IBM TRIRIGA Application Platform
3.7.0

OSLC REST API Integration Guide

Note

Before using this information and the product it supports, read the information in "Notices" on page 27.

This edition applies to version 3, release 7, modification 0 of IBM® TRIRIGA® Application Platform and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Integrating data by using the OSLC REST API

Your product applications and some external applications can link and share data by using Open Services for Lifecycle Collaboration (OSLC) integration. OSLC specifications define how lifecycle applications represent, link to, and access resources based on established internet and linked-data standards including representational state transfer (REST) API architecture, resource description framework (RDF) specifications, and hyper-text transfer protocol (HTTP) methods. OSLC makes it easier for tools to work together and share data. By following the rules and methods that are defined by the specifications, applications can perform create, request, update, delete operations on the resources of another application.

The OSLC community is a group of software developers and organizations who are working to standardize how software lifecycle tools share data such as requirements, defects, test cases, and change history. Version 2.0 of the OSLC specification is supported for OSLC integration of applications. The specifications and other OSLC resources are available on the web.

IBM TRIRIGA provides an OSLC REST API as part of the on-premises IBM TRIRIGA Connector for Business Applications (CBA) license and the IBM Facilities and Real Estate Management on Cloud (TRIRIGA) Enterprise User license.

OSLC integration with IBM TRIRIGA is accomplished between a consumer application and TRIRIGA, the provider application. As the OSLC provider application, TRIRIGA makes its resource data available to the consumer application through containers, which are known as service providers. With the resource data made available, the consumer application can create links between its data and the resource data of TRIRIGA. You can configure and enable any part of TRIRIGA to act as an OSLC provider application.

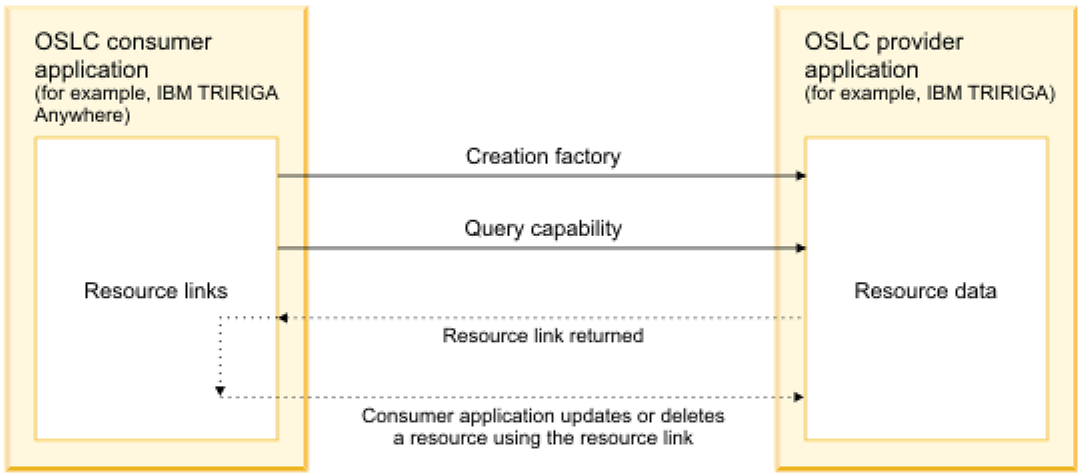
Consumer applications can find the URIs that identify the resources in the provider application and use those URIs to request query results, for example. For IBM TRIRIGA, service providers are available at `http://`*`yourserver`*`/oslc/sp`.

The URIs used in OSLC requests must be URL encoded.

Chapter 2. Integrating as an OSLC consumer

Your application, an OSLC consumer, can be configured to support two interaction types: query and creation. The resource links are obtained by the consumer application from the provider application.

The following figure illustrates the interactions between the consumer and provider applications. As an OSLC consumer, the application can query or create resources in the provider application and retain the links to those resources. With the links, the consumer application can make requests to the provider application to query, update, or delete the resources.


OSLC consumer application (for example, IBM TRIRIGA Anywhere)
Resource links
Creation factory
Query capability
Resource link returned
Consumer application updates or deletes a resource using the resource link
OSLC provider application (for example, IBM TRIRIGA)
Resource data

OSLC components

Service providers, resources, and ETags provide details that you need to consume IBM TRIRIGA data through OSLC.

OSLC terminology

Definitions are provided for IBM TRIRIGA OSLC terms.

consumer application	An application that uses the data in the central data warehouse for a specific business need.
creation factory	A URI that is used to create new resources by using HTTP POST.
provider application	In the context of OSLC, an application that makes its resource data available to the consumer application through containers that are called service providers. IBM TRIRIGA is a provider application for the IBM TRIRIGA Anywhere mobile applications.
provider record	A record that identifies the provider application and contains definitions for one or more OSLC interactions between a consumer application and the provider application.
public URI	The root URI that is used to access the OSLC provider application.
query capability	A base URI for forming query resource URIs.
resource	In the context of OSLC, a network data object or service that can be identified by a URI.
Resource Description Framework (RDF)	A framework for representing information on the web.

resource shape
A specification that defines a fixed list of properties for the resource, expected data types and values, and validation rules for new or changed resources.

resource type
In the context of OSLC, the type of data that is linked between integrated applications, for example, a work task status change request.

REST

Representation State Transfer: A software architectural style for distributed hypermedia systems like the World Wide Web. The term is also often used to describe any simple interface that uses XML (or YAML, JSON, plain text) over HTTP without an additional messaging layer such as SOAP.

service provider
In the context of OSLC, a container of resources that is hosted by a tool or product to enable the use of the resources.

shape document
A record that describes the resource shape and makes it available through the URI.

Service providers and service provider discovery

A *service provider* is a container or collection of resources that is hosted by a tool or product. Service providers support the grouping of similar resources, such as defects or tasks, that can be configured for integration.

A service provider in an OSLC provider application contains the resource data that can be linked to consumer application data through integration of the applications. The resource data in IBM TRIRIGA can be in multibyte languages. To integrate a consumer application and a provider application, the consumer must discover or identify the service providers that are available in the provider application.

In IBM TRIRIGA, a service provider can be simple or as complex as an application that contains many modules and business objects. You can discover the service providers in IBM TRIRIGA by using the following methods:

- From **Tools** > **System Setup** > **Integration** > **OSLC Manager**
- By using the URI: `http://`*`yourserver`*`/oslc/sp`

With each method, a list of service providers is returned. The following example shows the URI for employees: `http://`*`yourserver`*`/oslc/sp/Employee.`

The URI points to the service provider document that is in RDF/XML format. RDF/XML format is supported for shape documents, service providers, resources, and resource data. The consumer application can use the service provider form to determine which resources are available and what services they support, such as query or creation.

