

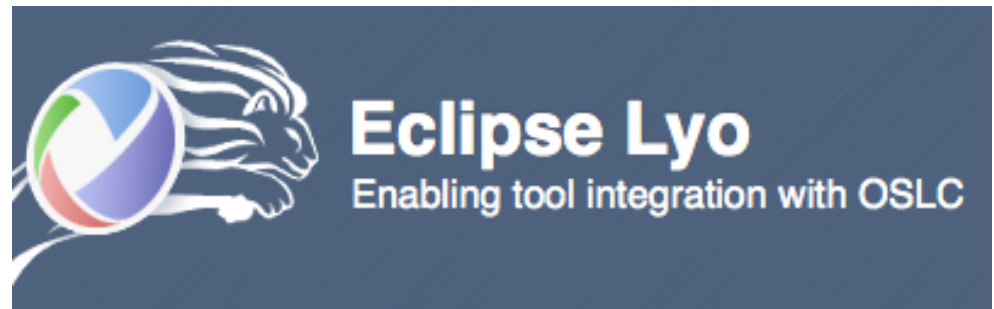


Open Services for Lifecycle Collaboration
open community. open interfaces. open possibilities.

Eclipse Lyo Overview

<http://eclipse.org/lyo>

Michael Fiedler, Eclipse Lyo committer





Agenda

- **Eclipse Lyo background**
- **Content and Plans**
- **Test Suites**
- **OSLC4J**
- **Samples**
- **Next Steps**



OSLC Specifications

- OSLC Core spec defines
 - ▶ HOW to use HTTP and RDF, how to define resources and services
 - ▶ Defines some common resource types and properties
- OSLC domain specs (Change Management, Requirements, etc.)
 - ▶ Define WHAT resources and services required in the domain
 - ▶ Resource types, properties and relationships
 - ▶ Service providers, creation factories, query capabilities, operations
- But, what do I do with an OSLC spec? How do I use it for real integrations?
 - ▶ Commercial tools available with OSLC APIs
 - ▶ Many organizations developing OSLC integrations in-house
 - ▶ Educational and research institutions developing tool integrations based on OSLC
 - ▶ Would be good to have open source sample code, SDKs, tests, etc to help get started.



Eclipse Lyo

Enabling tool integration with OSLC

The Eclipse Lyo project focuses on providing an SDK to help the Eclipse community to adopt **OSLC** (Open Services for Lifecycle Collaboration) specifications and build OSLC-compliant tools.

Check out the [original proposal](#).

What is OSLC?

An open community

We are an open, industry-wide **community** that is dedicated to reducing the barriers between lifecycle tools.

We want to help others build software that easily integrates with other software, which will let you build your ideal development environment, minimize frustration, and save time and money.

Writing open specifications

We write **specifications** that are...

- **Flexible:** Our specifications allow any OSLC-compliant tool to easily use the data of any other OSLC-compliant tool
- **Free to adopt:** All of our specifications are free to use.
- **Based on open data standards:** Our specifications use REST interfaces and URLs to expose lifecycle data

For many types of software

OSLC's scope started with Application Lifecycle Management (ALM) and is expanding to include integrations across Product Lifecycle Management (PLM) and IT Service Management (ISM/DevOps)

As OSLC expands, we want to help the Eclipse community to build OSLC-compliant software.

Lyo features

SDK

We'll provide the pieces to get your client or server up and running with OSLC. We're focusing on Java initially; however, JavaScript and other languages will soon follow.

Test suite

Utilize a test suite to help build interoperable OSLC tools.

Reference implementations

See how OSLC works directly with working samples and with a simple server to test against.

Project status

We are making the initial contributions for the project. View the detailed **project summary here**.

Additional project details on plan development can be found on the **project wiki** as well.

Source code now available in **Git repo!**



Keep in touch!

Join the **lyo-dev** mailing list.

You can also **follow us on Twitter**.

