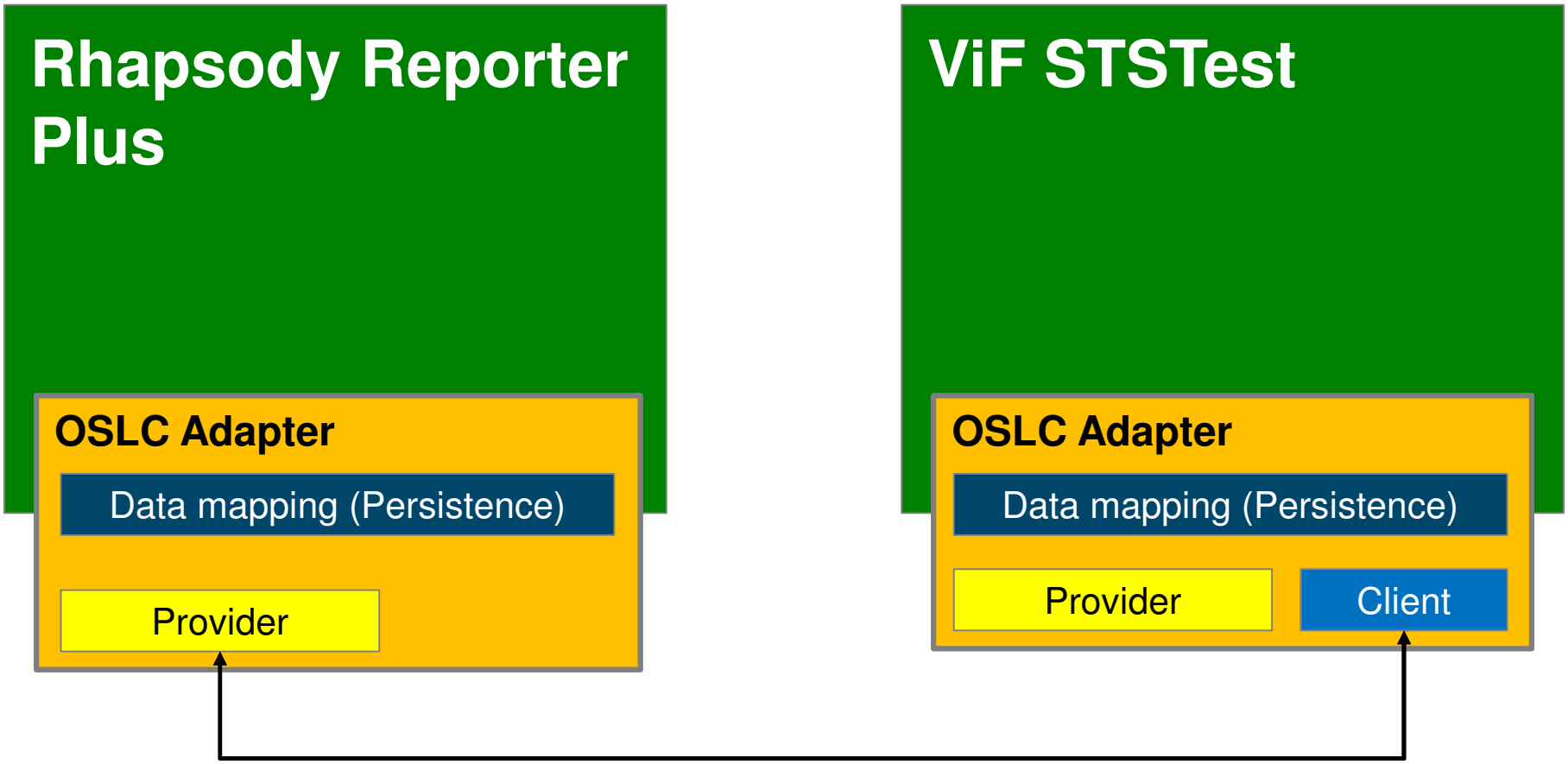


- Service Provider Catalog
  http://localhost:8080/OSLC4JRegistry/catalog
- Service Provider
  http://innb00889.v2c2.at:8080/OSLC4JRegistry/serviceProviders/2
  lists inline
  - services
  - query capability
  - base URI for query resource
    http://localhost:8080/RhapsodyProvider/resources
  - resource shape
    http://localhost:8080/RhapsodyProvider/resourceShapes/resource



- Native Support
  - Integrated tool is a web application
  - Change tool‘s code to support OSLC
- Plug-in
  - Integrated tool is a web application
  - Add OSLC support by writing code that plugs-in to the tool
  - Using add-on API
- Adapter
  - New web application as OSLC Adapter
  - Provides OSLC support
  - Example of “Under the hood” creation, retrieval, update and delete
    - Web-API
    - conventional API
    - direct database access

OSLC Integration - Adapter Concept



How to implement a Provider

- Implement Resource Class
- Implement Provider Logic
  - Setup the JAVA EE Web Application
  - Implement Service for each resource type
    - Provision of Resources – HTTP GET
    - Altering Resources
      - HTTP PUT, POST, DELETE
    - Route requests to underlying logic
  - Other Configurations
- Running the provider
- For Testing
  - Implement Junit Tests
  - Implement Consumer logic
    - Retrieve
    - Update
    - (Create, Delete)

Resource Class Implementation

Resource Class Implementation

- What specification do you want to consume?
  - http://open-services.net/specifications/
  - OSLC-QM, OSLC-RM, OSLC-AM
- Checkout Resource definition
  - Architecture_Management#Resource
  - some properties are always there
    - rdfTypes, title, identifier, created, modified, serviceProvider, instanceShape
- Mapping of POJO (Plain old Java Object) to OLSC representation (RDF-XML)

Resource Class Implementation

- Define needed Constants
  - `String ARCHITECTURE_MANAGEMENT_DOMAIN = "http://open-services.net/ns/am#";`
  - `String ARCHITECTURE_MANAGEMENT_PREFIX = "oslc_am";`
  - `String TYPE_RESOURCE = ARCHITECTURE_MANAGEMENT_NAMESPACE + "Resource";`
  - …
- `extends AbstractResource`
- Implement Specification
  - One Class for each type
    - Architecture_Management#Resource
    - foaf#Person
  - One variable for each property of an OSLC Resource
  - some properties are always there
    - rdfTypes, title, identifier, created, modified, serviceProvider, instanceShape

Resource Class Implementation

- Implement Specification
  - Getters and setters for each variable
  - Add OSLC Annotations to getter
  - Each Getter is associated with an OSLC Resource property

OSLC Annotations

- Assist with the serialization and deserialization of Java objects to
  - RDF
  - JSON
- Facilitate automatic creation of
  - OSLC resource shape documents
  - service provider documents
  - service provider catalog

- Class Annotations
  - `@OslcNamespace` defines Namespace for the resource class.
  - `@OslcResourceShape` specifies title and type for the resource shape.
- Method Annotations
  - `@OslcOccurs` provides the cardinality of the attribute.
  - `@OslcPropertyDefinition` providers the namespace qualified attribute name.
  - `@OslcReadOnly` indicates this attribute should appear in the resource shape as read only.
  - `@OslcValueType` indicates the type of the attribute.
    Is not there often, because the default type in OSLC4J is a string.

Resource Class Implementation

- Implement Specification
  - Getters and setters for each variable
  - Add OSLC Annotations to getter
  - Each Getter is associated with an OSLC Resource property

```
@OslcNamespace(Constants.ARCHITECTURE_MANAGEMENT_NAMESPACE)
@OslcResourceShape(title = "Architecture Management Resource Resource Shape", describes =
    Constants.TYPE_RESOURCE)
public class Resource extends AbstractResource {
…
    @OslcDescription("Title (reference: Dublin Core) or often a single line summary …")
    @OslcOccurs(Occurs.ExactlyOne)
    @OslcPropertyDefinition(OslcConstants.DCTERMS_NAMESPACE + "title")
    @OslcTitle("Title")
    @OslcValueType(ValueType.XMLLiteral)
    public String getTitle() {
        return this.title;
    }
…
}
```

Resource Class Implementation

- Provide Namespaces used in all resource classes in a package
- package-info.java
  - Alongside with resource class implementation
  - Add it in the same java package as the resource classes

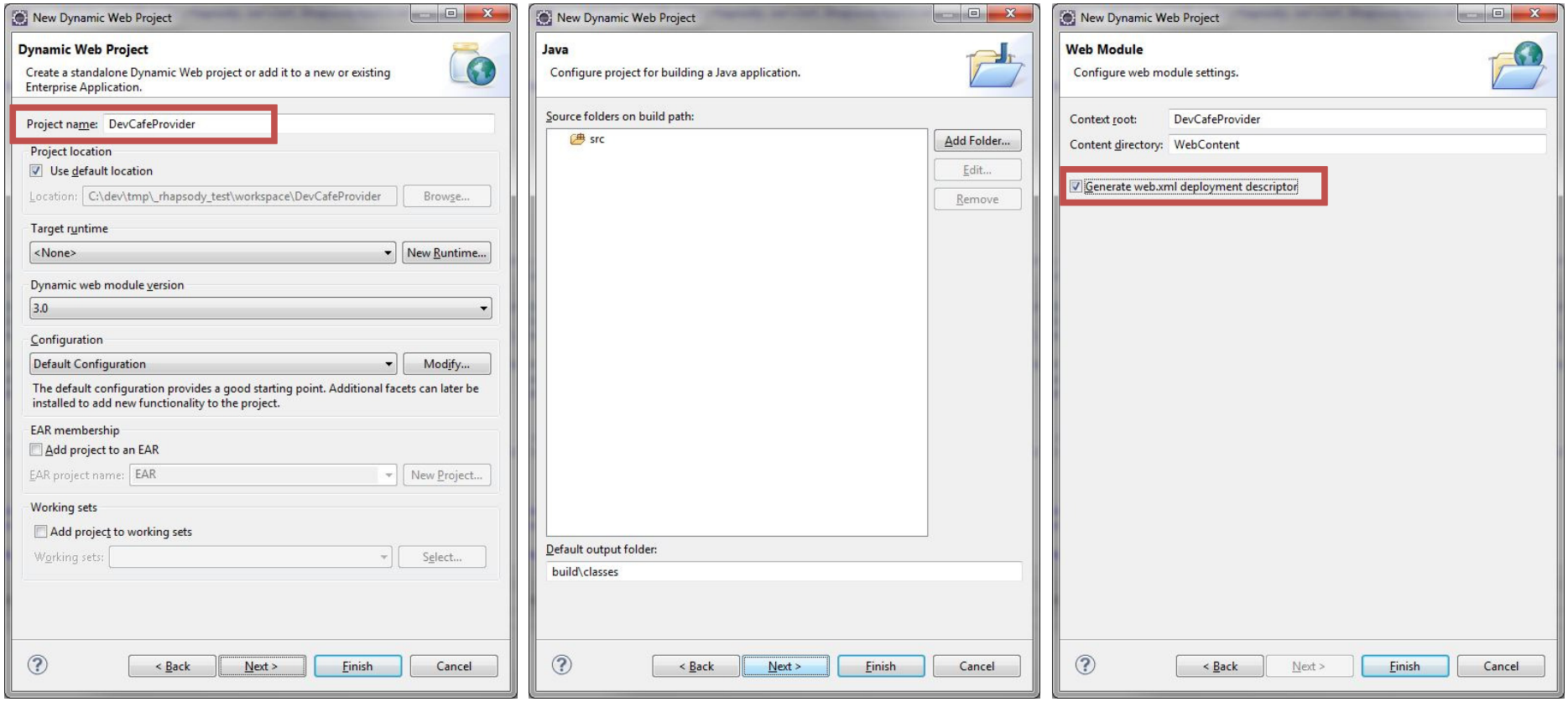
```
@OslcSchema({
        @OslcNamespaceDefinition(prefix = OslcConstants.DCTERMS_NAMESPACE_PREFIX,
    namespaceURI = OslcConstants.DCTERMS_NAMESPACE),
        @OslcNamespaceDefinition(prefix = OslcConstants.OSLC_CORE_NAMESPACE_PREFIX,
    namespaceURI = OslcConstants.OSLC_CORE_NAMESPACE),
        @OslcNamespaceDefinition(prefix = OslcConstants.OSLC_DATA_NAMESPACE_PREFIX,
    namespaceURI = OslcConstants.OSLC_DATA_NAMESPACE),
        @OslcNamespaceDefinition(prefix = OslcConstants.RDF_NAMESPACE_PREFIX,
    namespaceURI = OslcConstants.RDF_NAMESPACE),
        @OslcNamespaceDefinition(prefix = OslcConstants.RDFS_NAMESPACE_PREFIX,
    namespaceURI = OslcConstants.RDFS_NAMESPACE),
        @OslcNamespaceDefinition(prefix = Constants.FOAF_NAMESPACE_PREFIX, namespaceURI =
    Constants.FOAF_NAMESPACE),
        @OslcNamespaceDefinition(prefix = Constants.ARCHITECTURE_MANAGEMENT_PREFIX,
    namespaceURI = Constants.ARCHITECTURE_MANAGEMENT_NAMESPACE),
… })
package at.vif.oslc.rhapsody.resources;
```