The OSLC service provider supports the OSLC creation factory and query capability operations that provide consumers with the URI to create or search resources that are supported by the service provider. The service provider document describes the available resources and the namespace mappings, and the operations that are supported by the service provider for those resources.

In the following sample response, the OSLC service provider is referred from the `rdfs:member` property. The service provider document for the domain shows the URI for work task:

```
<rdf:RDF>
<rdf:Description rdf:about="http://yourserver/oslc/sp">
<rdfs:member rdf:resource="http://yourserver/oslc/sp/WorkTask">
</rdf:Description>
</rdf:RDF>
```

OSLC namespaces

OSLC defines common namespaces. The **`prefixDefinition`** property shows all the prefix-to-namespace mappings that the service provider uses to describe the resources that it manages.

The namespace for a property must end with a # or /. For example, `http://`*`yourserver`*`/ns/property#`.

The following table shows a sample of OSLC namespaces:

Prefix	Namespace
rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`
oslc	`http://open-services.net/ns/core#`
dcterms	`http://purl.org/dc/terms/`
asset	`http://open-services.net/ns#`
foaf	`http://xmlns.com/foaf/0.1/`
rdfs	`http://www.w3.org/2000/01/rdf-schema#`
rr	`http://jazz.net/ns/ism/registry#`
spi	`http://jazz.net/ns/tririga`

The following excerpt from the service section of the service provider document shows the OSLC and RDF namespaces:

```
<oslc:ServiceProvider rdf:about="http://yourserver/oslc/sp/WorkTask">
  <oslc:prefixDefinition>
    <oslc:PrefixDefinition>
      <oslc:prefixBase rdf:resource="http://open-services.net/ns/core#"/>
      <oslc:prefix>oslc</oslc:prefix>
    </oslc:PrefixDefinition>
  </oslc:prefixDefinition>

  <oslc:prefixDefinition>
    <oslc:PrefixDefinition>
      <oslc:prefixBase rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
      <oslc:prefix>rdf</oslc:prefix>
    </oslc:PrefixDefinition>
  </oslc:prefixDefinition>
```

OSLC operations and resources

The OSLC service provider supports the creation factory and query capability operations for the resources that are available in the service provider document. A creation factory provides the **`oslc:creation`** creation URI that you use to create new resources by using HTTP POST. You use the **`oslc:queryBase`** query URI to select a resource collection that is managed by the service provider. When the resource is obtained, either by query or creation, the resource can be updated or deleted.

Creation factory operation

If the resource supports creation, there can be one creation factory operation. The following excerpt from a service provider document shows the creation factory operation, the URL for the resource shape, and the URL for the creation resource operation that creates the shape.

```
<oslc:creationFactory>
    <oslc:CreationFactory>
      <oslc:resourceType rdf:resource="http://jazz.net/ns/tririga#WorkTask"/>
        <oslc:resourceShape rdf:resource="http://yourserver/oslc/shapes/WorkTask"/>
        <oslc:creation rdf:resource="http://yourserver/oslc/so/WorkTask"/>
        <oslc:label>Create WorkTask</oslc:label>
            <dcterms:title>OSLC creation factory for WorkTask</dcterms:title>
    </oslc:CreationFactory>
```

```
</oslc:creationFactory>
        ......
```

Query capability operation

The query URI is **`oslc:queryBase`**, and the following example shows a search for a work task by using the request: `<oslc:queryBase rdf:resource="http://`*`yourserver`*`/oslc/spq/oslcwodetail"/>`:

```
    <oslc:queryCapability>
           <oslc:QueryCapability>
             <oslc:resourceType rdf:resource="http://jazz.net/ns/tririga#WorkTask"/>
             <oslc:queryBase rdf:resource="http://yourserver/oslc/so/WorkTask"/>
             <oslc:labelQuery>WorkTask</oslc:label>
             <dcterms:taskname>OSLC query capability for WorkTask</dcterms:taskname>
           </oslc:QueryCapability>
         </oslc:queryCapability>
```

When you define an OSLC resource shape, all of the queries for the resource are available through the service provider. The queries are made available with OSLC query capabilities.

An OSLC resource shape is defined by a report that is defined in the IBM TRIRIGA Report Manager. The resource shape uses the report as a template to define the properties available for your resource. These properties are then returned when you run a query capability.

Each query capability contains a property that is named query base that you use to apply extended criteria to your resource. These criteria make it possible to predefine filters for the same shape. The **Query Base** field in query capability holds the name of a query that is compatible with the query that the resource is defined on. This query, if defined, is used for filtering. You can use TRIRIGA parameters in the query, such as **`$$USERID$$`**, **`$$RECORDID$$`**, or **`$$PARENT::SECTIONAME::FIELDNAME$$`**. The query base is a list and the contents of the list change when the resource changes. For example, the MyWorkTask query returns a list of work tasks that are assigned to the user who makes the query request.

Resource shapes

A *resource shape* is a Resource Description Framework (RDF) file that provides a description of the resource data types that can be used in an interaction. The shape contains a list of attributes of the resource.

You can view the RDF for a resource in the Preview tab of the resource form.

A resource shape is similar to an XML schema in the way that it defines the data structure of the resource.

Shape documents

A shape document in OSLC is an electronic way to see what a resource looks like including all of its dependencies, attributes, and properties. For example, a work task shape document lists the work task resource details.

Shape documents cover all resources, including assets, companies, purchase orders, and work tasks. A shape document also shows what is required. The resource shape document can include links to the shape documents for the child objects. RDF/XML is used as the format for the shape document.

A resource shape displays properties, actions, and the linked resources that are defined for the resource.

In IBM TRIRIGA, when you create a new resource with the OSLC Resource form, you must specify the module, business object, and a business object query, or you must select a module and multi-business object query. Begin by creating the query in IBM TRIRIGA. When you create the display columns in the query, you are defining the initial properties. You can use the **Import All Fields** action in the form to import the display columns from the query as resource properties. The import process attempts to set the IBM TRIRIGA fields to the corresponding OSLC property values, such as read-only. You can modify the fields after the import and you can remove properties. The **`dcterms:identifier`** property creates the record ID during the import process. The**`triRecordIdSY`** field is required if you plan to update the resource.