See also

■ **Learn more** about OSLC

■ **Get involved** with the OSLC community



Eclipse Lyo

- Evolution of OSLC tool repositories on SourceForge and some private to OSLC community participants
- Project approved by Eclipse PMC in July 2011 with the goal of providing an SDK to enable adoption of OSLC specifications, including
 - Code libraries
 - Reference implementations of specifications
 - Test suites and test reporting
 - Samples, tutorials and documentation
 - **NOT** a plugin for the Eclipse IDE, **NOT** related to OSGI – although Lyo artifacts could be used in Eclipse plugins.
- Eclipse chosen as the home for the project for its mature governance model and IP policies.
- Eclipse community includes tool vendors and tool interop projects. Other participants in the OSLC community are also active in Eclipse projects related to OSLC.
 - Tasktop (Mylyn)
 - Oracle (Hudson)
 - Institut Telecom
- Project content is dual-licensed under the Eclipse Public License and the Eclipse Distribution License



Agenda

- Eclipse Lyo background
- **Content and Plans**
- Test Suites
- OSLC4J
- Samples
- Next Steps



Initial Lyo Content

- Initial code contributions live in September 2011
 - Reference Implementations for OSLC (RIOs) for the Change, Requirement and Architecture Management specifications.
 - Provides samples of implementations
 - Enable prototyping and experimentation during spec development
 - Possible starting point for integration adapters or new service providers
 - Co-developed with the OSLC test suite to improve both
 - OSLC Test Suite and Reports
 - Measure implementation compliance against Core and domain specifications
 - Improve implementation quality by finding bugs
 - Currently covers core and CM. Other domains will follow
 - Initial focus is on MUST items, followed by SHOULD and MAY
 - Reports provide both summary and detailed results
 - Samples
 - Change Management adapter for Bugzilla
 - Change Management adapter for Microsoft Excel



Plans for new content in 2012

- Code libraries
 - OSLC4J SDK for Java with example implementations
 - Other technologies (.NET, PHP, JavaScript, Python, Perl)
 - ▶ Code contributions welcome

- Test Suites
 - Increase domain coverage
 - Increase specification coverage within domains
 - Move the tests towards a true compliance suite

- Samples
 - OAuth consumer and provider samples
 - Sample integrations with Microsoft products
 - Sample integrations with Rational Jazz products
 - OSLC Workshop/Tutorial code



Where is Lyo at today?

- See project plans at: <http://wiki.eclipse.org/Lyo/ProjectPlans>
- M1/M2 Milestones
 - ▶ M1 (4Q2011)
 - Focus was on test suite enhancements
 - 90% MUST coverage for Core and Change Management
 - Test suite reporting
 - Many bug fixes and enhancements
 - ▶ M2 (1Q2012)
 - OSLC4J SDK for Java – initial contribution of source
 - Example implementations based on OSLC4J
 - Start release engineering work to build consumable jars
 - Continue test suite work
 - Additional domains (QM)
 - Improvements to core tests
- What's not there yet
 - ▶ As of today, need to build the OSLC4J SDK and test suite from source
 - ▶ OSLC4J consumable JARs are close – working on publishing to Eclipse's Maven repo.
 - ▶ Getting started: <http://wiki.eclipse.org/Lyo>



Agenda

- Eclipse Lyo background
- Content and Plans
- **Test Suites**
- OSLC4J
- Samples
- Next Steps



OSLC Test Suites

- OSLC Core and Domain coverage
 - ▶ Initial focus on Core and CM testable MUSTs
 - ▶ QM coverage underway
 - ▶ Code contributions welcome

- Test suite as an OSLC consumer
 - ▶ Example of how to interact with an OSLC provider
 - ▶ Does not yet include OAuth interactions
 - ▶ Patterns for GET/POST/PUT
 - ▶ Patterns for validating and handling RDF using Jena

- Getting started with the tests and reports: <http://wiki.eclipse.org/Lyo/BuildTestSuite>

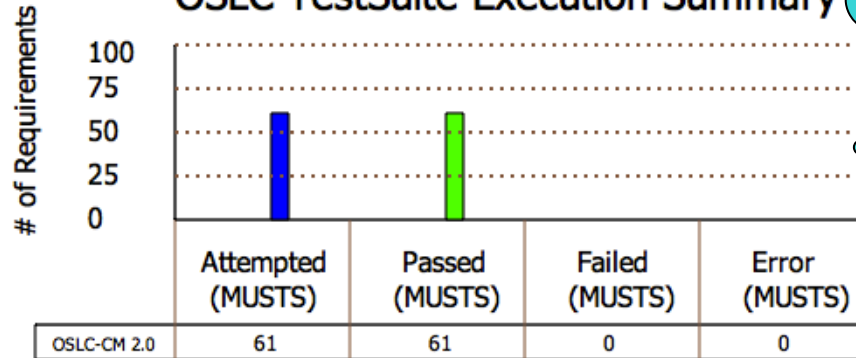


OSLC Test Suite

As an OSLC Compliance Assessor

Coverage Metrics

OSLC TestSuite Execution Summary



Distinct Compliance Levels

- Attempted = Pass + Fail + Error. # of Tests Executed for a Specification Type
- Pass: # of Test(s) achieving the respective test design's expected result
- Fail: # of Test(s) deviating from the respective test design's expected result.
- Error: # of Inconclusive Results. Test executions encountering error due to poor test design, faulty environment or invalid configuration. Test results could not be assessed.