Provider Logic Implementation

How to implement a Provider

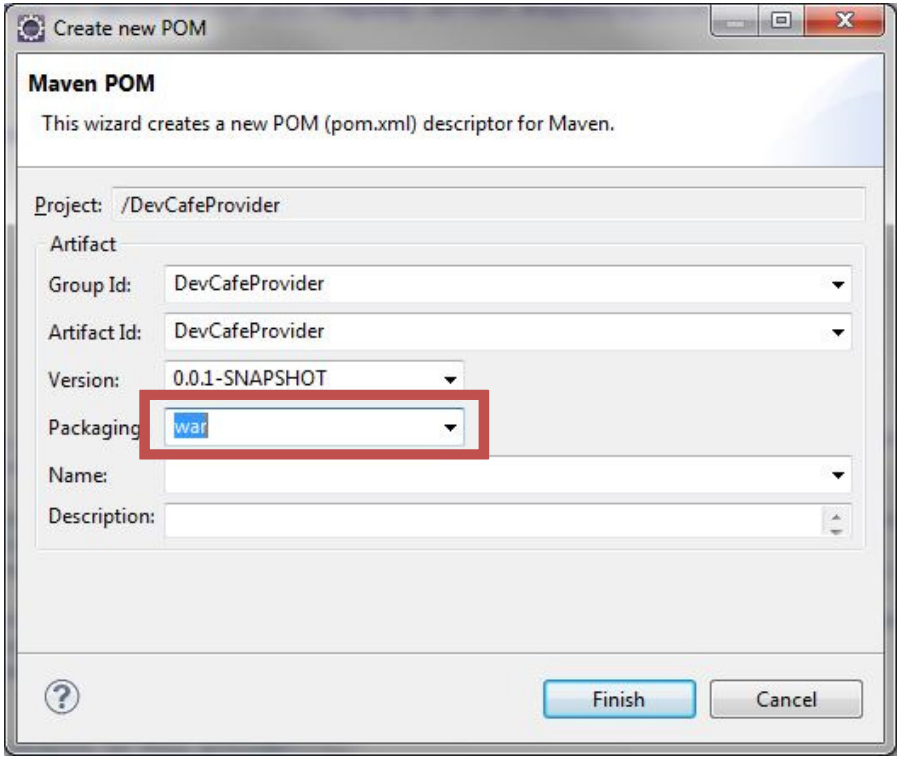
- ✓ Implement Resource Class
- Implement Provider Logic
  - Setup the JAVA EE Web Application
  - Implement Service for each resource type
    - Provision of Resources – HTTP GET
    - Altering Resources
      - HTTP PUT, POST, DELETE
    - Route requests to underlying logic
  - Other Configurations
- Running the provider
- For Testing
  - Implement Junit Tests
  - Implement Consumer logic
    - Retrieve
    - Update
    - (Create, Delete)

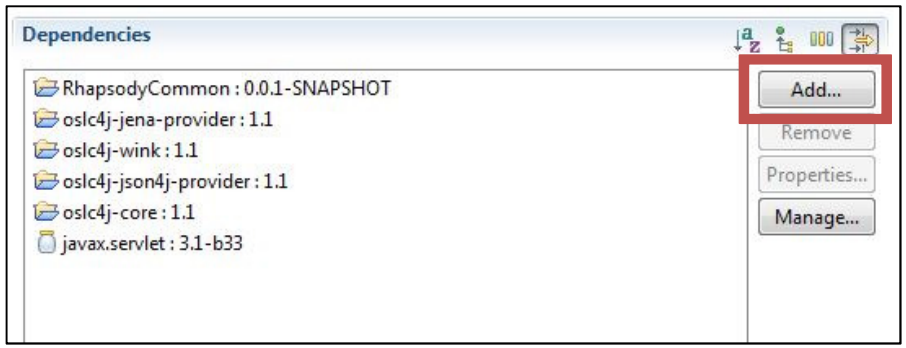
- File > New > Project
- Dynamic Web Project



Setup the JAVA EE Web Application

- Configure > Convert to Maven Project
- Generate war-package
- Open pom.xml
- Select tab dependencies
- Add dependencies
  - oslc4j-core
  - oslc4j-jena-provider
  - oslc4j-wink
  - oslc4j-json4j-provider
  - javax.servlet
    - org.glassfish
    - 3.1-b33
  - Own …Common project





pom.xml - Setup JAVA EE Web Application

- Check project description
- Check build configuration
- check source directory append ${basedir}
- Add missing plugins
  - maven-compiler-plugin

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.
    0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>RhapsodyProvider</groupId>
  <artifactId>RhapsodyProvider</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <build>
    <sourceDirectory>${basedir}/src/</sourceDirectory>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.3.2</version>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
        </configuration>
      </plugin>
```

pom.xml - Setup JAVA EE Web Application

- Add missing plugins
  - maven-war-plugin
  - maven-eclipse-plugin
- Check dependencies
  - see next slide

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <version>2.1.1</version>
  <configuration>
<warSourceDirectory>WebContent</warSourceDirectory>
    <webXml>WebContent\WEB-INF\web.xml</webXml>
    <webResources>
      <resource>
        <directory>src</directory>
        <include>log4j.properties</include>
        <targetPath>WEB-INF/classes</targetPath>
      </resource>
    </webResources>
  </configuration>
</plugin>
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-eclipse-plugin</artifactId>
  <version>2.8</version>
  <configuration>
    <wptversion>2.0</wptversion>
  </configuration>
</plugin>
</build>
```

pom.xml - Setup JAVA EE Web Application

```
<dependencies>
    <dependency>
      <groupId>RhapsodyCommon</groupId>
      <artifactId>RhapsodyCommon</artifactId>
      <version>0.0.1-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>org.eclipse.lyo.oslc4j.core</groupId>
      <artifactId>oslc4j-jena-provider</artifactId>
      <version>1.1</version>
    </dependency>
    <dependency>
      <groupId>org.eclipse.lyo.oslc4j.core</groupId>
      <artifactId>oslc4j-wink</artifactId>
      <version>1.1</version>
    </dependency>
    <dependency>
      <groupId>org.eclipse.lyo.oslc4j.core</groupId>
      <artifactId>oslc4j-json4j-provider</artifactId>
      <version>1.1</version>
    </dependency>
    <dependency>
      <groupId>org.eclipse.lyo.oslc4j.core</groupId>
      <artifactId>oslc4j-core</artifactId>
      <version>1.1</version>
    </dependency>
    <dependency>
      <groupId>org.glassfish</groupId>
      <artifactId>javax.servlet</artifactId>
      <version>3.1-b33</version>
    </dependency>
  </dependencies>
</project>
```

How to implement a Provider

- ✓ Implement Resource Class
- Implement Provider Logic
  - ✓ Setup the JAVA EE Web Application
  - Implement Service for each resource type
    - Provision of Resources – HTTP GET
    - Altering Resources
      - HTTP PUT, POST, DELETE
    - Route requests to underlying logic
  - Other Configurations
- Running the provider
- For Testing
  - Implement Junit Tests
  - Implement Consumer logic
    - Retrieve
    - Update
    - (Create, Delete)

Resource Service Implementation

- Implement Service Class
  - One for each resource type
- Implement methods for
  - Provision of Resources
    - RDF/XML, XML, JSON
    - HTML
    - HTTP GET
      - For all Resources
      - For one Resource
  - Altering Resources
    - HTTP PUT, POST, DELETE
  - Route requests to underlying logic
- Implement ServiceProvider Factory and Singelton
- Implement Servlet Listener (`implements ServletContextListener`)
- JAX-RS Servlet
  - Implement application class
  - Activate JAX-RS project Facet for application class
- Set all needed values in web.xml

GENERAL - Resource Service Implementation


```
package at.vif.oslc.rhapsody.servlet.services;

import java.net.URI;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

import org.eclipse.lyo.oslc4j.core.annotation.OslcCreationFactory;
import org.eclipse.lyo.oslc4j.core.annotation.OslcDialog;
import org.eclipse.lyo.oslc4j.core.annotation.OslcDialogs;
import org.eclipse.lyo.oslc4j.core.annotation.OslcQueryCapability;
import org.eclipse.lyo.oslc4j.core.annotation.OslcService;
import org.eclipse.lyo.oslc4j.core.model.OslcConstants;
import org.eclipse.lyo.oslc4j.core.model.OslcMediaType;

import at.vif.oslc.rhapsody.Constants;
import at.vif.oslc.rhapsody.dataaccess.Persistence;
import at.vif.oslc.rhapsody.resources.Resource;
```

GENERAL - Resource Service Implementation


```
@OslcService(Constants.ARCHITECTURE_MANAGEMENT_DOMAIN)
@Path("resources")
public class ResourceService {

// Use ETags for recognition of resource modifications

// http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.19


private static void setETagHeader(final String eTagFromResource,
   final HttpServletResponse httpServletResponse) {
        httpServletResponse.setHeader("ETag", eTagFromResource);
    }


private static String getETagFromResource(final Resource resource) {
        return Long.toString(resource.getModified().getTime());
    }
```

GET all - Resource Service Implementation


```
@OslcQueryCapability(
     title = "Resource Query Capability"
   , label = "Resource Catalog Query"
   , resourceShape = OslcConstants.PATH_RESOURCE_SHAPES + "/"
    + Constants.PATH_RESOURCE
   , resourceTypes = { Constants.TYPE_RESOURCE }
   , usages = { OslcConstants.OSLC_USAGE_DEFAULT }
)
@GET
@Produces({ OslcMediaType.APPLICATION_RDF_XML,
   OslcMediaType.APPLICATION_XML,
   OslcMediaType.APPLICATION_JSON })
public Resource[] getResources(@QueryParam("oslc.where")
   final String where) {
…
     }
```

GET all - Resource Service Implementation

```
@OslcDialogs({
                @OslcDialog(title = "Resources Selection Dialog"
                    , label = "Resources Selection Dialog"
                    , uri = ""
                    , hintWidth = "1000px"
                    , hintHeight = "600px"
                    , resourceTypes = { Constants.TYPE_RESOURCE }
                    , usages = { OslcConstants.OSLC_USAGE_DEFAULT }),
                @OslcDialog(title = "Resources List Dialog"
                    , label = "Resources List Dialog"
                    , uri = "UI/resources/list.jsp"
                    , hintWidth = "1000px"
                    , hintHeight = "600px"
                    , resourceTypes = { Constants.TYPE_RESOURCE }
                    , usages = { Constants.USAGE_LIST })
            })
…
public Resource[] getResources(@QueryParam("oslc.where") final String where) {
… }
```

GET one - Resource Service Implementation


```
@GET
@Path("{resourceId}")
@Produces({ OslcMediaType.APPLICATION_RDF_XML,
   OslcMediaType.APPLICATION_XML,
   OslcMediaType.APPLICATION_JSON })
public Resource getResource(@Context final
   HttpServletResponse httpServletResponse,
   @PathParam("resourceId") final String
   resourceId) {
      …
}
```