The following table shows how IBM TRIRIGA field types map to OSLC property value types:

IBM TRIRIGA field type	OSLC property value type
Boolean	OslcPropertyValueType.Boolean
Business Object	OslcPropertyValueType.String
Classification	OslcPropertyValueType.String
Classification Rollup	OslcPropertyValueType.Decimal
Color	OslcPropertyValueType.String Note: The URI must be URL encoded when you filter for color fields. Replace the # symbol with %23.
Control Number	OslcPropertyValueType.String
Date	OslcPropertyValueType.String
Date and Time	OslcPropertyValueType.String
Duration	OslcPropertyValueType.String
Financial Rollup	OslcPropertyValueType.Decimal
Image	OslcPropertyValueType.String
Label Only	OslcPropertyValueType.String
List	OslcPropertyValueType.String
Number	OslcPropertyValueType.Decimal
Password	OslcPropertyValueType.String
System Read Only	OslcPropertyValueType.String
Text	OslcPropertyValueType.String
Time	OslcPropertyValueType.String
UOM	OslcPropertyValueType.String
Url	OslcPropertyValueType.String

A linked resource points to the resource to be linked. It provides the association strings describe the relationship. The following are examples of association strings: Has Asset, Manages, and Assigned To. You also can add a linked resource to a property. A linked resource is allowed, and is optional, for locator fields and smart sections.

Example: Work task shape document

A work task shape document lists all of the properties, attributes, and dependencies of a work task. The following code shows an excerpt from a work task shape document named `WorkTask`. Four properties are included in this resource shape document but the document can have many more properties listed.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:oslc="http://open-services.net/ns/core#"
    xmlns:spi="http://jazz.net/ns/tririga/property#"
    xmlns:dcterms="http://purl.org/dc/terms/">
  <oslc:ResourceShape rdf:about="http://yourserver/oslc/shapes/WorkTask">
    <oslc:property>
      <oslc:Property>
        <oslc:representation rdf:resource="http://open-services.net/ns/core#Either"/>
        <oslc:readOnly>false</oslc:readOnly>
        <oslc:occurs rdf:resource="http://open-services.net/ns/core#Exactly-one"/>
        <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        <dcterms:title rdf:datatype="http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral"
        >ID</dcterms:title>
        <oslc:name>RecordInformation.triIdTX>/oslc:name>
        <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/tririga/property#triIdTX"/>
      </oslc:Property>
    </oslc:property>
    <oslc:property>
      <oslc:Property>
        <oslc:representation rdf:resource="http://open-services.net/ns/core#Either"/>
        <oslc.readOnly>false</oslc:readOnly>
        <oslc:occurs rdf:resource="http://open-services.net/ns/core#Exactly-one"/>
        <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
        <oslc:defaultValue>TIMESTAMP</oslc:defaultValue>
        <dcterms:title rdf:datatype="http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral"
        >Planned Start</dcterms:title>
        <oslc:name>RecordInformation.triPlannedStartDT</oslc:name>
        <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/tririga/property#triPlannedStartDT"/>
      </oslc:Property>
    </oslc:property>
    <oslc:property>
      <oslc:Property>
        <oslc:representation rdf:resource="http://open-services.net/ns/core#Either"/>
        <oslc:readOnly>true</oslc:readOnly>
        <oslc:occurs rdf:resource="http//open-services.net/ns/core#Exactly-one"/>
        <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#decimal"/>
        <dcterms:title rdf:datatype="http//www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral"
        >Actual Total Cost</dcterms:title>
        <oslc:name>RecordInformation.triActualTotalCostNU</oslc:name>
        <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/tririga/property#triActualTotalCostNU"/>
      </oslc:Property>
    </oslc:property>
    <oslc:property>
      <oslc:Property>
        <oslc:representation rdf:resource="http://open-services.net/ns/core#Either"/>
        <oslc:readOnly>false</oslc:readOnly>
        <oslc:occurs rdf:resource="http//open-services.net/ns/core#Exactly-one"/>
        <oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
        <dcterms:title rdf:datatype="http//www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral"
        >Task Name</dcterms:title>
        <oslc:name>RecordInformation.triNameTX</oslc:name>
        <oslc:propertyDefinition rdf:resource="http://jazz.net/ns/tririga/property#trNameTX"/>
      </oslc:Property>
    </oslc:property>
    <dcterms:title>WorkTask</dcterms:title>
  </oslc:ResourceShape>
</rdf:RDF>
```

Number fields

An attribute for scale is provided for number fields in OSLC. Scale is the number of digits to the right of the decimal for the number.

If the number has a custom display mask, the scale value that is returned is based on the custom display mask. If the display mask is missing, and there is no unit of measure (UOM) set, the value of two is returned by default. If there is a UOM set for the number field but there is no display mask set for the field, the scale of the UOM display mask is returned. The characters 0 and # in the display mask are considered when the scale is determined.

Example: Scale in number fields

The following example shows the returned results with a scale value of 3:

```
<oslc:property>
 <oslc:Property>
<spi:scale> 3 </spi:scale>
<oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#decimal" />
<oslc:readOnly> false </oslc:readOnly>
<oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one" />
<dcterms:title rdf:datatype="http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral">
Cost </dcterms:title>
<oslc:name> triCostNU </oslc:name>
<oslc:propertyDefinition rdf:resource="http://jazz.net/ns/spi#triCostNU" />
</oslc:Property>
</oslc:property>
```

List and UOM properties

You can view the values that are allowed for list and unit of measure (UOM) properties.

A list or UOM property has an **`oslc:allowedValues`** element that has a resource URI. The URI returns a list of available values that can be used for that list or UOM property.

Example

The following example shows a property with the **`oslc:allowedValues`** element, and then the list of allowed values that results from the URI in the element:

```
<oslc:property>
 <oslc:Property>
<oslc:defaultValue>Japan Yen</oslc:defaultValue>
<oslc:readOnly>false</oslc:readOnly>
<oslc:valueType rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
<oslc:usage rdf:resource="http://jazz.net/ns/ism/datatypes/
smarter_physical_infrastructure#uom" />
<oslc:occurs rdf:resource="http://open-services.net/ns/core#Zero-or-one" />
<oslc:name>exampleUOM</oslc:name>
<dcterms:title rdf:datatype="http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral">
exampleUOM</dcterms:title>
<oslc:allowedValues rdf:resource="http://yourserver/oslc/system/list/
resourceName/spi:exampleUOM" />
<oslc:propertyDefinition rdf:resource="http://jazz.net/ns/spi#exampleUOM" />
 </oslc:Property>
</oslc:property>
```