OSLC compliance

Report Date	OSLC Domain	Version	OS Service Provider	Compliance Level	Test Coverage Statement	Test Development Statement
Thu Nov 17 00:04:44 EST 2011	OSLC-CM	2.0	JIRA	Level 1 Compliance	54.2% (58/107) 58 of the 107 OSLC-CM 2.0 MUST requirements are currently testable via the Lyo OSLC testsuite.	87.9% (51/58) 51 of the 58 testable MUST requirements currently have JUnit test case coverage within the Lyo OSLC testsuite

Non-Compliant: One or more attempted tests covering a MUST requirement has encountered a failure or error


Level 1 Compliance: All Attempted Tests covering a MUST requirement are Passing and free of failure or error

Level 2 Compliance: All Attempted Tests covering a MUST and SHOULD requirement are Passing and free of failure or error

Level 3 Compliance: All Attempted Tests covering a MUST, SHOULD and MAY requirement are Passing and free of failure or error

Note: This testsuite will continue to evolve and expand. Requirements may have one or more associated test(s) for coverage to address positive and negative input behaviors.



- 

eclipse

[Home](#) [Downloads](#) [Users](#) [Members](#) [Committers](#) [Resources](#) [Projects](#) [About Us](#)

Navigation
 - [Main Page](#)
 - [Community portal](#)
 - [Current events](#)
 - [Recent changes](#)
 - [Random page](#)
 - [Help](#)**Toolbox**

[Page](#)

[Discussion](#)

[View source](#)

[History](#)

[Edit](#)

Lyo/BuildTestSuite

< Lyo

Contents [\[hide\]](#)
 - 1 Building and running the Lyo OSLC Test Suite
 - 1.1 Prerequisites
 - 1.2 Clone the Lyo OSLC Test Suite git repository

```
<testclass package="org.eclipse.lyo.testsuite.server.oslcv2tests" name="ChangeRequestRdfXmlTests">
  <testcase level="MUST">changeRequestHasOneTitle</testcase>
  <testcase level="MUST">changeRequestHasAtMostOneDescription</testcase>
<testclass package="org.eclipse.lyo.testsuite.server.oslcv2tests" name="CreationAndUpdateRdfXmlTests">
  <testcase level="SHOULD">createValidResourceUsingRdfXmlTemplate</testcase>
  <testcase level="SHOULD">createResourceWithInvalidContent</testcase>
```



Agenda

- **Eclipse Lyo background**
- **Content and Plans**
- **Test Suites**
- **OSLC4J**
- **Samples**
- **Next Steps**



OSLC4J Overview

- Getting started: <http://wiki.eclipse.org/Lyo/BuildingOSLC4J>
- Java SDK for OSLC provider and consumer implementations
 - ▶ Based on OSLC related Java annotations and JAX-RS for REST services
 - ▶ Includes a Change Management reference implementation and other samples
 - ▶ Modular structure avoids forcing dependence on specific technologies.
 - ▶ Jena and Apache Wink provide RDF, JSON and JAX-RS capabilities out the box
 - ▶ Implementers can choose to use OSLC4J with other “providers” such as OpenRDF and Jersey
- What OSLC4J and JAX-RS handle for you
 - ▶ Resource shapes and service provider documents
 - ▶ Marshaling of Java objects to RDF (XML or JSON)
 - ▶ Un-marshaling of RDF (XML or JSON) to Java objects
 - ▶ Mapping REST service calls (GET, POST, PUT, DELETE) to Java methods
- What OSLC4J and JAX-RS do not handle for you
 - ▶ Persistence of your OSLC resources
 - ▶ Business logic for mapping Java objects to native resources
 - ▶ Automatic support for OSLC query syntax (working on some helpers)