POST - Resource Service Implementation

```
@OslcCreationFactory(title = "Resource Creation Factory"
            , label = "Resource Creation"
            , resourceShapes = { OslcConstants.PATH_RESOURCE_SHAPES +
   "/" + Constants.PATH_RESOURCE }
            , resourceTypes = { Constants.TYPE_RESOURCE }
            , usages = { OslcConstants.OSLC_USAGE_DEFAULT })
@POST
@Consumes({ OslcMediaType.APPLICATION_RDF_XML,
   OslcMediaType.APPLICATION_XML, OslcMediaType.APPLICATION_JSON })
@Produces({ OslcMediaType.APPLICATION_RDF_XML,
   OslcMediaType.APPLICATION_XML, OslcMediaType.APPLICATION_JSON })
public Response addResource(@Context final HttpServletRequest
   httpServletRequest, final Resource resource) {
        …
    }
```

PUT - Resource Service Implementation


```
@PUT
@Consumes({ OslcMediaType.APPLICATION_RDF_XML,
  OslcMediaType.APPLICATION_XML,
  OslcMediaType.APPLICATION_JSON })
@Path("{resourceId}")
public Response
  updateResource(@PathParam("resourceId") final
  String resourceId, final Resource resource) {

…
}
```

DELETE - Resource Service Implementation


```
@DELETE
@Path("{resourceId}")
public Response
 deleteResource(@PathParam("resourceId") final
 String resourceId) {
…
}
```

Resource Service Implementation

- ✓ Implement Service Class
  - One for each resource type
- ✓ Implement methods for
  - Provision of Resources
    - RDF/XML, XML, JSON
    - **HTML**
    - HTTP GET
    - For all Resources
    - For one Resource
  - Altering Resources
    - HTTP PUT, POST, DELETE
  - Route requests to underlying logic
- Implement ServiceProvider Factory and Singleton
- Implement Servlet Listener (`implements ServletContextListener`)
- JAX-RS Servlet
  - Implement application class
  - Activate JAX-RS project Facet for application class
- Set all needed values in web.xml

Service Provider Factory Implementation


```
import java.net.URI;
import java.net.URISyntaxException;

import org.eclipse.lyo.oslc4j.client.ServiceProviderRegistryURIs;
import org.eclipse.lyo.oslc4j.core.exception.OslcCoreApplicationException;
import org.eclipse.lyo.oslc4j.core.model.OslcConstants;
import org.eclipse.lyo.oslc4j.core.model.PrefixDefinition;
import org.eclipse.lyo.oslc4j.core.model.Publisher;
import org.eclipse.lyo.oslc4j.core.model.ServiceProvider;

import at.vif.oslc.devcafe.rhapsody.servlet.services.ResourceService;
import at.vif.oslc.rhapsody.Constants;

public class ServiceProviderFactory {
  …
}
```

Service Provider Factory Implementation

- Create a ServiceProvider using ServiceProviderFactory
- Service description document accessible under passed URI
- Lists the services defined by RESOURCE_CLASSES

```
public class ServiceProviderFactory {
  private static final Class<?>[] RESOURCE_CLASSES = { ResourceService.class };

  private ServiceProviderFactory() {
    super();
  }

  public static ServiceProvider createServiceProvider(final String baseURI)
           throws OslcCoreApplicationException, URISyntaxException {
 final ServiceProvider serviceProvider = org.eclipse.lyo.oslc4j.core.model.ServiceProviderFactory
     .createServiceProvider(
         baseURI,
         ServiceProviderRegistryURIs.getUIURI(),
         "OSLC ViF Rhapsody Service Provider",
         "OSLC ViF Rhapsody Service Provider as contribution for MBAT Use Case 7 - Hybrid Power Train"
         + " Control Unit",
         new Publisher("ViF", "urn:oslc:ServiceProvider“)
         , RESOURCE_CLASSES);
```

Service Provider Factory Implementation


```
        URI[] detailsURIs = { new URI(baseURI) };
        serviceProvider.setDetails(detailsURIs);

        final PrefixDefinition[] prefixDefinitions = {
                    new PrefixDefinition(OslcConstants.DCTERMS_NAMESPACE_PREFIX, new URI(
                            OslcConstants.DCTERMS_NAMESPACE)),
                    new PrefixDefinition(OslcConstants.OSLC_CORE_NAMESPACE_PREFIX, new URI(
                            OslcConstants.OSLC_CORE_NAMESPACE)),
                    new PrefixDefinition(OslcConstants.OSLC_DATA_NAMESPACE_PREFIX, new URI(
                            OslcConstants.OSLC_DATA_NAMESPACE)),
                    new PrefixDefinition(OslcConstants.RDF_NAMESPACE_PREFIX, new URI(
                            OslcConstants.RDF_NAMESPACE)),
                    new PrefixDefinition(OslcConstants.RDFS_NAMESPACE_PREFIX, new URI(
                            OslcConstants.RDFS_NAMESPACE)),
                    new PrefixDefinition(Constants.FOAF_NAMESPACE_PREFIX, new URI(Constants.FOAF_NAMESPACE)),
                    new PrefixDefinition(Constants.ARCHITECTURE_MANAGEMENT_PREFIX, new URI(
                            Constants.ARCHITECTURE_MANAGEMENT_NAMESPACE)),
                    new PrefixDefinition(Constants.QUALITY_MANAGEMENT_PREFIX, new URI(
                            Constants.QUALITY_MANAGEMENT_NAMESPACE)),
                    new PrefixDefinition(Constants.VIF_QM_NAMESPACE_PREFIX, new URI(Constants.VIF_QM_NAMESPACE))
        };

        serviceProvider.setPrefixDefinitions(prefixDefinitions);

        return serviceProvider;
    }
}
```

Service Provider Singleton Implementation

- Holds a reference to the created Service Provider

```
import java.net.URI;
import org.eclipse.lyo.oslc4j.core.model.ServiceProvider;

public class ServiceProviderSingelton {
    private static ServiceProvider serviceProviderSingelton;
    private static URI serviceProviderURISingelton;

    private ServiceProviderSingelton() {
        super();
    }

    public static synchronized ServiceProvider getServiceProvider() {
        return serviceProviderSingelton;
    }

    public static void setServiceProvider(final ServiceProvider serviceProvider) {
        ServiceProviderSingelton.serviceProviderSingelton = serviceProvider;
    }

    public static synchronized URI getServiceProviderURI() {
        return serviceProviderURISingelton;
    }

    public static void setServiceProviderURI(final URI serviceProviderURI) {
        ServiceProviderSingelton.serviceProviderURISingelton = serviceProviderURI;
    }
}
```

Resource Service Implementation

- ✓ Implement Service Class
  - One for each resource type
- ✓ Implement methods for
  - Provision of Resources
    - RDF/XML, XML, JSON
    - HTML
    - HTTP GET
    - For all Resources
    - For one Resource
  - Altering Resources
    - HTTP PUT, POST, DELETE
  - Route requests to underlying logic
- ✓ Implement ServiceProvider Factory and Singleton
- Implement Servlet Listener (`implements ServletContextListener`)
- JAX-RS Servlet
  - Implement application class
  - Activate JAX-RS project Facet for application class
- Set all needed values in web.xml

Servlet Listener Implementation

- Implementation of the Interface ServletContextListener
- Receives notification events about ServletContext lifecycle changes
- **contextInitialized(ServletContextEvent)** is invoked when the web application initialization process is starting
- in reverse order **contextDestroyed(ServletContextEvent)** is invoked when the **ServletContext** is about to be shut down.

- 2 Phases to implement
- Initialization
  - A ServiceProvider
    - Is **created** using ServiceProviderFactory.createServiceProvider(String)
    - **registered** in the catalog using ServiceProviderRegistryClient.registerServiceProvider(ServiceProvider).
  - Reference to the created service provider object and the URL of the catalog entry are stored in ServiceProviderSingelton.
  - Initialize persistence logic

- Shut down
  - The ServiceProvider stored in ServiceProviderSingelton is deregistered from the catalog
  - using ServiceProviderRegistryClient.deregisterServiceProvider(URI)
  - Shutdown persistence logic

Servlet Listener Implementation


```
import java.net.InetAddress;
import java.net.URI;
import java.net.UnknownHostException;
import java.util.Properties;
import java.util.Timer;
import java.util.TimerTask;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import org.eclipse.lyo.oslc4j.client.ServiceProviderRegistryClient;
import org.eclipse.lyo.oslc4j.core.model.ServiceProvider;
import org.eclipse.lyo.oslc4j.provider.jena.JenaProvidersRegistry;
```

Servlet Listener Implementation


```
public class ServletListener implements ServletContextListener {

    private static final String PROPERTY_SCHEME = ServletListener.class.getPackage().getName() + ".scheme";

    private static final String PROPERTY_PORT = ServletListener.class.getPackage().getName() + ".port";

    private static final Logger LOGGER = Logger.getLogger(ServletListener.class.getName());

    private static final String HOST = getHost();

    private ServiceProviderRegistryClient client;

    public ServletListener() {
        super();
    }

private static String getHost() {
        try {
            return InetAddress.getLocalHost().getCanonicalHostName();
        } catch (final UnknownHostException exception) {
            return "localhost";
        }
    }
```

Servlet Listener Implementation

- Generates the URI that is used to access the service provider that is initialized.
- servletContextEvent comprises ServletContext which is used to determine several portions (e.g. port) of the returned URI.

```
private static String generateBasePath(final ServletContextEvent servletContextEvent) {
        final ServletContext servletContext = servletContextEvent.getServletContext();

        String scheme = System.getProperty(PROPERTY_SCHEME);
        if (scheme == null) {
            scheme = servletContext.getInitParameter(PROPERTY_SCHEME);
        }

        String port = System.getProperty(PROPERTY_PORT);
        if (port == null) {
            port = servletContext.getInitParameter(PROPERTY_PORT);
        }

        System.out.println(scheme + "://" + HOST + ":" + port + servletContext.getContextPath());

        return scheme + "://" + HOST + ":" + port + servletContext.getContextPath();
    }
```

Servlet Listener Implementation

- 2 Phases to implement
- Initialization
  - A ServiceProvider
    - Is **created** using ServiceProviderFactory.createServiceProvider(String)
    - **registered** in the catalog using ServiceProviderRegistryClient.registerServiceProvider(ServiceProvider).
  - Reference to the created service provider object and the URL of the catalog entry are stored in ServiceProviderSingelton.
  - Initialize persistence logic
- Shut down
  - The ServiceProvider stored in ServiceProviderSingelton is deregistered from the catalog
  - using ServiceProviderRegistryClient.deregisterServiceProvider(URI)
  - Shutdown persistence logic

Servlet Listener Implementation


```
@Override
    public void contextInitialized(final ServletContextEvent servletContextEvent) {
        // URI the initialized service provider can be accessed at.
        // (e.g. http://INNB00889.v2c2.at:8081/RhapsodyProvider)
        final String basePath = generateBasePath(servletContextEvent);
        final URI serviceProviderURI;

        try {
            final ServiceProvider serviceProvider = ServiceProviderFactory.createServiceProvider(basePath);

            client = new ServiceProviderRegistryClient(JenaProvidersRegistry.getProviders());

            serviceProviderURI = client.registerServiceProvider(serviceProvider);

            ServiceProviderSingelton.setServiceProviderURI(serviceProviderURI);

            ServiceProviderSingelton.setServiceProvider(serviceProvider);

            LOGGER.log(Level.INFO, "Service provider registration complete.");
        } catch (final Exception exception) {
            client = null;
            LOGGER.log(Level.SEVERE, "Unable to register with service provider catalog", exception);
            return;
        }

        // Establish connection to data backbone etc ...
    }
```

Servlet Listener Implementation

- 2 Phases to implement
- Initialization
  - A ServiceProvider
    - Is **created** using ServiceProviderFactory.createServiceProvider(String)
    - **registered** in the catalog using ServiceProviderRegistryClient.registerServiceProvider(ServiceProvider).
  - Reference to the created service provider object and the URL of the catalog entry are stored in ServiceProviderSingelton.
  - Initialize persistence logic
- Shut down
  - The ServiceProvider stored in ServiceProviderSingelton is deregistered from the catalog
  - using ServiceProviderRegistryClient.deregisterServiceProvider(URI)
  - Shutdown persistence logic

Servlet Listener Implementation


```
@Override
    public void contextDestroyed(final ServletContextEvent servletContextEvent) {
        final String basePath = generateBasePath(servletContextEvent);
        int serviceProviderPort = URI.create(basePath).getPort();

        if (client != null) {
            // Don't try to deregister if catalog is on same HOST as RhapsodyServiceProvider.

            if (!client.getClient().getUri().contains(HOST)
                    || URI.create(client.getClient().getUri()).getPort() != serviceProviderPort) {
                try {
                  client.deregisterServiceProvider(ServiceProviderSingelton.getServiceProviderURI());
                } catch (final Exception exception) {
                  LOGGER.log(Level.SEVERE, "Unable to deregister with service provider catalog", exception);
                } finally {
                  client = null;

                  ServiceProviderSingelton.setServiceProviderURI(null);
                }
            }
        }

        // Shutdown connections to data backbone etc...
    }
```

Resource Service Implementation

- ✓ Implement Service Class
  - One for each resource type
- ✓ Implement methods for
  - Provision of Resources
    - RDF/XML, XML, JSON
    - HTML
    - HTTP GET
    - For all Resources
    - For one Resource
  - Altering Resources
    - HTTP PUT, POST, DELETE
  - Route requests to underlying logic
- ✓ Implement ServiceProvider Factory and Singleton
- ✓ Implement Servlet Listener (`implements ServletContextListener`)
- JAX-RS Servlet
  - Implement application class
  - Activate JAX-RS project Facet for application class
- Set all needed values in web.xml

JAX-RS Servlet Implementation

- JAX-RS Servlet application
- Represents the service provider webapp
- Comprises the OSLC services classes
- Declaration as JAX-RS Servlet application in web.xml

JAX-RS Servlet Implementation

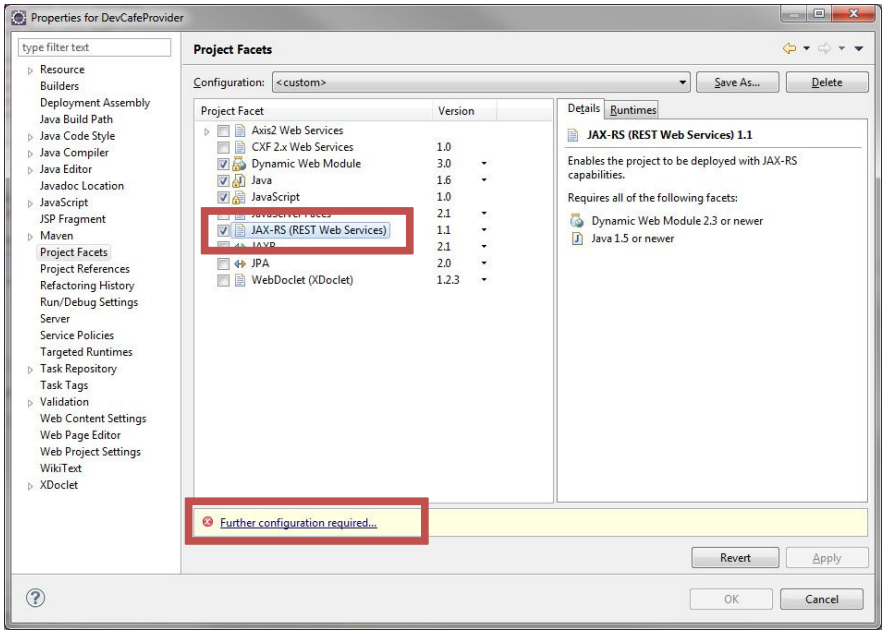

```
import java.net.URISyntaxException;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import org.eclipse.lyo.oslc4j.application.OslcWinkApplication;
import org.eclipse.lyo.oslc4j.core.exception.OslcCoreApplicationException;
import org.eclipse.lyo.oslc4j.core.model.OslcConstants;
import org.eclipse.lyo.oslc4j.provider.jena.JenaProvidersRegistry;
import org.eclipse.lyo.oslc4j.provider.json4j.Json4JProvidersRegistry;

import at.vif.oslc.rhapsody.Constants;
import at.vif.oslc.rhapsody.resources.Resource;
import at.vif.oslc.devcafe.rhapsody.servlet.services.ResourceService;
```

JAX-RS Servlet Implementation


```
public class RhapsodyApplication extends OslcWinkApplication {

  private static final Set<Class<?>> RESOURCE_CLASSES = new HashSet<Class<?>>();

  private static final Map<String, Class<?>> RESOURCE_SHAPE_PATH_TO_RESOURCE_CLASS_MAP
      = new HashMap<String, Class<?>>();

  static {
    RESOURCE_CLASSES.addAll(JenaProvidersRegistry.getProviders());
    RESOURCE_CLASSES.addAll(Json4JProvidersRegistry.getProviders());
    RESOURCE_CLASSES.add(ResourceService.class);


    RESOURCE_SHAPE_PATH_TO_RESOURCE_CLASS_MAP.put(Constants.PATH_RESOURCE, Resource.class);
  }

  public RhapsodyApplication() throws OslcCoreApplicationException, URISyntaxException {
    super(RESOURCE_CLASSES, OslcConstants.PATH_RESOURCE_SHAPES,
        RESOURCE_SHAPE_PATH_TO_RESOURCE_CLASS_MAP);
  }

}
```

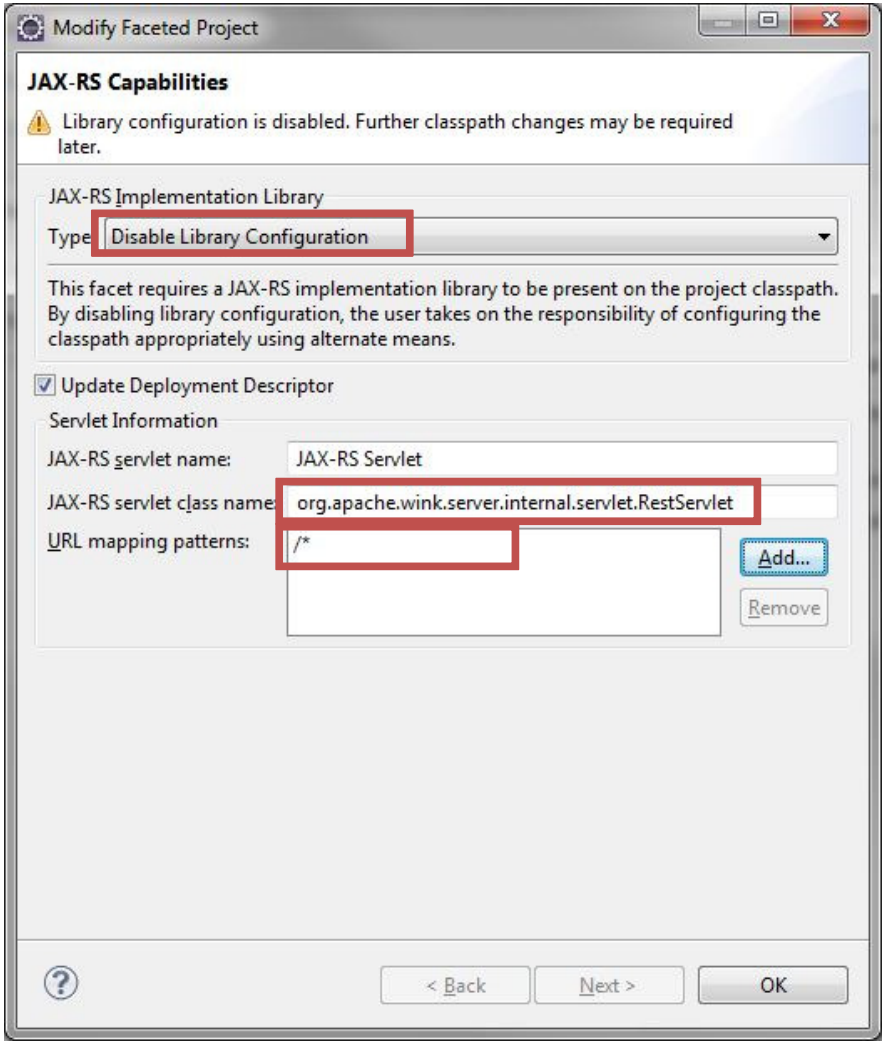
Activate Project Facet JAX-RS

- Right click provider project
- Select **properties**
- Open **Project Facets**
- Select **JAX-RS (REST Web Services)**
- Click **Further Configuration required…**



Activate Project Facet JAX-RS

- Set **Type:** Disable Library Configuration
- Enter **JAX-RS servlet class name:** org.apache.wink.server.internal.servlet.RestServlet
- **URL mapping pattern:**
  - Remove the existing one
  - Add a new one /*



Resource Service Implementation

- ✓ Implement Service Class
  - One for each resource type
- ✓ Implement methods for
  - Provision of Resources
    - RDF/XML, XML, JSON
    - HTML
    - HTTP GET
    - For all Resources
    - For one Resource
  - Altering Resources
    - HTTP PUT, POST, DELETE
  - Route requests to underlying logic
- ✓ Implement ServiceProvider Factory and Singleton
- ✓ Implement Servlet Listener (`implements ServletContextListener`)
- ✓ JAX-RS Servlet
  - Implement application class
  - Activate JAX-RS project Facet for application class
- Set all needed values in web.xml

```
<?xml version="1.0" encoding="UTF-8"?><web-app …
  <display-name>DevCafeProvider</display-name>
  <context-param>
    <description>Scheme used for URI when registering ServiceProvider.  Can be overridden by System property of the
      same name.</description>
    <param-name>at.vif.oslc.devcafe.rhapsody.servlet.scheme</param-name>
    <param-value>http</param-value>
  </context-param>
  <context-param>
    <description>Port used for URI when registering ServiceProvider.  Can be overridden by System property of the
      same name.</description>
    <param-name>at.vif.oslc.devcafe.rhapsody.servlet.port</param-name>
    <param-value>8080</param-value>
  </context-param>
  <listener>
    <description>Listener for ServletContext lifecycle changes</description>
    <listener-class>at.vif.oslc.devcafe.rhapsody.servlet.ServletListener</listener-class>
  </listener>
  <servlet>
    <description>JAX-RS Tools Generated - Do not modify</description>
    <servlet-name>JAX-RS Servlet</servlet-name>
    <servlet-class>org.apache.wink.server.internal.servlet.RestServlet</servlet-class>
    <init-param>
      <param-name>javax.ws.rs.Application</param-name>
      <param-value>at.vif.oslc.devcafe.rhapsody.servlet.RhapsodyApplication</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>JAX-RS Servlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
 <servlet>
    <servlet-name>restSdkAdmin</servlet-name>
    <servlet-class>org.apache.wink.server.internal.servlet.AdminServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>restSdkAdmin</servlet-name>
    <url-pattern>/admin</url-pattern>
  </servlet-mapping>
</web-app>
```

- Set context-params readout in ServletListener
- Define the listener class
- Add initialization parameter to the JAX-RS Servlet
  - Define JAX-RS application
  - class at.vif.oslc.devcafe.rhapsody.servlet.RhapsodyApplication
- Add restSdkAdmin Servlet