```
<rdf:RDF>
<oslc:allowedValues rdf:about="http://yourserver/oslc/system/list/
resourceName/spi:exampleUOM">
<oslc:allowedValues>
 <oslc:allowedValues>
<oslc:allowedValue>US Dollars</oslc:allowedValue>
<oslc:allowedValue>Swedish Krona/Kronor</oslc:allowedValue>
<oslc:allowedValue>Brazilian Real</oslc:allowedValue>
<oslc:allowedValue>Russian Ruble</oslc:allowedValue>
<oslc:allowedValue>Norwegian Krone</oslc:allowedValue>
<oslc:allowedValue>New Zealand Dollars</oslc:allowedValue>
<oslc:allowedValue>United Kingdom Pounds</oslc:allowedValue>
<oslc:allowedValue>Thai Baht</oslc:allowedValue>
<oslc:allowedValue>Canadian Dollars</oslc:allowedValue>
<oslc:allowedValue>Egyptian Pound</oslc:allowedValue>
<oslc:allowedValue>Euro</oslc:allowedValue>
<oslc:allowedValue>Polish Zloty</oslc:allowedValue>
<oslc:allowedValue>Korea Won</oslc:allowedValue>
<oslc:allowedValue>Hungarian Forint</oslc:allowedValue>
<oslc:allowedValue>South Africa Rand</oslc:allowedValue>
<oslc:allowedValue>Switzerland Francs</oslc:allowedValue>
<oslc:allowedValue>Japan Yen</oslc:allowedValue>
<oslc:allowedValue>Australia Dollars</oslc:allowedValue>
<oslc:allowedValue>Israeli New Shekel</oslc:allowedValue>
<oslc:allowedValue>Danish Krone</oslc:allowedValue>
<oslc:allowedValue>Indian Rupees</oslc:allowedValue>
<oslc:allowedValue>clIndiaRupee</oslc:allowedValue>
 </oslc:allowedValues>
</oslc:allowedValues>
</oslc:allowedValues>
</rdf:RDF>
```

ETags

An ETag (entity tag) is an HTTP header that is used to validate that the client (such as a mobile device) has the most recent version of a record. When a GET request is made, the ETag is returned as a response header. The ETag also allows the client to make conditional requests.

In addition to supporting the basic update methods of HTTP PUT and HTTP PATCH, OSLC also supports conditional updates. Conditional updates use HTTP entity tags and If-Match headers to validate whether clients have the most current entity for a resource. This process is used to detect bad updates and race conditions. For example, if two clients load the same resource, OSLC sends the ETag header with the response.

The ETag value is the date and time of the last update to the resource. The client stores the ETag header value and sends it as part of HTTP If-Match header for a subsequent update request. The server evaluates the If-Match header and determines whether the client has an old version or the most recent version of the resource. If the server determines that the client version is old, it sends back an `HTTP 412 precondition failed` response. The client gets the resource again and submits a request that is based on the updated ETag. However, if the server determines that the client version is the most recent version, the update is implemented unless any business validation or database constraints are found.

The client can also submit the request without the If-Match header or with the If-Match header value set to * (asterisk). Submitting this request is semantically equivalent to having no If-Match header in the update request. In both cases, the update is unconditional. If the resource that is referred by the URI exists and no business validation or database constraints are found, the update is implemented.

IBM TRIRIGA URIs for OSLC

The URIs you use to connect to IBM TRIRIGA by using OSLC are unique.

You use the following URIs to connect with IBM TRIRIGA:

OSLC component	**Description**
System resource page	`http://`*`yourserver`*`/oslc`
Login	`http://`*`yourserver`*`/oslc/login`
Logout	`http://`*`yourserver`*`/oslc/logout`
Creation factory	`http://`*`yourserver`*`/oslc/so`
Query capability	`http://`*`yourserver`*`/oslc/spq`
Query details	`http://`*`yourserver`*`/oslc/so`
Resource shape	`http://`*`yourserver`*`/oslc/shapes` To access the resource shape page, use `http://`*`yourserver`*`/oslc/shapes/`*`ResourceShapeName`*
Service provider	`http://`*`yourserver`*`/oslc/sp`

Chapter 3. Working with OSLC resources

You use HTTP methods to define how users create, query, update, or delete OSLC resources.

IBM TRIRIGA security is applied to all data activities. A user cannot create, query, update, or delete a record without appropriate security access.

Querying OSLC resources

You can query OSLC resources by using the HTTP GET method or the HTTP POST method. You change the query parameters to control how users search resources. OSLC defines a lightweight query syntax that is based on the SPARQL standard to query resources.

Query by using GET or POST methods

When you query OSLC resources, you can use either the HTTP GET method or HTTP POST method.

When you query OSLC resources by using HTTP GET, you specify the query parameters in the URI. If the URI is longer than 2000 characters, you must query by using HTTP POST, instead of HTTP GET. You can also use HTTP POST if the URI is less than 2000 characters but still long, or if you want to hide the query parameters so that they are not displayed in the URI.

When you query by using HTTP POST, you set the HTTP header Content-Type to application/x-www-form-urlencoded, send the URI without parameters, and specify the query parameters in the HTTP request body.

Example: Query by using HTTP GET

The following is an example of a query by using HTTP GET. All of the query parameters are displayed in the URI.

```
http://yourserver/oslc/spq/WorkTaskQuery?oslc.select=
spi:triNameTX,spi:RCA{spi:triRCARemedyCL}&oslc.where=
spi:RCA{spi:triRCARemedyCL="Clean"}&oslc.orderBy=%2Bspi:triNameTX
```

Example: Querying using HTTP POST

The following is an example of the same query by using HTTP POST. The URI becomes shortened.

```
http://yourserver1/oslc/spq/WorkTaskQuery
```

HTTP body

The HTTP request body contains all of the query parameters.

```
oslc.select=spi:triNameTX,spi:RCA{spi:triRCARemedyCL}&oslc.where=
spi:RCA{spi:triRCARemedyCL="Clean"}&oslc.orderBy=%2Bspi:triNameTX
```

OSLC query parameters

The OSLC query parameters provide options for how OSLC resources are queried. For example, the **oslc.orderBy** parameter defines the order of query results. The OSLC HTTP query parameters that are supported are **oslc.properties**, **oslc.where**, **oslc.orderBy**, **oslc.select**, **oslc.pageSize**, and **pageno**.

If a property is not in the **General** section of a form, you must specify the section name for the property, in the format `sectionname#fieldname`. For example, `triDetails#triTaskTypeCL`.

oslc.properties parameter

The **`oslc.properties`** query parameter specifies the list of properties for an OSLC resource. The properties can be from the resource itself or from a linked resource. It is used to get a partial representation of the resource. The **`oslc.properties`** parameter is not applicable to collection resources. A collection resource is an OSLC resource that has other OSLC resources as members.

Example: Requesting attributes

The following request example specifies that the values for the **`shortTitle`** and **`isTask`** attributes are returned in the results:

```
http://yourserver/oslc/so/WorkTask/
337?oslc.properties=oslc:shortTitle,spi:isTask
```

Example: Requesting attributes from linked resources

The following request specifies that the value for the name of the customer organization is returned in the results:

```
http://yourserver/oslc/so/WorkTask/
13353622?oslc.properties=*,spi:triCustomerOrgTX{spi:triNameTX}
```

This request produces the following results:

```
{
   spi:triNameTX: "WorkTask"
   spi:triStatusCL: "Draft"
   spi:triIdTX: "1027019"
  -spi:triCustomerOrgTX: {
      spi:triNameTX: "Company 01"
      rdf:about: "http://yourserver/oslc/so/OrganizationRS/12877121"
   }
   rdf:about: "http://yourserver/oslc/so/triWorkTaskRS/13353622"
  -trira:action: [10]
      0:  "triDelete"
      1:  "triInvalidUploadHidden"
      2:  "triIssue"
      3:  "triBaseline"
      4:  "triApplyTemplate"
      5:  "triCopy"
      6:  "triPlanHidden"
      7:  "triSave"
      8:  "triSaveAndClose"
      9:  "triTemporaryTemplate"
  -prefixes: {
      oslc: "http://open-services.net/ns/core#"
      rdf: "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      trira: "http://jazz.net/ns/tririga#"
      dcterms: "http://purl.org/dc/terms/"
   }
}
```

oslc.where parameter

The **`oslc.where`** parameter specifies the WHERE clause for filtering the result set of a query. For example, you want to see a list of work task OSLC resources that were created within a time range and that are approved by management. You can filter by linked resources by using the oslc.where parameter. For example, you might want to filter people according to the name of the manager that the people report to.

The OSLC WHERE clause supports the following basic comparison operators:

Symbol	Description
=	Equality • "*value*" = Equal to

Symbol	Description
	• "%*value*" = Ends with • "*value*%" = Starts with • "%*value*%" = Contains Note that if you manually type the URI with this % symbol as part of the search, you must encode the symbol as follows: "%25*value*"
!=	Inequality
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

If you need to filter for null or not null, you enter the word null after the symbol.

Requirements for the oslc.where parameter

Dates are expressed in ISO 8601 format. For **Date** and **Time** field types, milliseconds show if the field has a nonzero millisecond value.

The OSLC specification supports *and* as the Boolean operator between Boolean expressions. The Boolean operator *or* is not supported. In the following example, the literal value for status is in quotation marks as the status property has a data type of string. The quantity value does not have quotation marks because it is a decimal data type. Integers and Boolean values also do not require quotation marks. For example, `spi:status="Closed" and spi:quantity>10.5 and spi:active=true` where spi:active has a Boolean data type.

The OSLC specification supports *or* implicitly within a single property by using the *in* operator. For example, to see all work tasks that are in either Issued or Active status, use the query `spi:status in ["Issued","Active"]`

Example: Searching for work tasks that were created within a time range and are approved

The following clause lists work task resources that were created within a specific time range:

`spi:status="Approved" and dcterms:created>"2003-07-07T09:50:00-04:00" and dcterms:created<="2004-07-07T09:50:00-04:00"`.

Using the oslc.orderBy parameter to specify the sort order

The **`oslc.orderBy`** parameter defines how the results of a query are ordered. For example, a list of work tasks can be ordered by date or by ID.

To arrange work tasks with the creation date in ascending order and the estimated duration in descending order, use the following **`oslc.orderBy`** parameter: `+dcterms:created,-spi:estimatedDuration`. The + indicates an ascending sort order and the `-` indicates a descending sort order. The values are separated by commas. The following **`oslc.orderBy`** parameter is not valid because there is no default sort order in OSLC query syntax: `dcterms:created,-spi:estimatedDuration`. There must be an explicit + or `-` with the property name. The **`oslc.orderBy`** parameter also supports nested properties, for example, `dcterms:creator{+foaf:name}`. In a real URL, + and `-` do not work. You must use %2B and %2D in the URL instead.

Example: Sorting based on linked resources

You can use the fields of linked resources as the order criteria, to sort the parent records and the linked resources inside of parent records.

For example, there might be two main objects that are named M1 and M2. M1 has two linked resources that are named L1 and L3 and M2 has the linked resources L2 and L4. If you sort the main objects in ascending order, without taking the linked resources into consideration, the results are M1, M2. If you sort the main objects in descending order, the results are M2, M1. However, because there are multiple linked resources for the objects, when you sort in ascending or descending order, the linked resources are also sorted. L1 and L3 are sorted within M1, L2 and L4 are sorted within M2.

oslc.select parameter

The **`oslc.select`** parameter requests a partial resource representation of collection member resources. The **`oslc.select`** parameter always applies to a collection resource. You specify the list of properties to include in the request. The properties that you select can be from the resource itself or from a linked resource.

The **`oslc.select`** parameter provides a comma-separated list of qualified property names. The oslc.prefix parameter is not supported.

Example: Partial resource request

The following request is an example of a partial resource request:

`oslc.select=oslc:shortTitle,dcterms:creator`

Example: Properties from referenced resources

With the **`oslc.select`** parameter, you can select properties from referenced resources. To retrieve information such as the name of the creator, you specify the SELECT statement as

`oslc.select= oslc:shortTitle,dcterms:creator{foaf:name}`.

The foaf:Person resource is the name of the person that is specified in the creator property value. To get all properties from the resource, you can use `oslc.select=*`. The same syntax can be applied to the **`oslc.properties`** parameter when you search for an OSLC resource.

Example: Properties from linked resources

With the **`oslc.select`** parameter, you can select properties from linked resources. To retrieve information such as the value for the name of the customer organization, you specify the SELECT statement as

```
http://yourserver/oslc/spq/WorkTaskQC
?oslc.select=*,spi:triCustomerOrgTX{spi:triNameTX}
&oslc.where=spi:triCustomerOrgTX!="null"
```

This SELECT statement gives the following response:

```
{
  -rdf:members: [1]
     -0:  {
         spi:triNameTX: "WorkTask"
         spi:triStatusCL: "Draft"
         spi:triIdTX: "1027019"
        -spi:triCustomerOrgTX: {
            spi:triNameTX: "Company 01"
            rdf:about: "http://yourserver/oslc/so/Organization/12877121"
         }
         rdf:about: "http://yourserver/oslc/so/WorkTask/13353622"
        -trira:action: [10]
            0:  "triDelete"
            1:  "triInvalidUploadHidden"
            2:  "triIssue"
            3:  "triBaseline"
            4:  "triApplyTemplate"
            5:  "triCopy"
            6:  "triPlanHidden"
            7:  "triSave"
            8:  "triSaveAndClose"
            9:  "triTemporaryTemplate"
      }
   rdf:about: "http://yourserver/oslc/spq/WorkTask"
  -prefixes: {
      oslc: "http://open-services.net/ns/core#"
      rdf: "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      spi: "http://jazz.net/ns/tririga#"
      dcterms: "http://purl.org/dc/terms/"
   }
}
```

oslc.pageSize parameter

The **oslc.pageSize** parameter specifies the number of results the server is to return per page. For example, `oslc.pageSize=20` causes the query to return 20 results per page.

pageno parameter

The **pageno** parameter specifies the page that the server is to return. For example, `pageno=3` causes the query to return only the data for the third page.

Creating OSLC resources

You use the HTTP POST method to create an instance of an OSLC resource. You can then share the resource with other applications, and update the resource by using the PUT, PATCH, or MERGE methods.

About this task

The OSLC client sends a JSON document that conforms to the shape of the resource as published in the shape document. The data is sent to OSLC in the HTTP request body in JSON format and the HTTP header Content-Type is set as the MIME type `application/json`. If the request is processed successfully, the OSLC client receives a location HTTP header with the URI of the newly created resource. If you want to see the properties of the newly created resource, you can include a properties header in the request and indicate which properties you want to see. A header property that is named **Properties** is returned with the specified properties.

When you send a create request, you can include the transaction ID in the HTTP request header. The transaction ID must be unique across all client applications. OSLC saves the transaction ID status in the database when the request is completed. If you submit a request with a transaction ID, OSLC checks whether the transaction exists in the database. If the transaction does exist, OSLC does not run the request. It returns an error that indicates that the request was not completed because the transaction ID exists. The following is an example header with the transaction ID specified:

```
Accept: application/rdf-xml
transactionid: 6001
Content-Type: application/json;charset=utf-8
```

If the request is processed successfully, the OSLC client receives the following response:

```
201 Created
Location: http://yourserver/oslc/so/WorkTask/10269
ETag: 1376596202470
```

If an action is passed in, the action must be a valid action for the current state of the record. For example, if the current state of the record is Draft, Save is often a valid action. The action is called after the record is created.

In the query result page (**oslc/spq**), if **oslc.select** includes a wildcard (*), or in the query details page (**oslc/os**), if no **oslc.properties** parameter is provided, the result includes all actions that can be performed for each record that is retrieved. The form of the action is, "tririga:action":["action1","action2", ...].

OSLC requests can fail for various reasons, such as business validation, authentication, or authorization. For example, the OSLC client might receive a `400 Bad Request` error followed by the HTTP body that contains the details of the error.

Creating records that are associations of primary records

You can create linked dependent resources for primary records.

About this task

You can add associated dependent records to a primary record. The linked resource determines the association and the associated resource.

Example: Creating a purchase order with two purchase order line items

The following method creates a purchase order resource with two purchase order line item resources. The name of the linked resource is HasPOLineItem.

```
{
  "spi:action":"Create Draft (triCreateDraft)",
  "spi:triNameTX":"oslcPO",
  "trirldr:HasPOLineItem":
    [
      {
        "spi:action":"Create (triCreate)",
        "spi:triNameTX":"POLineItem1"
      },
      {
        "spi:action":"Create (triCreate)",
        "spi:triNameTX":"POLineItem2"
      }
    ]
}
```

If the request is processed successfully, the OSLC client receives the following response:

```
201 Created
Location: http://yourserver/oslc/so/PO/10269
ETag: 1376596202470
```

Updating OSLC resources

You can use the HTTP PUT method to replace an OSLC resource and the HTTP POST with a PATCH override to partially update an OSLC resource.

An HTTP PUT completely replaces the data in the resource with the properties in the request.

An HTTP POST with an x-method-override of PATCH replaces a local resource property with the content in the request.

An HTTP POST with an x-method-override of PATCH and a PATCHTYPE of MERGE finds and matches the local resource elements from the request with the elements on the server. Depending on whether a match was found, the local resource elements are updated or inserted. A local element is never deleted from the local resource property.

When you send an update request, you can include the transaction ID in the HTTP request header. The transaction ID must be unique across all client applications. OSLC saves the transaction ID status in the database when the request is completed. If you submit a request with a transaction ID, OSLC checks whether the transaction exists in the database. If the transaction does exist, OSLC does not run the request. It returns an error that indicates that the request was not completed because the transaction ID exists. The following is an example header with the transaction ID specified:

```
Accept: application/rdf-xml
transactionid: 6001
Content-Type: application/json;charset=utf-8
```

HTTP PUT method

The HTTP PUT operation is used for a full replacement of an OSLC resource. The PUT method updates both literal properties and local resource properties, and it deletes any local resource properties that are not included in the request.

The following rules apply when you use the PUT method to replace an OSLC resource:

- All literal properties that are specified in the request document are updated. Any literal property that is not specified as part of the request is not impacted explicitly. However, it can be implicitly impacted by the business logic that is attached to the resource. This rule is the same as when you use the PATCH method to update a resource.
- All local resource properties are replaced by corresponding property values from the request. If the resource property is not included in the request, the corresponding resource is deleted. If the resource property is included, its value replaces the value in the server.
- Reference resources cannot be updated explicitly. However, you can update properties that refer to the resource, and the properties follow the update model of literal properties. This rule is the same as when you use the PATCH method to update a resource.

In the following examples, the work task resource has one literal property, **`taskname`**, and one resource property **`parts`**. The **`parts`** property points to the local resource parts and is associated with two parts records. If a PUT request contains the **`taskname`** property and no **`parts`** property, the task name is updated and the parts data is deleted.

Example: Updating a literal property

The following method updates the literal property, **`taskname`**:

```
PUT http://yourserver/oslc/so/WorkTask/123

{
   "dcterms:taskname": "Check-out Leaking - Modified for Test"
}
```

If the request is processed successfully, the OSLC client receives the following response:

```
204 No Content
ETag: 1376596202470
```

The task name is changed to `Check-out Leaking - Modified for Test`. Because the **`parts`** data was not included in the method, the parts records are deleted.

Example: Updating a local resource property

The following method updates the resource property, **`parts`**:

```
PUT http://yourserver/oslc/so/WorkTask/123

{
   “spi:parts”: [
    {
         “spi:partsid": "0000000067",
 "spi:quantity": 5
            }
]
}
```

If the request is processed successfully, the OSLC client receives the following response:

```
204 No Content
ETag: 1376596202470
```

A search is made for a parts record with the ID `0000000067`. If such a parts record exists, it is updated. If no match is found, a new parts record is created. All other parts data for this work task resource is deleted. Because the **taskname** property is not included in the method, the task name is not part of the request and the value is unaffected.

HTTP PATCH method

PATCH is used for a partial update of an OSLC resource. A PATCH does not delete any local resource properties that are not included in the request. The PATCH request is sent by a POST method with the x-method-override header set to PATCH.

The following rules apply when you use a PATCH to replace an OSLC resource:

- All literal properties that are specified in the request document are updated. Any literal property that is not specified as part of the request is not impacted explicitly. However, it can be implicitly impacted by the business logic that is attached to the resource. This rule is the same as when you use the PUT method to replace a resource.
- All local resource properties are updated or replaced by corresponding property values from the request. If the resource property is not included in the request, the corresponding local resource is not explicitly impacted. If the resource property is included, its value replaces or updates the value in the server. Other resource properties are deleted by a PATCH and not deleted by a MERGE.
- Reference resources cannot be updated explicitly. However, you can update properties that refer to the resource, and the properties follow the update model of literal properties. This rule is the same as when you use the PUT method to replace a resource.

Example: Updating a literal property

The following method updates the task name property of the work task:

```
POST http://yourserver/oslc/so/WorkTask/123
x-method-override: PATCH

{
   "dcterms:taskname": "Check-out Leaking – Modified for Test"
}
```

Unlike the PUT method, this PATCH method does not update other properties of the work task.

Example: Updating a local resource property

The following method updates a specified parts record and deletes other data:

```
POST http://yourserver/oslc/so/WorkTask/123
x-method-override: PATCH

{
   "dcterms:taskname": "Check-out Leaking – Modified for Test",
    “spi:parts”: [
    {
         “spi:partsid": "0000000067",
 "spi:quantity": 5
            }
]
}
```

This method behaves similarly to the PUT method. The system searches for a parts record with the ID 0000000067. If such a parts record exists, it is updated. If no match is found, a new parts record is created. All other parts records for this work task resource are deleted.

Example: Updating and merging a local resource property

The following method updates the resource with the PATCHTYPE header set to MERGE:

```
POST http://yourserver/oslc/so/WorkTask/123
x-method-override: PATCH
PATCHTYPE: MERGE

{
   "dcterms:taskname": "Check-out Leaking - Modified for Test",
    “spi:parts”: [
    {
         “spi:partsid": "0000000067",
 "spi:quantity": 5
            }
]
}
```

A search is made for a parts record with the ID `0000000067`. If such a parts record exists, it is updated. If no match is found, a new parts record is created. Because the PATCHTYPE header is set to `MERGE`, the other parts records for this work task resource are left intact.

Example: Making a conditional update

The following method updates the resource if the ETag value is `1376596202470`:

```
POST http://yourserver/oslc/so/WorkTask/123
x-method-override: PATCH
if-match: 1376596202470
```

If the ETag value is `1376596202470`, the work task resource is updated and an `HTTP 204` message is sent.

If the ETag value is not `1376596202470`, the server responds with an `HTTP 412 Precondition failed` message. This message implies that the resource was updated by some other process and the requesting client has a stale copy of the resource. The client must perform a GET method on the **123** resource to get a fresh copy of the resource.

The following table summarizes the result of each update method when applied to different types of resources.

Method	Literal properties	Local resources	Reference resources
PUT	If omit a property, the property is not affected.	If omit a property, the property is deleted.	If omit a property, the property is not affected.
PATCH	If omit a property, the property is not affected.	If omit a property, the property is not affected.	If omit a property, the property is not affected.
MERGE	If omit a property, the property is not affected.	If omit a property, the property is not affected.	If omit a property, the property is not affected.

Deleting OSLC resources

You use the HTTP DELETE method to delete an OSLC resource.

Write the HTTP DELETE on the URI of the resource. If the resource business object has a state transition from the current state of the resource to a null state, the state of the object changes to null.

If the resource business object does not have a state transition from the current state of the resource to a null state, the state of the object does not change. You can modify the business object to include a state transition from the current state to null. Or you can update the record with an HTTP PUT or HTTP PATCH instead of an HTTP DELETE and pass the action name to trigger the deletion.

If a resource has child records, the child records are deleted when the resource is deleted. If a resource has associated records, the association is deleted when the resource is deleted, but the associated records are not affected.

Working with attachments and binary data

OSLC clients can retrieve, create, and update attachments and binary data through the OSLC API.

OSLC attachment processing

Attachment processing in OSLC involves processing the attachment itself, which is an unstructured document, and processing the associated metadata for that document.

The metadata is described in an AttachmentDescriptor resource RDF. AttachmentDescriptor resources are always associated with an attachment in a 1:1 relationship.

Creating OSLC attachments

You create attachments by using the HTTP POST method with binary content. Do not use multi-part HTTP POST requests.

The following example shows an HTTP request to create an attachment to show an image of a broken part:

```
POST http://yourserver/oslc/os/oslcwodetail/_abcd123/attachments
Slug: brokenpart.jpeg
Content-Type: image/jpeg
Content-Length: 18124
x-document-description: A broken part
x-document-meta: Attachment

[binary content]
```

The following example shows the response to the request:

```
HTTP/1.1 201 CREATED
Location: http://yourserver/oslc/os/oslcwodetail/_abcd123/attachments/1
Link: <http://yourserver/oslc/os/oslcwodetail/_abcd123/attachments/meta/1>;
rel="describes"
Content-Length: 0
```

The Slug header indicates the file name. You can use the header x-document-description to describe the attachment. This description is mapped to dcterms:description of the attachment descriptor resource. You can use the x-document-meta header to indicate the folder name for storing the attachment.

Updating OSLC attachments

You can update OSLC attachments by using the HTTP PUT method with binary content. Do not use multipart HTTP PUT requests.

The following example shows HTTP request to update an attachment:

```
PUT http://yourserver/oslc/os/oslcwodetail/_abcd123/attachments/1
Slug: brokenpart2.jpeg
Content-Type: image/jpeg
Content-Length: 18124
x-document-description: A broken part
x-document-meta: Attachment

[binary content]
```

The following example shows the response to the request:

```
HTTP/1.1 204
Content-Length: 0
```

To update just the description of the attachment, you can use a PATCH request to the meta URI, as shown in the following example:

```
PATCH http://yourserver/oslc/os/oslcwodetail/_abcd123/attachments/meta/1
Content-Type: application/json
```

```
{
    “dcterms:description”: “Broken pipe”
}
```

Selecting OSLC attachments

Attachments typically are related resources to structured resources such as work orders or assets. Attachments also can be associated with child resources. When a structured resource is fetched, only the link to its related attachment collection is provided by default. If the consumer expects the attachment details to be in lined as part of the owning structured resource, use the following query format:

```
/oslc/os/oslcwodetail?oslc.select=res1,res2,spi:attachments{*}
```

Deleting OSLC attachments

To delete OSLC attachments, use the following request:

```
DELETE <attachment uri>
```

OSLC binary data processing

OSLC clients can query and update binary data by using the OSLC API.

OSLC supports the following two types of binary data:

- TRIRIGA binary data is stored in the dm_content table. Each data element has a unique content ID, a file name, and a MIME type for data rendering. The binary field contains the content ID.
- Image data is stored as image files. The image field contains a partial path to the image file.

Querying binary data

When binary data is retrieved through an OSLC query or record details page, the values are presented with an URI. The following example shows a URI with the binary data information:

```
http://yourserver/oslc/so/supApp/168867/tririga:supBinary
```

The client can use the URI to retrieve the actual data. In the HTTP response of the URI, the body contains the content that is read from the content field of the dm_content table. The Content-Type header contains the MIME type of the binary data.

Similarly, when image data is retrieved through an OSLC query or record details page, the values are presented with an URI. The following example shows a URI with the image information

```
http://yourserver/oslc/so/supApp/168867/tririga:supImage
```

The client can use the URI to retrieve the actual image. In the HTTP response of the URI, the body contains the image that is read from the image file. The Content-Type header is a MIME type that is composed of the string `image/` plus the extension of the image file.