Implementing providers with OSLC4J – JAX-RS Application

- Some basics
 - ▶ Set up an OslcWink application (extends JAX-RS Application class)

```
public final class Oslc4JChangeManagementApplication
    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(ChangeRequestResource.class);

        RESOURCE_SHAPE_PATH_TO_RESOURCE_CLASS_MAP.put(Constants.PATH_CHANGE_REQUEST, ChangeRequest.class);
    }

    public Oslc4JChangeManagementApplication()
        throws OslcCoreApplicationException,
            URISyntaxException
    {
        super(RESOURCE_CLASSES,
            OslcConstants.PATH_RESOURCE_SHAPES,
            RESOURCE_SHAPE_PATH_TO_RESOURCE_CLASS_MAP);
    }
}
```




Implementing providers with OSLC4J - Namespaces

- Some basics
 - ▶ Tell OSLC4J what namespaces you are going to use

```
@OslcSchema ({
    @OslcNamespaceDefinition(prefix = OslcConstants.DCTERMS_NAMESPACE_PREFIX,
    @OslcNamespaceDefinition(prefix = OslcConstants.OSLC_CORE_NAMESPACE_PREFIX,
    @OslcNamespaceDefinition(prefix = OslcConstants.OSLC_DATA_NAMESPACE_PREFIX,
    @OslcNamespaceDefinition(prefix = OslcConstants.RDF_NAMESPACE_PREFIX,
    @OslcNamespaceDefinition(prefix = OslcConstants.RDFS_NAMESPACE_PREFIX,
    @OslcNamespaceDefinition(prefix = Constants.CHANGE_MANAGEMENT_NAMESPACE_PREFIX,
    @OslcNamespaceDefinition(prefix = Constants.FOAF_NAMESPACE_PREFIX,
    @OslcNamespaceDefinition(prefix = Constants.QUALITY_MANAGEMENT_PREFIX,
    @OslcNamespaceDefinition(prefix = Constants.REQUIREMENTS_MANAGEMENT_PREFIX,
    @OslcNamespaceDefinition(prefix = Constants.SOFTWARE_CONFIGURATION_MANAGEMENT_PREFIX, namespaceURI = OslcConstants.DCTERMS_NAMESPACE),
    namespaceURI = OslcConstants.OSLC_CORE_NAMESPACE),
    namespaceURI = OslcConstants.OSLC_DATA_NAMESPACE),
    namespaceURI = OslcConstants.RDF_NAMESPACE),
    namespaceURI = OslcConstants.RDFS_NAMESPACE),
    namespaceURI = Constants.CHANGE_MANAGEMENT_NAMESPACE),
    namespaceURI = Constants.FOAF_NAMESPACE),
    namespaceURI = Constants.QUALITY_MANAGEMENT_NAMESPACE),
    namespaceURI = Constants.REQUIREMENTS_MANAGEMENT_NAMESPACE),
    namespaceURI = Constants.SOFTWARE_CONFIGURATION_MANAGEMENT_NAMESPACE)
})
package org.eclipse.lyo.oslc4j.changemanagement;

import org.eclipse.lyo.oslc4j.core.annotation.OslcNamespaceDefinition;
import org.eclipse.lyo.oslc4j.core.annotation.OslcSchema;
import org.eclipse.lyo.oslc4j.core.model.OslcConstants;
```



Implementing providers with OSLC4J – Annotating Resources

- Model OSLC resources with Plain Old Java Objects
 - ▶ Tell OSLC4J which class maps to an OSLC resource type (e.g. ChangeRequest)

```
@OslcNamespace(Constants.CHANGE_MANAGEMENT_NAMESPACE)
@OslcResourceShape(title = "Change Request Resource Shape", describes = Constants.TYPE_CHANGE_REQUEST)
public final class ChangeRequest
    extends AbstractResource
{
    private Boolean    approved;
    private Boolean    closed;
    private Date       closeDate;
    private Date       created;
    private String     description;
```

- ▶ OSLC Java notations to decorate POJOs
 - Decorate the getters for resource attributes with resource shape information (OSLC name, description, range, cardinality, read/write ability).

```
@OslcDescription("A unique identifier for a resource. Assigned by the service provider when a resource i
@OslcOccurs(Occurs.ExactlyOne)
@OslcPropertyDefinition(OslcConstants.DCTERMS_NAMESPACE + "identifier")
@OslcReadOnly
@OslcTitle("Identifier")
public String getIdentifier()
{
    return identifier;
}
```



Implementing providers with OSLC4J – Annotating Services

- Decorate Java methods with OSLC and JAX-RS annotations
 - ▶ HTTP GET for all ChangeRequests – supports RDF XML, XML and JSON

```
@GET
@Produces({OslcMediaType.APPLICATION_RDF_XML, OslcMediaType.APPLICATION_XML, OslcMediaType.APPLICATION_JSON})
public ChangeRequest[] getChangeRequests(@QueryParam("oslc.where") final String where)
{
    final List<ChangeRequest> results = new ArrayList<ChangeRequest>();
    //Business logic for retrieving change requests would replace the following line
    final ChangeRequest[] changeRequests = Persistence.getChangeRequests();

    for (final ChangeRequest changeRequest : changeRequests)
    {
        changeRequest.setServiceProvider(ServiceProviderSingleton.getServiceProviderURI());
        results.add(changeRequest);
    }
    return results.toArray(new ChangeRequest[results.size()]);
}
```



Implementing providers with OSLC4J

- Decorate Java methods with OSLC and JAX-RS
 - ▶ HTTP POST to create new ChangeRequest

```
@OslcCreationFactory
(
    title = "Change Request Creation Factory",
    label = "Change Request Creation",
    resourceShapes = {OslcConstants.PATH_RESOURCE_SHAPES + "/" + Constants.PATH_CHANGE_REQUEST},
    resourceTypes = {Constants.TYPE_CHANGE_REQUEST},
    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 addChangeRequest(@Context final HttpServletRequest httpRequest,
                                @Context final HttpServletResponse httpResponse,
                                final ChangeRequest changeRequest)
    throws URISyntaxException
{
    //Business logic for creating a new change request goes here

    return Response.created(about).entity(changeRequest).build();
}
```



OSLC consumers using OSLC4J

- OSLC REST Client based on Apache Wink
 - ▶ Included in the OSLC4JWink project
 - ▶ Methods to Get/Create/Update/Delete OSLC Resources
 - ▶ Handles Java/RDF marshaling
 - ▶ Requires same POJO resource definitions used by the service provider
- ▶ See the OSLC4J Junit Tests for examples of using the client

```
protected void testRetrieve(final String mediaType)
    throws URISyntaxException
{
    assertNotNull(CREATED_CHANGE_REQUEST_URI);

    final OslcRestClient oslcRestClient = new OslcRestClient(PROVIDERS,
                                                             CREATED_CHANGE_REQUEST_URI,
                                                             mediaType);

    final ChangeRequest changeRequest = oslcRestClient.getOslcResource(ChangeRequest.class);

    verifyChangeRequest(mediaType,
                        changeRequest,
                        true);
}
```



Agenda

- Eclipse Lyo background
- Content and Plans
- Test Suites
- OSLC4J
- **Samples**
- Next Steps



OSLC4J Samples

- Change Management provider
 - ▶ Working on delegated UI, but provider available
- Stock Quote provider
 - ▶ example of OSLC app not based on a current domain specification
 - ▶ Uses OSLC core techniques to define a new resource type (StockQuote)
- OSLC4J Registry application
 - ▶ Implementation of a standalone OSLC catalog registry server
 - ▶ REST API to allow service providers to register/de-register themselves
 - ▶ Used by the OSLC4J CM and StockQuote samples



Other samples and reference implementation

- Bugzilla Adapter
 - ▶ Full CM service provider adapter for Bugzilla
 - ▶ Includes OAuth and Rational Jazz rootservices support
 - ▶ Good example for connecting to Rational Jazz OSLC providers
- OAuth sample web app
 - ▶ Sample code for handling OAuth tokens and authentication
- Reference implementations for OSLC (RIOs)
 - ▶ Example OSLC providers built using OpenRDF and a traditional servlet approach
 - ▶ Change, Architecture and Requirement Managemt providers
 - ▶ Include simple delegated UIs
 - ▶ Includes OSLC query support (oslc.where)
- Microsoft Excel change management adapter
 - ▶ Example of exposing rows of an Microsoft Excel sheet as change requests
 - ▶ Map columns to OSLC attributes

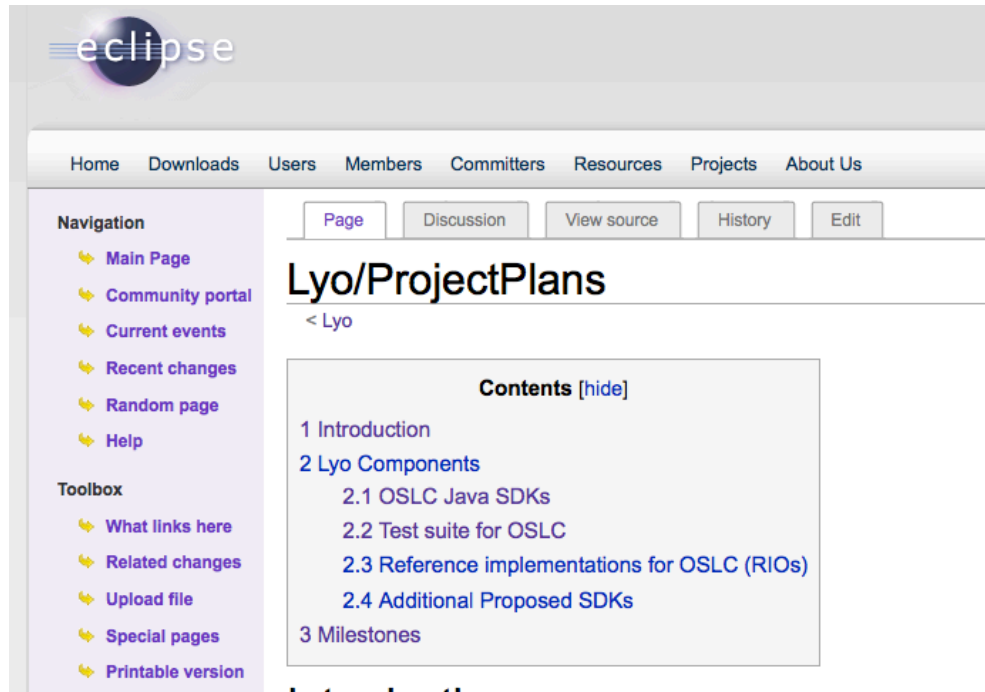


Agenda

- Eclipse Lyo background
- Content and Plans
- Test Suites
- OSLC4J
- Samples
- **Next Steps**



Participating in Lyo



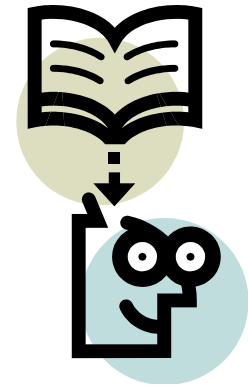
■ Participating in Lyo

- ▶ Lyo is currently source only – goal to have builds of consumable jars starting in February
- ▶ Looking for developers interested in promoting OSLC adoption by developing SDKs, reference implementations, compliance tests and examples
- ▶ Visit <http://wiki.eclipse.org/Lyo> to get more info, see milestone plans, etc
- ▶ Open Bugzilla requests at: https://bugs.eclipse.org/bugs/enter_bug.cgi?product=Lyo
- ▶ Subscribe to the lyo-dev@eclipse.org mailing list and introduce yourself.



Resources

- OSLC Web Site
 - ▶ <http://open-services.net>
- OSLC Primer
 - ▶ <http://open-services.net/primer>
- OSLC Tutorial
 - ▶ <http://open-services.net/tutorial>
- Open source - Eclipse Lyo Project
 - ▶ <http://eclipse.org/lyo>
 - ▶ <http://wiki.eclipse.org/Lyo>





Questions?

- Questions?



Backup

- OSLC4J project mapping - repo is `git://git.eclipse.org/gitroot/lyo/org.eclipse.lyo.core.git`
 - ▶ OSLC4J – Core project
 - ▶ OSLC4JJenaProvider – RDF and RDF/XML provider based on Jena
 - ▶ OSLC4JJSON4JProvider – Apache Wink JSON4J provider
 - ▶ OSLC4JWink – Apache Wink JAX-RS provider + OSLC client
 - ▶ OSLC4JRegistry – Sample catalog registry application.
 - ▶ OSLC4JTest – Test provider
 - ▶ OSLC4JTestTest – Junit Tests for Test provider
 - ▶ OSLC4JStockQuote – Stock Quote provider
 - ▶ OSLC4JStockQuoteCommon – Stock Quote common classes
 - ▶ OSLC4JStockQuoteTest – Junit Tests for StockQuote provider
 - ▶ OSLC4JCoreRelEng- release engineering files (master pom.xml for Maven)

- OSLC4J Change Mgmt sample – repo is : `git://git.eclipse.org/gitroot/lyo/org.eclipse.lyo.rio.git`
 - ▶ OSLC4JChangeManagement – CM provider
 - ▶ OSLC4JChangeManagementCommon – CM common classes
 - ▶ OSLC4JChangeManagementTest – Junit tests for CM provider