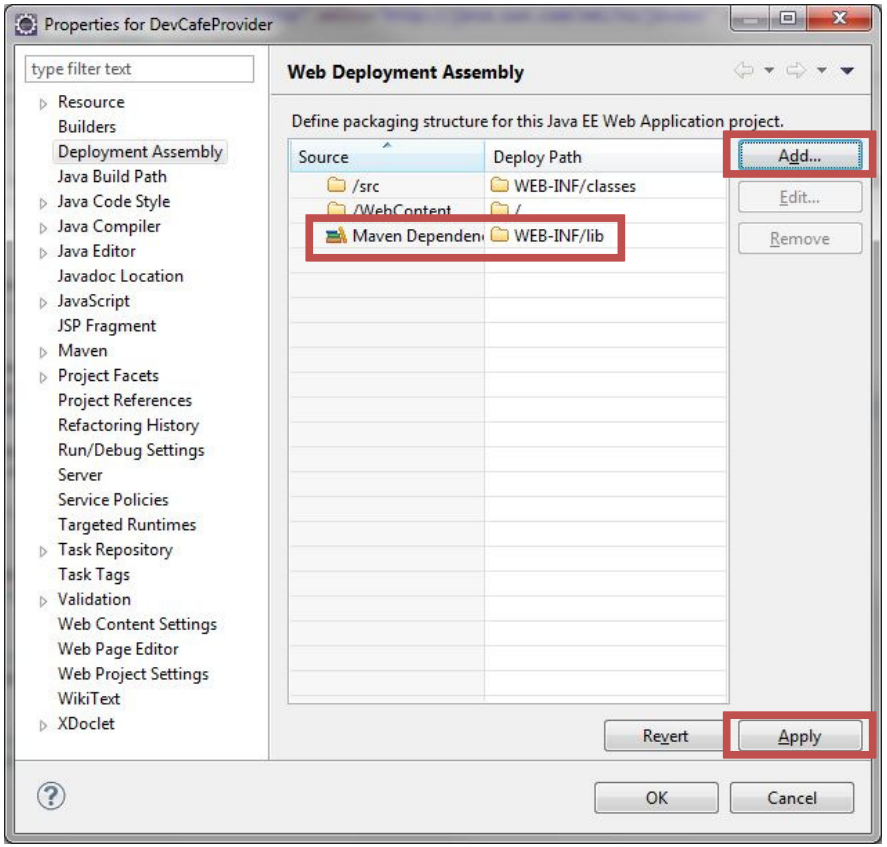
Resource Service Implementation

- ✓ Implement Service Class
  - One for each resource type
- ✓ Implement methods for
  - Provision of Resources
    - RDF/XML, XML, JSON
    - HTML
    - HTTP GET
    - For all Resources
    - For one Resource
  - Altering Resources
    - HTTP PUT, POST, DELETE
  - Route requests to underlying logic
- ✓ Implement ServiceProvider Factory and Singleton
- ✓ Implement Servlet Listener (`implements ServletContextListener`)
- ✓ JAX-RS Servlet
  - Implement application class
  - Activate JAX-RS project Facet for application class
- ✓ Set all needed values in web.xml

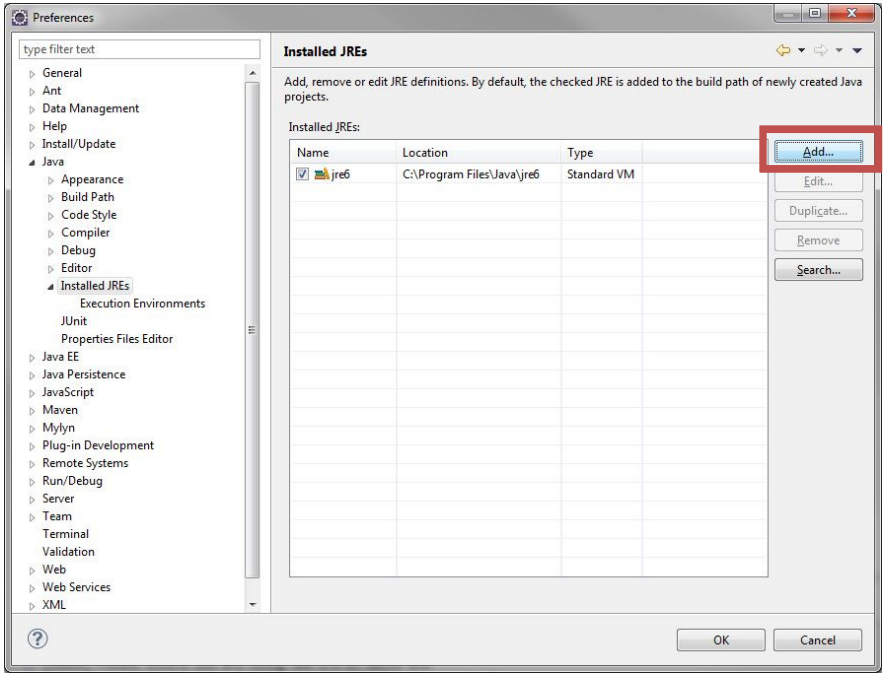
How to implement a Provider

- ✓ Implement Resource Class
- Implement Provider Logic
  - ✓ Setup the JAVA EE Web Application
  - ✓ Implement Service for each resource type
    - Provision of Resources – HTTP GET
    - Altering Resources
      - HTTP PUT, POST, DELETE
    - Route requests to underlying logic
  - Other Configurations
- Running the provider
- For Testing
  - Implement Junit Tests
  - Implement Consumer logic
    - Retrieve
    - Update
    - (Create, Delete)

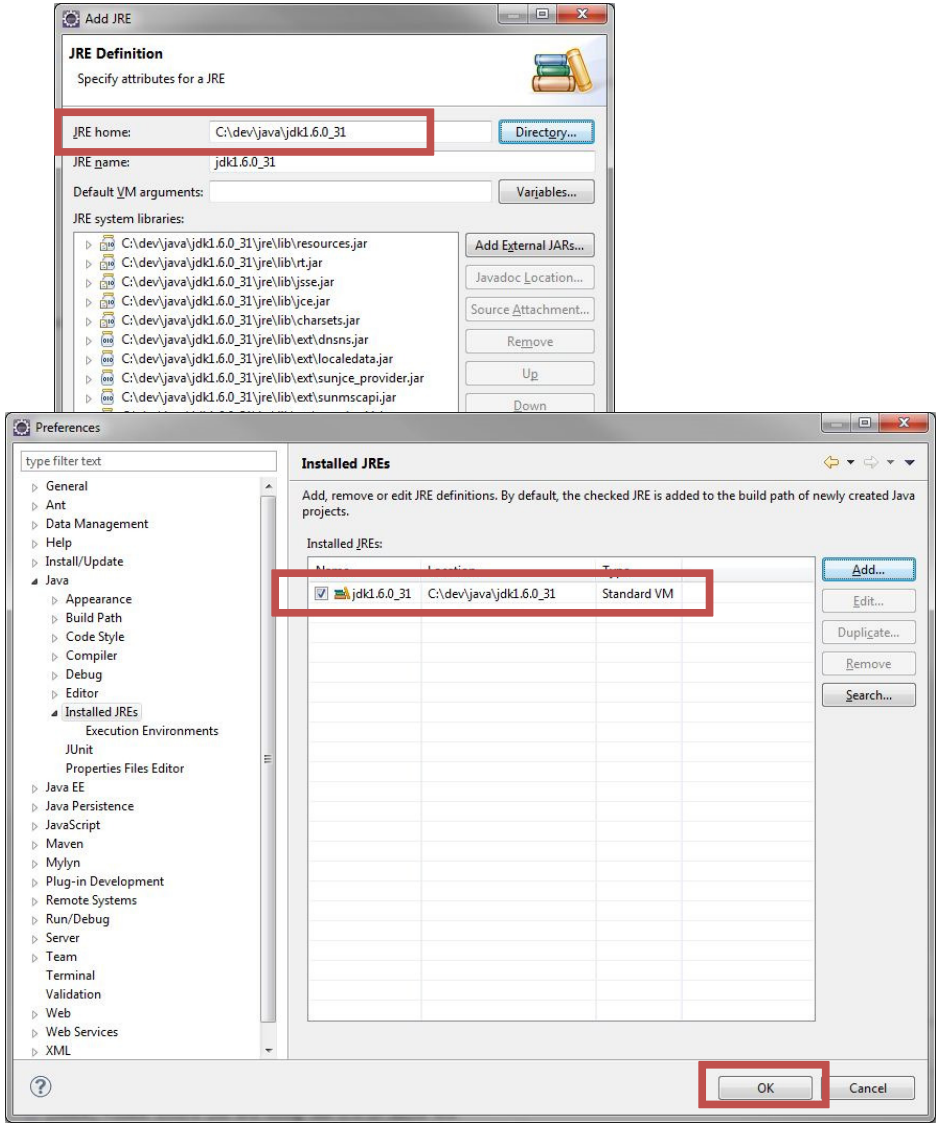
- right click provider project
- select **Properties**
- select **Deployment Assembly**
- **Add…**
  - select **Java Build Path Entries**
  - click **Next >**
  - select **Maven Dependencies**
  - click **Finish**
- click **Apply**



- Is ***JAVA_HOME*** environment variable set to a JDK 1.6?
  - e.g. JAVA_HOME=C:\dev\java\jdk1.6.0_31)
- JRE of eclipse needs to use JRE in the JDK dir
  - ***Window*** > ***Preferences***
  - ***Java*** > ***Installed JREs***
  - Click ***Add…***



- Choose ***Standard VM*** as JRE Type
- click ***Next***.
- As ***JRE home*** choose the installation dir of your JDK 1.6
  - e.g. C:\dev\java\jdk1.6.0_31)
- click ***Finish***.
- Back to the preferences screen
  - select the old JRE
  - click Remove
  - check **the newly added JDK** and
  - click ***OK***.



Running the provider

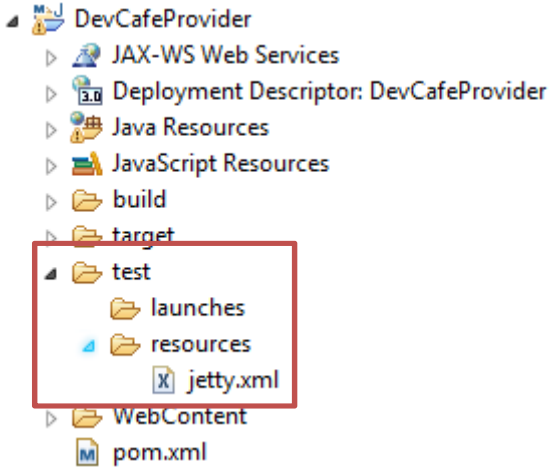
How to implement a Provider

- ✓ Implement Resource Class
- ✓ Implement Provider Logic
  - ✓ Setup the JAVA EE Web Application
  - ✓ Implement Service for each resource type
    - Provision of Resources – HTTP GET
    - Altering Resources
      - HTTP PUT, POST, DELETE
    - Route requests to underlying logic
  - ✓ Other Configurations
- Running the provider
- For Testing
  - Implement Junit Tests
  - Implement Consumer logic
    - Retrieve
    - Update
    - (Create, Delete)

- Using a tomcat in the eclipse IDE
  - http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.jst.server.ui.doc.user%2Ftopics%2Ftomcat.html
- Using a Jetty app server
  - Configure jetty
    - add needed resources
    - in pom.xml
      - include test resources
      - start server
      - start OLSC4Catalog
      - start developed provider
  - Alter ServletListener class
    - timing issues
    - provider needs to register with OSLC4JCatalog a bit delayed
  - Add a run configuration for jetty

Jetty configuration

- Add a folder **test** to the project
- Create the following folders under test
  - launches
  - resources
- in folder resources create a jetty.xml




```
<?xml version="1.0"?>
<!DOCTYPE Configure PUBLIC "-
    //Jetty//Configure//EN"
    "http://www.eclipse.org/jetty/configure.dtd
    ">

<Configure id="Server"
    class="org.eclipse.jetty.server.Server">
  <!-- see also:
    http://download.eclipse.org/jetty/stable-
    7/apidocs/ -->
</Configure>
```

Jetty configuration – pom.xml (1)

- Alter head of pom.xml
- folder with jetty.xml needs to be included as test resource

```
<project …>
  <modelVersion>4.0.0</modelVersion>
  <groupId>DevCafeProvider</groupId>
  <artifactId>DevCafeProvider
   </artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <build>
    <sourceDirectory>
   ${basedir}/src/</sourceDirectory>
    <testResources>
      <testResource>
        <directory>
   test/resources</directory>
      </testResource>
    </testResources>
```

Jetty configuration – pom.xml (2)

- in the <plugins> section
- Add the jetty-maven-plugin plugin
- Configure the URL path of the provider
  - same as web.xml <display-name>
- Add start of OSLC4J catalog
  - war file
  - ${env.OSLC4JREG} will be set in run configuration
  - configure URL path for OSLC4JCatalog

```
<plugin>
  <groupId>org.mortbay.jetty</groupId>
  <artifactId>jetty-maven-
   plugin</artifactId>
  <configuration>
    <webAppConfig>
       <contextPath>/DevCafeProvider
       </contextPath>
    </webAppConfig>
    <contextHandlers>
      <contextHandler
    implementation="org.eclipse.jetty.
    webapp.WebAppContext“>
        <war>
${env.OSLC4JREG}/target/oslc4j-
   registry-1.0.war</war>
        <contextPath>/OSLC4JRegistry
        </contextPath>
      </contextHandler>
    </contextHandlers>
```

Jetty configuration – pom.xml (3)

```
<plugin>
  <configuration>
…
    <!-- Jetty config adds logging -->
    <jettyConfig>${project.build.directory}/test-
     classes/jetty.xml</jettyConfig>
     <!-- enable hot deploy -->
    <reload>automatic</reload>
    <scanIntervalSeconds>5</scanIntervalSeconds>
    <scanTargets>
      <scanTarget>WebContent</scanTarget>
    </scanTargets>
    <systemProperties>
      <systemProperty>
        <name>config.dir</name>
        <value>${basedir}/src/test/resources</value>
      </systemProperty>
      <systemProperty>
        <name>jetty.logs</name>
        <value>${basedir}/target</value>
      </systemProperty>
      <systemProperty>
        <name>jetty.port</name>
        <value>8080</value>
      </systemProperty>
    </systemProperties>
  </configuration>
</plugin>
```

- Add folder for jetty.xml
- Add hot deploy
  - check every 5 secs
- Add properties

- Using a tomcat in the eclipse IDE
  - http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.jst.server.ui.doc.user%2Ftopics%2Ftomcat.html
- Using a Jetty app server
  - ✓ Configure jetty
    - o add needed resources
    - o in pom.xml
      - include test resources
      - start server
      - start OLSC4Catalog
      - start developed provider
  - Alter ServletListener class
    - o timing issues
    - o provider needs to register with OSLC4JCatalog a bit delayed
  - Add a run configuration for jetty

ServletListener delayed registration

- Task is a class that extends TimerTask
- Create a Timer
  - Schedule a task
  - Delay can be set when scheduling
- Trial and error for right delay
  - Needs to be long enough so OSLC4J Catalog is started
- Move everything from **contextInitialized(ServletContextEvent)** to TimerTask class
  - except generation of base path.

Creation of TimerTask


```
private class RegistrationTask extends TimerTask {
    private String basePath;

    protected RegistrationTask(final String basePath) {
        this.basePath = basePath;
    }

    @Override
    public void run() {
        final URI serviceProviderURI;

        try {
            final ServiceProvider serviceProvider = ServiceProviderFactory.createServiceProvider(basePath);

            client = new ServiceProviderRegistryClient(JenaProvidersRegistry.getProviders());
            serviceProviderURI = client.registerServiceProvider(serviceProvider);

            ServiceProviderSingelton.setServiceProviderURI(serviceProviderURI);
            ServiceProviderSingelton.setServiceProvider(serviceProvider);

            LOGGER.log(Level.INFO, "Service provider registration complete.");
        } catch (final Exception exception) {
            client = null;
            LOGGER.log(Level.SEVERE, "Unable to register with service provider catalog", exception);
            return;
        }

        // Establish connection to data backbone etc ...
    }
}
```

contextInitialized(…) - BEFORE


```
@Override
    public void contextInitialized(final ServletContextEvent servletContextEvent) {
        // URI the initialized service provider can be accessed at.
        // (e.g. http://INNB00889.v2c2.at:8081/RhapsodyProvider)
        final String basePath = generateBasePath(servletContextEvent);
        final URI serviceProviderURI;

        try {
            final ServiceProvider serviceProvider = ServiceProviderFactory.createServiceProvider(basePath);

            client = new ServiceProviderRegistryClient(JenaProvidersRegistry.getProviders());

            serviceProviderURI = client.registerServiceProvider(serviceProvider);

            ServiceProviderSingelton.setServiceProviderURI(serviceProviderURI);

            ServiceProviderSingelton.setServiceProvider(serviceProvider);

            LOGGER.log(Level.INFO, "Service provider registration complete.");
        } catch (final Exception exception) {
            client = null;
            LOGGER.log(Level.SEVERE, "Unable to register with service provider catalog", exception);
            return;
        }

        // Establish connection to data backbone etc ...
    }
```

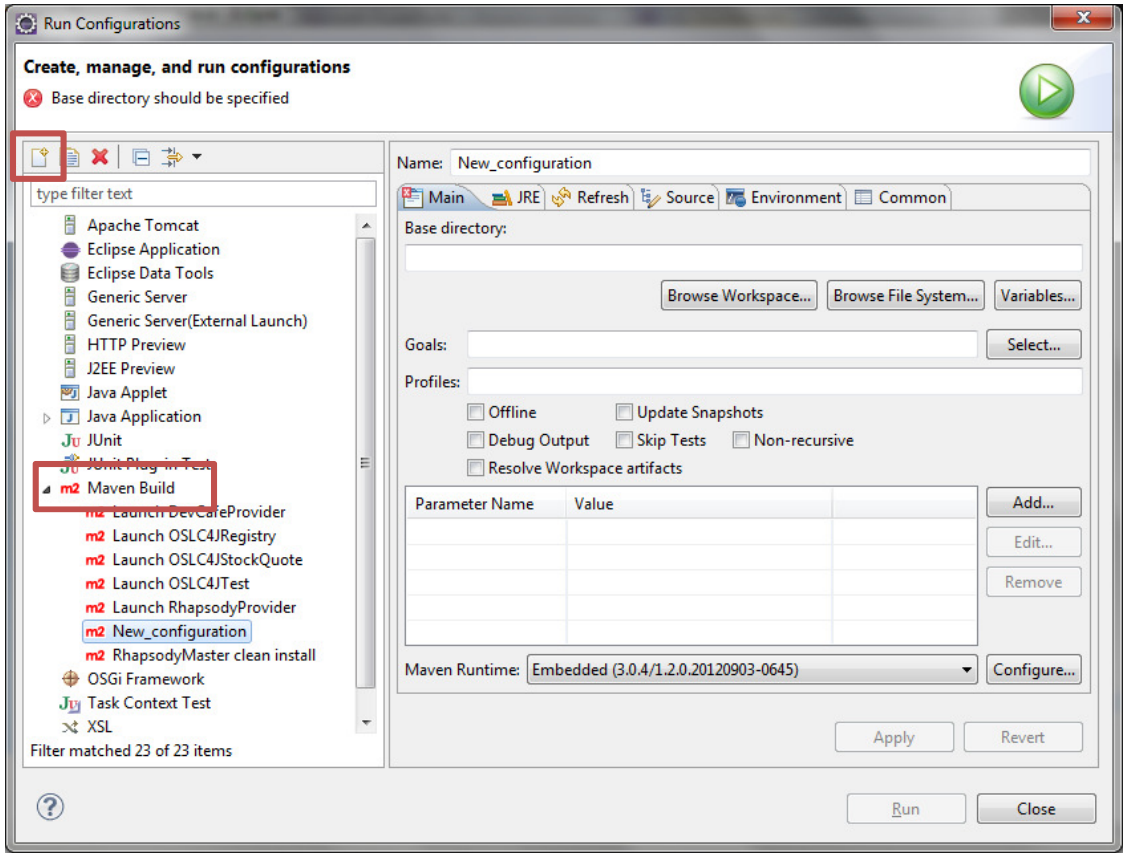
contextInitialized(…) - AFTER


```
@Override
public void contextInitialized(final ServletContextEvent servletContextEvent) {
    // URI the initialized service provider can be accessed at.
    // (e.g. http://INNB00889.v2c2.at:8081/RhapsodyProvider)
    final String basePath = generateBasePath(servletContextEvent);
    Timer timer = new Timer();
    timer.schedule(new RegistrationTask(basePath), 10000);
}
```

- Using a tomcat in the eclipse IDE
  - http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.jst.server.ui.doc.user%2Ftopics%2Ftomcat.html
- Using a Jetty app server
  - ✓ Configure jetty
    - o add needed resources
    - o in pom.xml
      - include test resources
      - start server
      - start OLSC4Catalog
      - start developed provider
  - ✓ Alter ServletListener class
    - o timing issues
    - o provider needs to register with OSLC4JCatalog a bit delayed
  - Add a run configuration for jetty

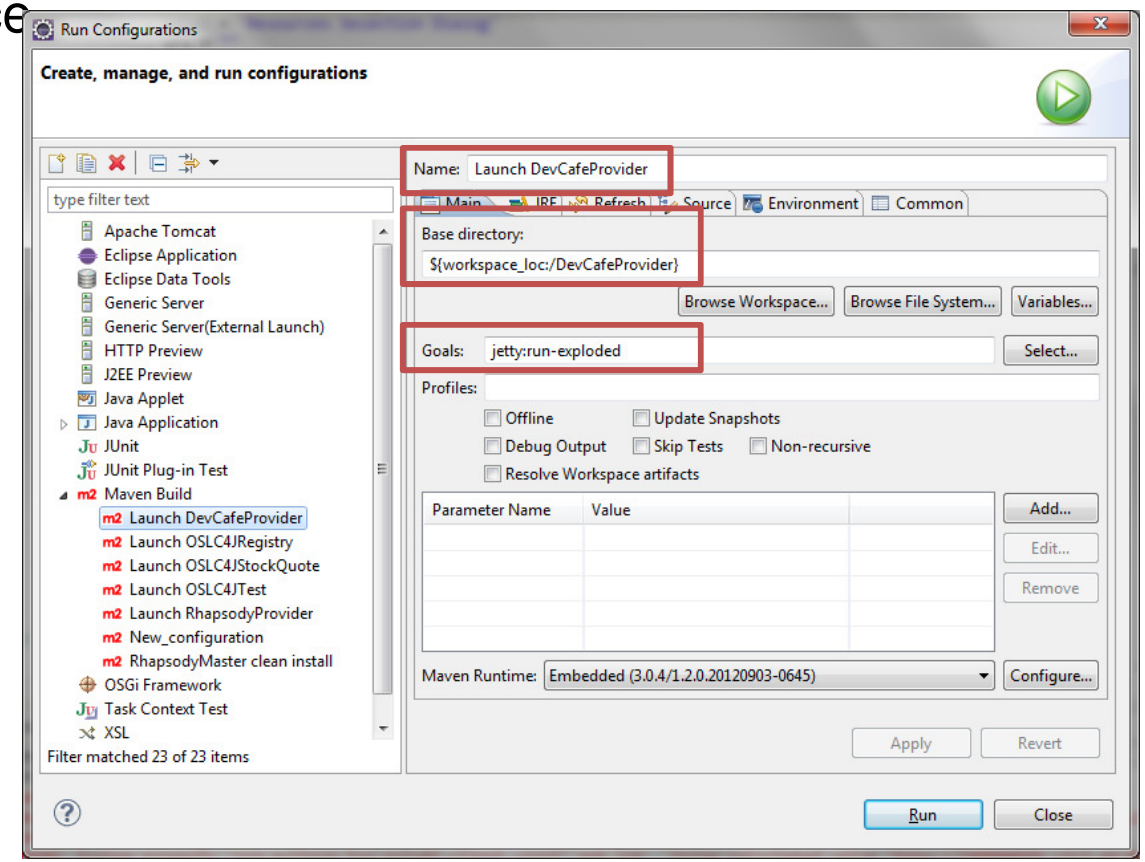
Run configuration for jetty

- Select Run > Run Configurations …
- Select Maven Build
- Create a new Configuration



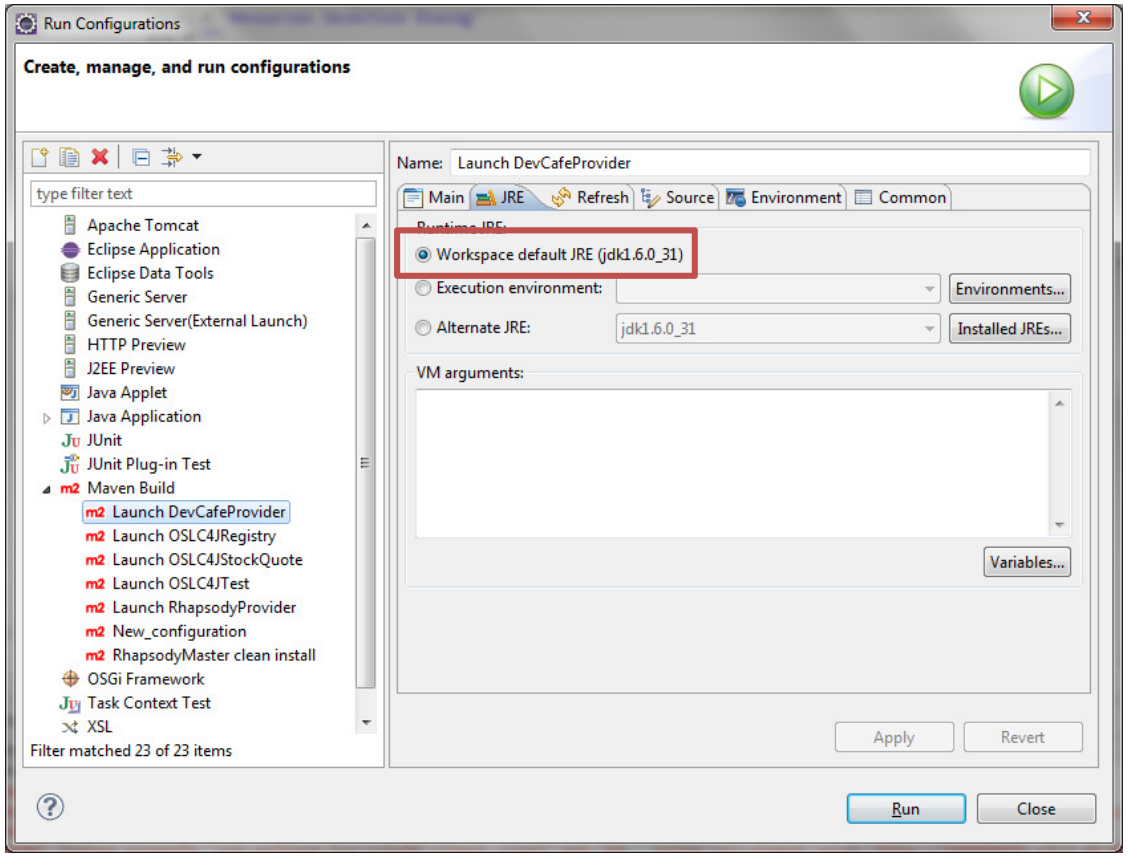
Run configuration for jetty

- Enter a **name**
- Define **Base directory**
  - path of provider project relative to workspace
- Define **Goal**
  jetty:run-exloded

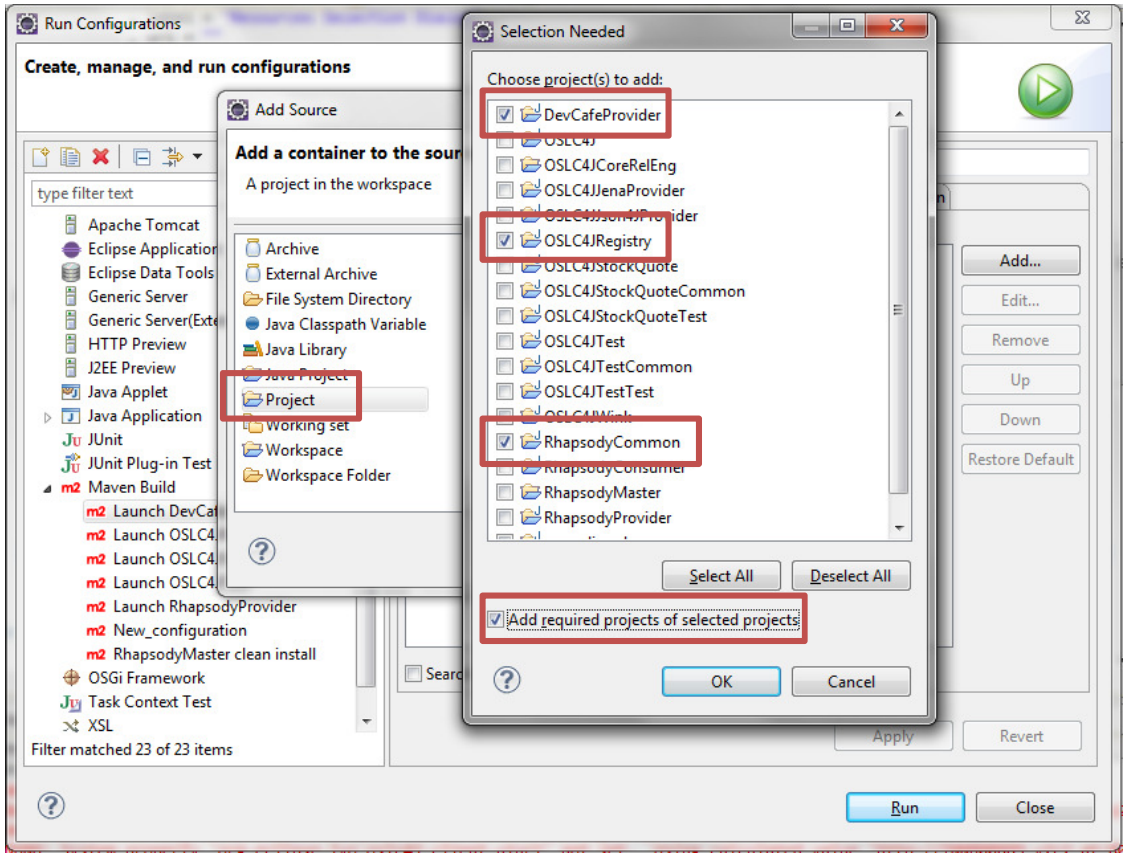


Run configuration for jetty

- Switch to tab **JRE**
- Set **Runtime JRE** to Workspace default JRE…

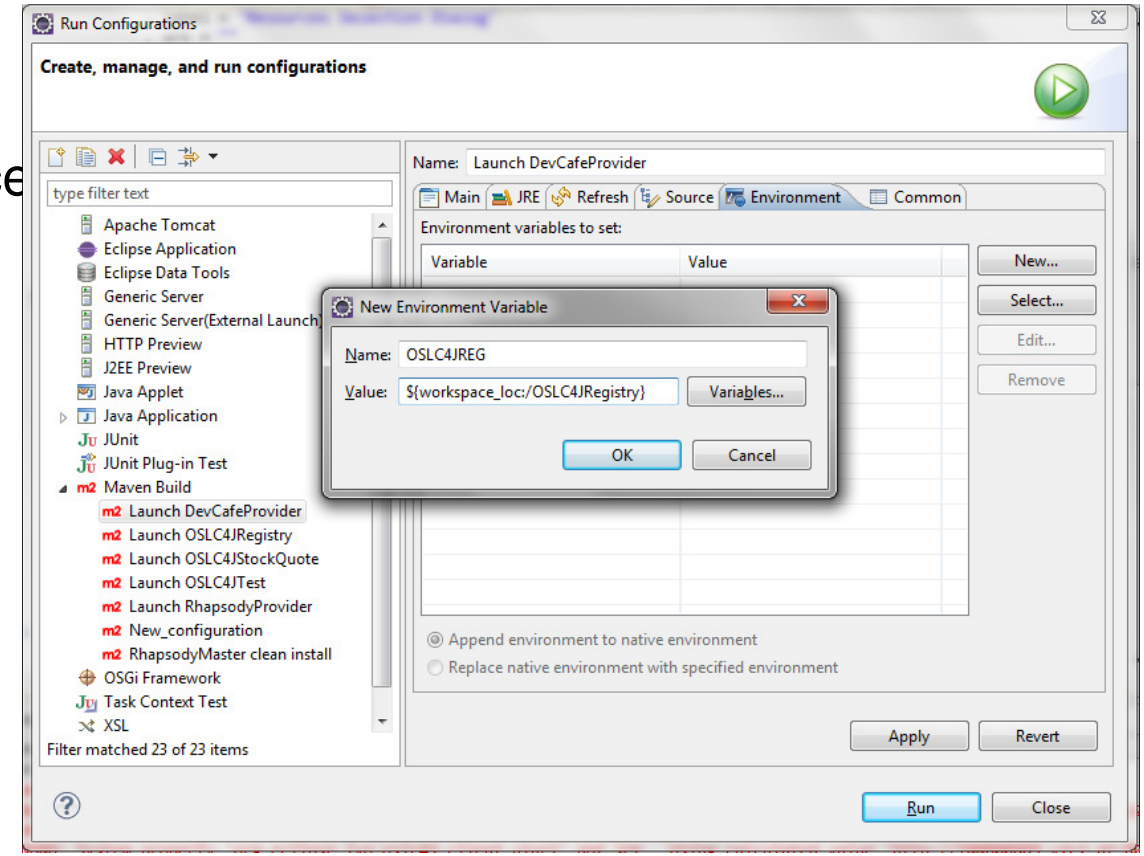


- Switch to tab **Source**
- Click Add…
- Select Project and click OK
- Check projects
  - provider project
  - OSLC4JRegistry
  - common project
- Check add required projects of selected projects
- Click OK



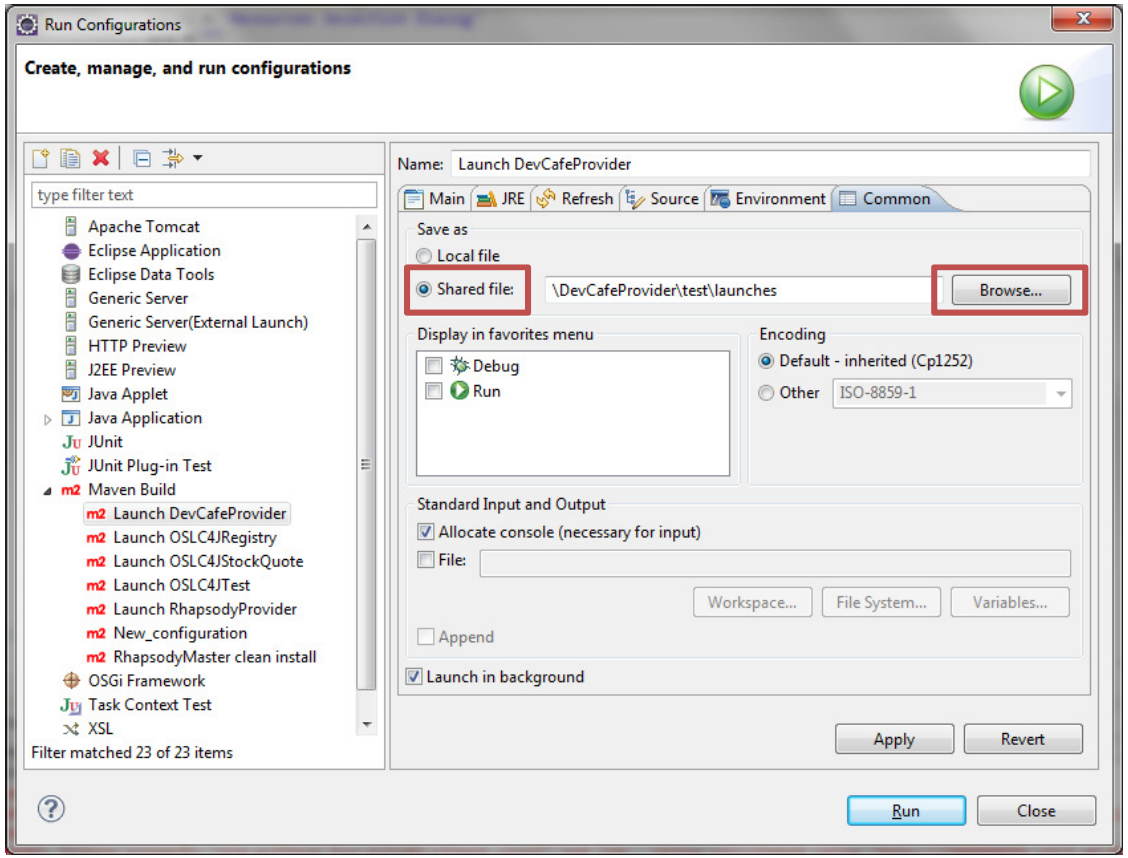
Run configuration for jetty

- Switch to tab **Environment**
- Set environment variable
  - OSLC4JREG
  - path to OSLC4J Registry project relative to workspace
- Click New…
- Enter Name OSLC4JREG
- Enter Value ${workspace_loc:/OSLC4JRegistry}
- Click OK



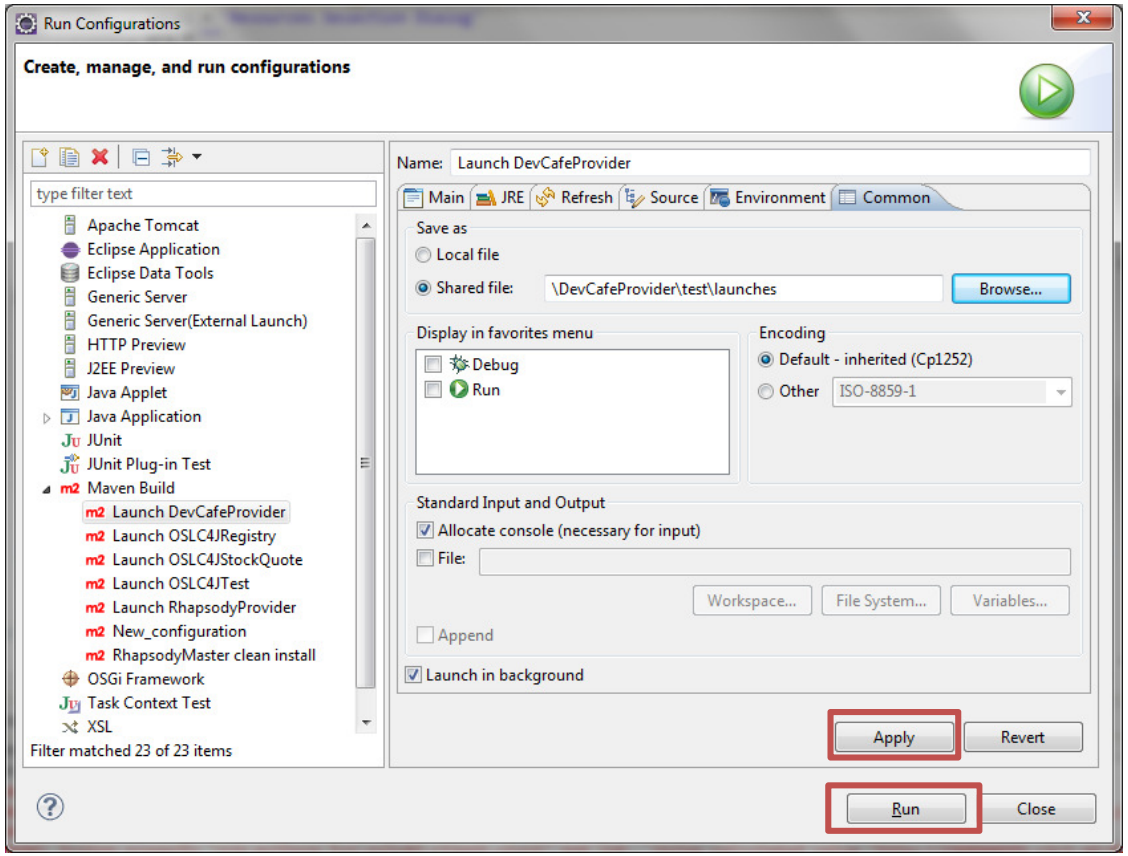
Run configuration for jetty

- Switch to Tab **Common**
- Select Shared Files
- Click Browse…
- Select previously created folder test/launches in provider project
- click OK

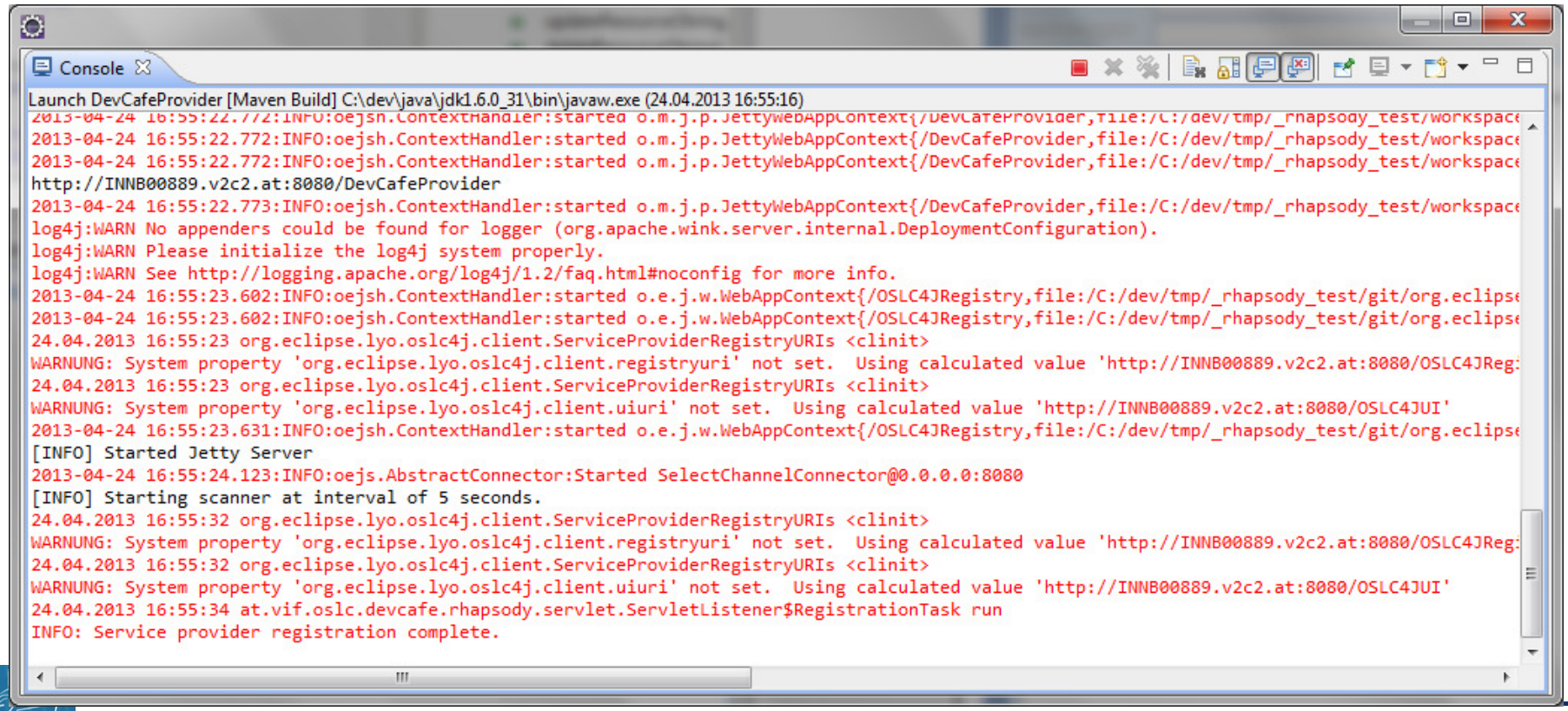


Run configuration for jetty

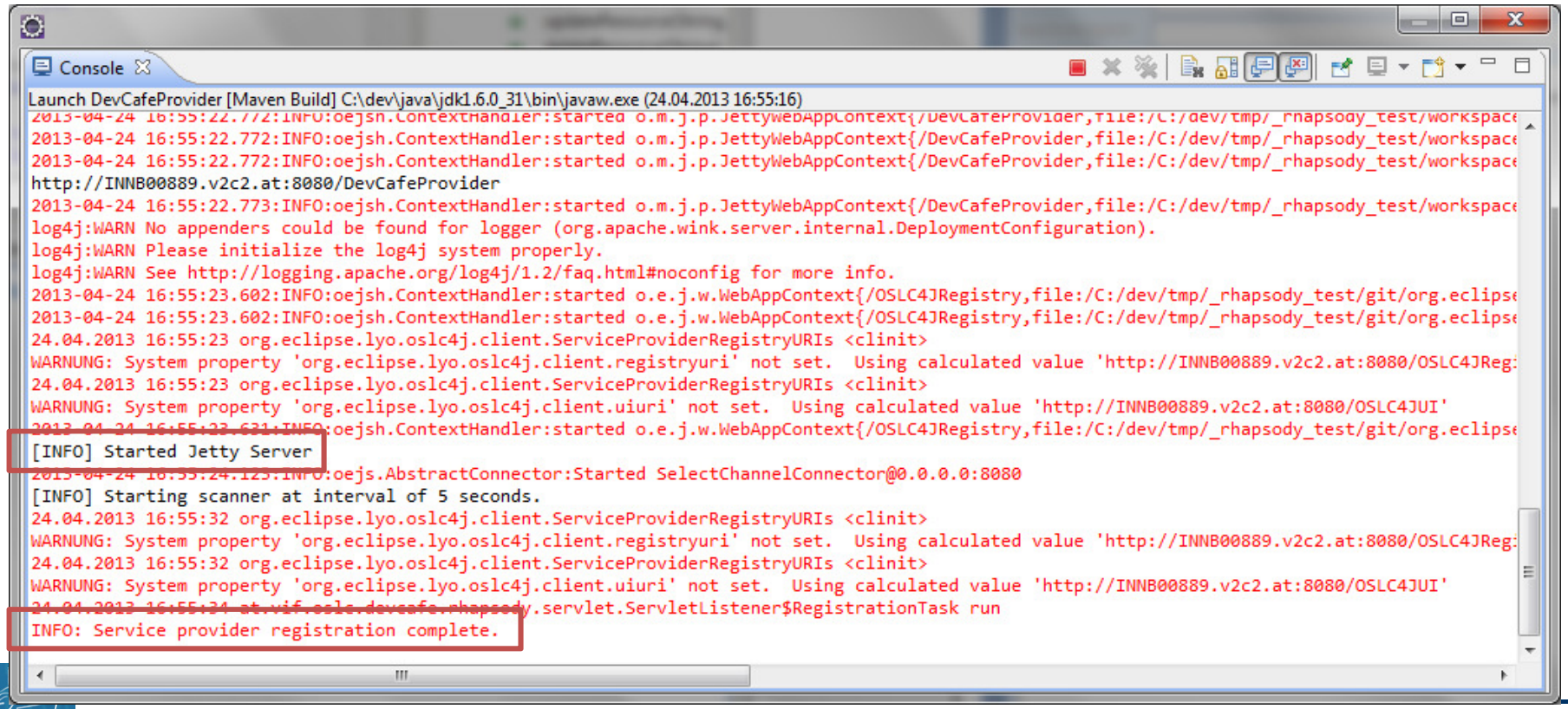
- Click Apply
- Click Run



- Goto output on Console
- When finished check
  - no java errors
  - End of output should look like depicted



- Look for
  - **[INFO] Started Jetty Server**
  - **INFO: Service provider registration complete**
- Open http://localhost:8080/DevCafeProvider/resources



- Service Provider Catalog
  http://localhost:8080/OSLC4JRegistry/catalog/
  - Dev Cafe Service Provider
    http://INNB00889.v2c2.at:8080/OSLC4JRegistry/serviceProviders/2
    - Services
      - Query Capability

        QueryBase
        http://localhost:8080/DevCafeProvider/resources

        Resource shape
        http://localhost:8080/DevCafeProvider/resourceShapes/resource

- Using a tomcat in the eclipse IDE
  - http://help.eclipse.org/indigo/index.jsp?topic=%2Forg.eclipse.jst.server.ui.doc.user%2Ftopics%2Ftomcat.html
- Using a Jetty app server
  - ✓ Configure jetty
    - o add needed resources
    - o in pom.xml
      - include test resources
      - start server
      - start OLSC4Catalog
      - start developed provider
  - ✓ Alter ServletListener class
    - o timing issues
    - o provider needs to register with OSLC4JCatalog a bit delayed
  - ✓ Add a run configuration for jetty

How to implement an OSLC Provider with OSLC4J

Stefan Paschke, ViF

25th April 2013

How to implement a Provider

- ✓ Implement Resource Class
- ✓ Implement Provider Logic
  - ✓ Setup the JAVA EE Web Application
  - ✓ Implement Service for each resource type
    - o Provision of Resources – HTTP GET
    - o Altering Resources
      - ▪ HTTP PUT, POST, DELETE
    - o Route requests to underlying logic
  - ✓ Other Configurations
- ✓ Running the provider
- For Testing
  - Implement Junit Tests
  - Implement Consumer logic
    - o Retrieve
    - o Update
    - o (Create, Delete)

Questions?

Thank you for your attention!