```

```

Updating binary data

You use a separate HTTP request to update each binary or image property. The following example shows the format of a request to update binary or image data:

```
http://localhost:8001/oslc/so/soID/property
```

In the example, soID is the ID of the smart object that the binary or image property belongs to. The property identifies the binary or image property that the value is set to. It takes the regular property form of prefix:sectionName-propertyName, where sectionName is optional.

The HTTP PUT method is used, and the content-type of the HTTP header contains the MIME type of the data, as follows:

- For binary properties, the MIME type that is passed in from the content-type header is carried over to the MIME type of the data. You can pass in a file name through the optional Slug header.
- For image properties, the content-type must be image type. The type is the format of the image, such as PNG or JPG. The file name is generated and the image type is used as the file extension.

Chapter 4. Administering OSLC resources

Authentication and authorization support is provided for OSLC services by IBM TRIRIGA security. You use OSLC logging to debug and to evaluate performance.

OSLC security

Authentication and authorization support for OSLC services is provided by IBM TRIRIGA security.

Native authentication

The consumer request can provide the *user:password* values that are base64 encoded and are in the OSLC HTTP header property.

Explicit login and logout

If the consumer application needs to run explicit login commands, you use the following request:

```
GET http://yourserver/oslc/login?USERNAME=username&PASSWORD=password
```

If the consumer application needs to run explicit logout commands, you use the following request:

```
GET http://yourserver/oslc/logout
```

Authorization

Authorization control is provided at the business object level of the resource. The security processing of the resource data is then based on both the configuration of security of the application and the user group of the user who made the request. When OSLC resources are processed, any object attribute that is configured as hidden through security is not included in the response to an OSLC request.

Password changes

In order for a client user to be able to change a password, you must create a MyProfile OSLC resource that is based on the IBM TRIRIGA My Profile business object.

To support password changes, you must create the MyProfile resource with at least the **`Password`** property defined. You can define other properties to support other profile changes as necessary.

You gain access to the MyProfile resource in OSLC by using the following URI:

```
http://yourserver/oslc/so/MyProfile/userId
```

This URI always returns the current user's profile, regardless of what user ID is entered, since a user is not allowed to view the profiles of other users. You can define query capabilities for the MyProfile resource, but the query results do not return more than the profile of the current user. You cannot define creation factories the MyProfile resource.

You cannot create or delete MyProfile objects through OSLC from the MyProfile resource.

Use the PATCH method along with the URI to change the profile. Include the following JSON string in the request to change the password:

```
{..."spi:Password":"password",...}
```

The password is in plain text form and is encrypted internally.

Use the HTTP POST method, along with the following headers, in password change requests:

```
- x-method-override: PATCH
- PATCHTYPE: CHPWD
```

The ID of the resource that represents My Profile is set in `tririgaweb.properties` file, as follows:

`OSLC_MYPROFILE_RESOURCE=MyProfile`

Expired passwords

If a password is expired, the consumer application cannot access any OSLC resource other than the password change URI or the logout URI.

An error message with HTTP status 403 Forbidden is returned. If the system property OSLC_MYPROFILE_RESOURCE is defined, the JSON error message that is returned includes a URI that guides the consumer application to the password change request URI.

Password expiry rules are set in TRIRIGA at `ToolsSystem SetupSystemPassword Setup`.

Example: Expired password response

```
{
oslc:Error:
{
spi:user:
{
rdf:resource: "http://yourserver/oslc/so/MyProfile/13417792"
}
oslc:message: "Password Expired"
oslc:statusCode: 403
oslc:extendedError: "OSLC0054"
}

}
```

OSLC logging

Logs can record information that can be useful when you debug or evaluate performance.

Logging is managed in the IBM TRIRIGA Administrator Console. After you log in, select **Platform Logging** > **OSLC**. To turn off logging, clear the **OSLC** check box. For more information, see the *IBM TRIRIGA Application Platform 3 Administrator Console User Guide*.

Chapter 5. Troubleshooting OSLC

The following tips can help you troubleshoot issues when you are an OSLC consumer of IBM TRIRIGA.

Table 1. Tips for troubleshooting OSLC.

Concern	Remedy
The resource shape no longer works after you rename the IBM TRIRIGA report	If you rename the IBM TRIRIGA report that is defined in a resource shape, it breaks the resource shape because the defined report no longer exists. You see an error message when you preview the query capability or use the creation factory. The general practice is to update the resource shape when the name of the report changes. Also, if either the module or the business object changes in the IBM TRIRIGA report, that value must be updated in the resource shape as well. The Where Used tab in the report identifies which resource shape is using the report.
Need logs to debug the OSLC implementation	Logging is managed in the IBM TRIRIGA Administrator Console. After you log in, select **Platform Logging** > **OSLC**. To turn off logging, clear the **OSLC** check box. For more information, see the *IBM TRIRIGA Application Platform 3 Administrator Console User Guide*.
Stack trace is thrown in olsc.where	If you put single quotation marks in an oslc.where parameter, you create an invalid URL. The exception comes from an Apache filter that occurs before the oslc.where reaches the OSLC servlet. It is not within IBM TRIRIGA control.

OSLC explanations for HTTP codes

OSLC uses standard HTTP response codes as error messages. For example, the HTTP 404 response is normally returned when a web page is not found, and in OSLC the 404 response code is returned when the resource cannot be found.

Some existing error codes are mapped to HTTP codes by default, but you can map extra codes as required.

The following HTTP response codes are implemented by OSLC:

HTTP code	OSLC explanation
200	Success
201	Success. The response contains a link.
204	Resource successfully updated. There is no response entity.
400	Error handling request. This error might be due to the request content or URI. For example, there might be a business logic validation error on the server side.

HTTP code	OSLC explanation
401	Authentication failure.
403	Forbidden. The user password expired.
404	Resource cannot be found or an invalid resource type was provided.
405	HTTP method cannot be used for the resource.
406	Requested representation is not supported.
410	Stable resource page expired.
412	Resource on the client side is stale and must be refreshed from the server. The conditional update failed because the resource was updated by another user or process.
500	All other server errors.

The messages support the languages that are supported by IBM TRIRIGA.

Notices

Trademarks

Terms and conditions for product documentation

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at https://www.ibm.com/privacy/details/us/en/ in the section entitled “Cookies, Web Beacons and Other Technologies.”

Part Number:

(1P) P/N: