ONMS ONMSi 6.00.2.0

Back to the WS page

ONMSi REST Web Service API

Table of content

Download examples source files

1. REST Web Services

This new API is emerging in the industry because of its lightweight and ease of implementation.

One advantage of REST API is that it is accessible from the simplest web client. There is no need for any language dependent third party library. But you can use some to make it easier, such as taking care of data serialization and work with java objects instead of manually parsing or creating XML.

1.1. Authentication

ONMSi REST API supports 2 different authentication systems:

- **Basic authentication**: the client needs to provide a username and password when making a request to the API.
- **mTLS authentication**: use of TLS certificates to communicate and make API requests through HTTPS procotol.

Please read the API Authentication to understand and configure your API authentication system.

> **WARNING**: if you experience connection issues, please read the network considerations.

1.2. Main resources

The main resources exposed by the ONMSi REST API are:

Resource	URL
Alarms	http://myonmsi/rs/alarms
Cable heads	http://myonmsi/rs/cableHeads
Central Offices	http://myonmsi/rs/centralOffices
Events	http://myonmsi/rs/events
Homes	http://myonmsi/rs/homes
Links	http://myonmsi/rs/links
Monitoring Tests	http://myonmsi/rs/monitoringTests
OTUs	http://myonmsi/rs/otus
PONs	http://myonmsi/rs/pons
PON test points	http://myonmsi/rs/pon-test-points

1.3. Simple API call

Following code snippet demonstrates how to make an authenticated call to the API to retrieve events using this method:

```
POST /events
```

Notice however that a more flexible way of doing the same thing is described later in this page.

```
    public void basicEventRetrieval() throws IOException
    {
```

```
        // create the HTTP client
        HttpClient client = new HttpClient();

        // create the method to retrieve the events
        String hostName = InetAddress.getLocalHost().getHostName();
        HttpMethod method = new PostMethod( String.format( "http://%s/rs/events", hostName ) );

        // setup basic authorization for user 'jdoe' with password 'password123'
        String authorization = "Basic " + new String( Base64.encodeBase64( "jdoe:password123".getBytes() ) );
        method.addRequestHeader( "Authorization", authorization );

        // execute the method
        client.executeMethod( method );

        // read the response
        byte[] response = method.getResponseBody();
        System.out.println( new String( response ) );
    }
```

1.4. Event types, API users and alarm event filtering

The **events** resource has one method to retrieve newly available events.

```
POST /events
```

Because getting the events follows the long delay polling strategy, calling the method may last up to 30 seconds (configurable). You will have to deal with this in the architecture of the client application, typically by using several threads.

The events that can be received can be of different kinds:

- `SequentialAlarmEvent` objects are sent for each change in the lifecycle of an alarm (creation, acknowledgment, clearing, commenting...) and hold a sequence number so that clients can detect missing events easily.
- `OperationEvent` objects are sent in response to a request to an operation triggered from the API (HomeTestOnDemandEvent, PonTestOnDemandEvent or MonitoringTestOnDemandEvent)

1.4.1. Operation events

Operation events are neither filtered nor persisted by ONMSi. Therefore, a client application which is stopped and restarted will not receive the events that occurred while it was not running.

Client applications also need to ignore operation event they receive that do not match the operation ID they triggered.

1.4.2. Sequential alarm events

In the contrary, ONMSi both filters and persists the **sequential alarm events** as they occur over time. Filtering is done per user according to rules specified for each declared API-user in the system settings of ONMSi.

A sequential alarm event is produced for every alarm event that passes through the filter defined for the user authenticated in the API client.

In order to prevent unnecessary database size explosion, sequential alarm events are not persisted for all ONMSi users, but only for notified users registered in the API system settings of ONMSi. **Make sure that you authenticate on the API using a registered API-user, or you will never receive any sequential alarm event, hence no alarm event**.

The filter defined for each user is not only configurable but also highly additionalizable because it is written in Python (Jython actually) which is a popular, powerful (yet simple) scripting language that is embedded inside the ONMSi server.

1.4.3. Getting again lost alarm events

The **alarms** resource also provides two mechanisms to get again alarm events that were lost in the client:

- `resendAlarmEvents(long startingSequenceNumber)` requests the server to send again all sequential alarm events starting with the specified sequence number. The sending only occurs for the calling user, and only holds events that passed the filter defined at the time when the event occurred.
- `resynchronize()` requests the server to send again all alarm events for the currently active alarms for the logged user.

Here is a way to get again all the alarm events starting with the sequence number `1234567890123L`:

```
        long sequenceNumber = 1234567890123L;

        // Have the server clear its "sent" flag on alarm events starting from a sequenceNumber

        // Set the proper MIME type headers
        helper.setContentTypeHeader( "text/plain" );
        helper.post( "/alarms/resendAlarmEvents", String.valueOf( sequenceNumber ) );
        helper.setContentTypeHeader( "application/xml" );

        // Get events, including the resent alarm events
        String eventsXml = helper.post( "/events" );
        Events events = OnmsiJaxbHelper.convertXmlToObject( eventsXml, Events.class );
```

Following call will reset the sequence counter and send again all alarm events for all active alarms (i.e. not cleared).

```
        // Have the server rebuild the notification events for the user
        helper.post( "/alarms/resynchronize" );

        // Get events, including the resent alarm events
        eventsXml = helper.post( "/events" );
        events = OnmsiJaxbHelper.convertXmlToObject( eventsXml, Events.class );
```

1.5. Finding objects

Some resources serve a particular type of business object: for instance, `/homes` allows to get, find and update `Home` objects.

All those resources provide some common methods:

- `GET /myResources?attribute=myAttributeName&value=myValue` will find the objects of type *myResources* with the attribute named *myAttributeName* holding a value of *myValue*. Attributes that can be used for such query are tagged as findable attributes.
- `GET /myResources/attributes` returns a list of attributes for the object of type *myResources*, including their name, whether they can be used to find objects, whether they are modifiable and whether the attribute is additional or standard.

1.6. Additional Attributes

The API is designed to allow for any number of additional attributes defined by the customer.
In order to avoid clashes between ONMSi attribute names and additional attribute names, the latter are always prefixed with an underscore, like `_externalKey`

2. Running repetitive tasks without pain

This section shows how common use cases can be implemented in Java. The reader is expected to already understand the ONMSi API principles.

In order to simplify data management, we will use the OnmsiRsApiHelper class for REST calls, and the JABXUtils class to build and parse XML from/to DTO objects.

DTOs may be created by clients, as long as they match the exchange specifications. But it's safer to use classes generated from topaz-api.xsd.

2.7. Simple API call - revised

The OnmsiRsApiHelper class embeds an HTTP client and takes care of authentication.

Please note that the full class is available for download.

```
public final class OnmsiRsApiHelper
{
    /** The HTTP client */
    private final HttpClient m_httpClient = new HttpClient( new MultiThreadedHttpConnectionManager() );
    /** The REST server host name */
    private final String m_hostname;
    /** The authorization string */
    private final String m_authorization;
    /** Flag to use HTTPS */
    private final String m_protocol;
    /** Set in the content-type HTTP header for POST and PUT requests */
    private String m_contentTypeHeader;
    /** Set in the accept HTTP header for GET and POST requests */
    private String m_acceptHeader;

    public OnmsiRsApiHelper( String hostname, String user, String password, String protocol )
    {
        m_hostname = hostname;
        m_authorization = "Basic " + new String( Base64.encodeBase64( String.format( "%s:%s", user, password ).getBytes() ) );
        m_protocol = protocol;
    }

}
```

Standard HTTP methods GET, PUT, POST are available with different parameters.

```
    public String get( String path )
    public String get( String path, NameValuePair... requestParams )

    public String post( String path )
    public String post( String path, NameValuePair... requestParams )
    public String post( String path, String body )
    public String post( String path, String body, NameValuePair... requestParams )

    public String put( String path, String body )
    public String put( String path, String body, NameValuePair... requestParams )
    public String delete( String path )
    public String delete( String path, NameValuePair... requestParams )
```

2.8. XML management

The OnmsiJaxbHelper class takes care of XML manipulation, and converts it to DTO objects, or creates it from DTO objects:

```
public final class OnmsiJaxbHelper
{
    public static <T> T convertXmlToObject( String xml, Class<T> clazz )
    public static <T> String convertObjectToXml( T instance, Class<T> clazz )
}
```

2.9. Waiting for a specific event

The event polling can become quite redundant as well if not designed properly. Here is one way of dealing with it. In your client, you will likely have to adapt this pattern by creating a dedicated thread for events and using a listener mechanism to register specific handlers to determined events.

We first create a class which helps specifying what events we are interested in. This is done with an event matcher class shown below.

Again, note that the full class is available for download.

```
import com.topaz.api.rs.Event;

public abstract class OnmsiRsEventMatcher< T extends Event >
{
    private final Class< T > m_eventClass;

    public OnmsiRsEventMatcher( Class< T > eventClass )
    {
        m_eventClass = eventClass;
    }

    public Class< T > getEventClass()
    {
        return m_eventClass;
    }

    public abstract boolean match( T event );
}
```

Using this base class, specifying what events we want to listen to is done very easily by creating an anonymous event matcher with the expected event type and only writing a matching condition, as shown in the example below:

```
        OnmsiRsEventMatcher< PonTestOnDemandEvent > matcher = new OnmsiRsEventMatcher< PonTestOnDemandEvent >( PonTestOnDemandEvent.class )
        {
            private final EnumSet< Outcome > m_endOfTestValues = EnumSet.of( Outcome.FAILURE, Outcome.SUCCESS );

            @Override
            public boolean match( PonTestOnDemandEvent event )
            {
                // compare to the 'final' operation ID stored above
```

```
            return (event.getOperationId() == operationId) && m_endOfTestValues.contains( event.getOutcome() );
        }
    };
    OnmsiRsEventMatcher< PonCalibrationEvent > matcher = new OnmsiRsEventMatcher< PonCalibrationEvent >( PonCalibrationEvent.class )
    {
        private final EnumSet< Outcome > m_endOfTestValues = EnumSet.of( Outcome.FAILURE, Outcome.SUCCESS );

        @Override
        public boolean match( PonCalibrationEvent event )
        {
            // compare to the 'final' operation ID stored above
            return (event.getOperationId() == operationId) && m_endOfTestValues.contains( event.getOutcome() );
        }
    };
```

Once this event matching mechanism is put in place properly, we just need to add the following `waitForEvent()` function to the `OnmsiRsApiHelper` class created earlier.

```
public < T extends Event > T waitForEvent( OnmsiRsEventMatcher< T > matcher, int timeoutSec )
    throws TimeoutException, IOException, JAXBException
{
    long timeoutMs = timeoutSec * 1000L;
    long started = System.currentTimeMillis();
    while ( (System.currentTimeMillis() - started) < timeoutMs )
    {
        // retrieve events
        String eventsXml = post( "/events" );
        Events events = OnmsiJaxbHelper.convertXmlToObject( eventsXml, Events.class );

        for ( Event event : events.getEvent() )
        {
            if ( matcher.getEventClass().isAssignableFrom( event.getClass() ) )
            {
                T typedEvent = matcher.getEventClass().cast( event );
                boolean matches = matcher.match( typedEvent );
                if ( matches )
                {
                    return typedEvent;
                }
            }
        }
    }
    throw new TimeoutException( "No event" );
}

public String getContentTypeHeader()
{
    return m_contentTypeHeader;
}

public void setContentTypeHeader( String contentTypeHeader )
{
    m_contentTypeHeader = contentTypeHeader;
}

public String getAcceptHeader()
{
    return m_acceptHeader;
}

public void setAcceptHeader( String acceptHeader )
{
    m_acceptHeader = acceptHeader;
}
```

3. Reference guide

The data used by the API are formatted in XML and defined by topaz-api.xsd.
You should easily find tools to process this file and generate objects suited to your favorite programming language.

3.10. OTUs

Method	Path	Query Parameters	Content	Returned Value	Description
GET	/otus	**attribute**=*attributeName* **value**=*matchingValue*		A contained collection of Otus	Retrieves the named *attribu* *matchingValue* Findable attrib are: name, ipA centralOfficeIn additional attr
GET	/otus/all	**offset**=*result offset (for pagination)* **limit**=*the max results count (may be forced to a lower value than requested)*		An EntityList	Retrieves all th in a paginated minimal data a
POST	/otus /otus/in-domain		A OtusInDomain		Imports OTUs (create or upd **This operatio "Upload Data permission**
GET	/otus/**{internalKey}**			An Otu	Retrieves the key of **interna**
GET	/otus/**{internalKey}**/ports	**attribute**=*attributeName (optional)* **value**=*matchingValue (optional)*		A contained collection of Ports	Retrieves the an internal ke and, if provide named *attribu* *matchingValue* Findable attrib position
PUT	/otus/**{internalKey}**		A Otu		Updates an ex only updatable updated (nam additionnal att
PUT	/otus/**{internalKey}**/ports/**{portInternalKey}**		A single Port		Update the Po attribute for th the Port.**inter** that path's **int portInternall**

GET	/otus/attributes			A contained collection of Attributes	Retrieves the a resource.
POST	/otus/**{internalKey}**/ports/**{portInternalKey}**/lightsource-measure		A LightsourceMeasureParameters	An operation ID	Start a lightso port. The resu LightsourceMe

3.11. Links

Method	Path	Query Parameters	Content	Returned Value
GET	/links	**attribute**=*attributeName* **value**=*matchingValue*		A contained collectic Links
GET	/links/all	**offset**=*result offset (for pagination)* **limit**=*the max results count (may be forced to a lower value than requested)*		An EntityList
POST	/links		A contained collection of Links	
GET	/links/**{internalKey}**			A Link
PUT	/links/**{internalKey}**		A Link	
GET	/links/**{internalKey}**/monitored			A boolean
GET	/links/attributes			A contained collectic Attributes
GET	/links/link-types			A contained collectic LinkTypes
POST	/links/**{internalKey}**/measurements/start		A MeasurementOnLinkParameters to define the start date and/or period of the measurement	A MeasurementOnLink containing the meas **UID** that can be use MeasurementOnLink stop the measureme
POST	/links/measurements/**{measurementUid}**/stop			
GET	/links/**{internalKey}**/measurements	**history**=*boolean, if true return several measurements*		A contained collectic MeasurementOnLink
GET	/links/measurements/**{measurementOnLinkInternalKey}**/trace			A trace file
GET	/links/measurements/**{measurementOnLinkInternalKey}**/cdm			A JSON file

3.12. Monitoring Tests

Method	Path	Query Parameters	Content	Returned Value	Description
GET	/monitoringTests	**attribute**=*attributeName* **value**=*matchingValue*		A contained collection of MonitoringTests	Retrieves the monitoring tests whose **attribute** named *attributeName* has a **value** of *matchingValue* Findable attribute names for monitoring tests are: name, linkInternalKey and any additional attribute name.
GET	/monitoringTests/**{internalKey}**			A MonitoringTest	Retrieves the monitoring test with an internal key of **internalKey**
POST	/monitoringTests/**{internalKey}**/testOnDemand			An operation ID	Triggers a test on demand for the monitoring test with an internal key of **internalKey**. The result will be received as a MonitoringTestOnDemandEvent.
GET	/monitoringTests/attributes			A contained collection of Attributes	Retrieves the attributes of the monitoring test resource.

3.13. Central Offices

Method	Path	Query Parameters	Content	Returned Value	Description
GET	/centralOffices	**attribute**=*attributeName* **value**=*matchingValue*		A contained collection of CentralOffices	Retrieves the central offices whose **attribute** named *attributeName* has a **value** of *matchingValue* Findable attribute names for

					central offices are: name and any additional attribute name.
GET	/centralOffices/all	**offset**=*result offset (for pagination)* **limit**=*the max results count (may be forced to a lower value than requested)*		An EntityList	Retrieves all the central offices in the system in a paginated manner, returning minimal data about each of them.
POST	/centralOffices		A CentralOffices		Imports central offices in the default domain (create or update) **This operation requires the "Upload Data" system permission**
POST	/centralOffices/in-domain		A CentralOfficesInDomain		Imports central offices in a specific domain (create or update) **This operation requires the "Upload Data" system permission**
GET	/centralOffices/**{internalKey}**			A CentralOffice	Retrieves the central office with an internal key of **internalKey**
GET	/centralOffices/attributes			A contained collection of Attributes	Retrieves the attributes of the central office resource.
DELETE	/centralOffices/**{internalKey}**			An operation ID	Trigger a delete on the central office with an internal key of **internalKey**. The outcome will be received as a ObjectRemovalEvent

3.14. Cable Heads

Method	Path	Query Parameters	Content	Returned Value	Description
GET	/cableHeads	**attribute**=*attributeName* **value**=*matchingValue*		A contained collection of CableHeads	Retrieves the cable heads whose **attribute** named *attributeName* has a **value** of *matchingValue* Findable attribute names for cable heads are: name, centralOfficeInternalKey and any additional attribute name.
GET	/cableHeads/all	**offset**=*result offset (for pagination)* **limit**=*the max results count (may be forced to a lower value than requested)*		An EntityList	Retrieves all the cable heads in the system in a paginated manner, returning minimal data about each of them.
POST	/cableHeads		A contained collection of CableHeads		Imports cable heads (create or update) **This operation requires the "Upload Data" system permission**
GET	/cableHeads/**{internalKey}**			A CableHead	Retrieves the cable head with an internal key of **internalKey**
DELETE	/cableHeads/**{internalKey}**	**deleteAssociatedLinks**=*boolean, if true deletes the associated links*		An operation ID	Trigger a delete on the cable head with an internal key of **internalKey**. The outcome will be received as a ObjectRemovalEvent
GET	/cableHeads/attributes			A contained collection of Attributes	Retrieves the attributes of the cable head resource.
POST	/cableHeads/**{internalKey}**/startCableRouteScan			An operation ID	Triggers a start for a scan on the cable route with an internal key of **internalKey** (a cable head declared to be used as a cable route). The result will be received as a CableRouteScanOnDemandEvent.
POST	/cableHeads/**{internalKey}**/enableMeasure	**attribute**=*value* **value**=*true* or *false*			Enable or disable the measure on the links of the cable head, specified with an internal key of **internalKey**.

3.15. PONs

Method	Path	Query Parameters	Content	Returned Value	Description
GET	/pons	**attribute**=*attributeName* **value**=*matchingValue*		A contained collection of Pons	Retrieves the PONs whose **attribute** named *attributeName* has a **value** of *matchingValue* Findable attribute names for PONs are: name, centralOfficeInternalKey and any additional attribute name.
GET	/pons/all	**offset**=*result offset (for pagination)* **limit**=*the max results count (may be forced to a lower value than requested)*		An EntityList	Retrieves all the PONs in the system in a paginated manner, returning minimal data about each of them.
POST	/pons		A contained collection of Pons		Imports PONs (create or update) **This operation requires the "Upload Data" system permission**

GET	/pons/**{internalKey}**		A Pon	Retrieves the PON with an internal key of **internalKey**
DELETE	/pons/**{internalKey}**		An operation ID	Trigger a delete on the PON with an internal key of **internalKey**. The outcome will be received as a ObjectRemovalEvent
POST	/pons/**{internalKey}**/calibration	A PonCalibrationConfig	An operation ID	Triggers a calibration on the PON with an internal key of **internalKey**. The outcome will be received as a PonCalibrationEvent
GET	/pons/**{internalKey}**/calibration		A PonCalibrationResult	Get the calibration result of the PON with an internal key of **internalKey**.
GET	/pons/**{internalKey}**/testHistory		A contained collection of PonTestHistorys	Retrieves the test history for the PON with an internal key of **internalKey**. The result is a collection of PonTestHistory containing a PON test internal key that can be used for pon test result data retrieval with `POST /pons/testResult`.
POST	/pons/**{internalKey}**/testOnDemand		An operation ID	Triggers a test on demand for the PON with an internal key of **internalKey**. The result will be received as a PonTestOnDemandEvent.
GET	/pons/attributes		A contained collection of Attributes	Retrieves the attributes of the PON resource.
GET	/pons/testResult/**{ponTestInternalKey}**		A PonTestResult	Retrieves the PON test result matching the PON test internal key.
POST	/pons/testResult/**{ponTestInternalKey}**/trace		A OtdrTraceDownloadKey	Allows downloading an OTDR trace for a PON test result.
GET	/pons/peaks/**{peakInternalKey}**		A Peak	Retrieves a PON peak given its internal key
PUT	/pons/peaks/**{peakInternalKey}**	A Peak		Updates a PON peak
GET	/pons/peaks/attributes		A contained collection of Attributes	Retrieves the attributes of the PON peak

3.16. Homes

Method	Path	Query Parameters	Content	Returned Value	Description
GET	/homes	**attribute**=*attributeName* **value**=*matchingValue*		A contained collection of Homes	Retrieves the homes whose **attribute** named *attributeName* has a **value** of *matchingValue* Findable attribute names for homes are: name, identifier, ponInternalKey, centralOfficeInternalKey and any additional attribute name.
POST	/homes		A contained collection of Homes		Import homes (create or update) **This operation requires the "Upload Data" system permission**
GET	/homes/**{internalKey}**			A Home	Retrieves the home with an internal key of **internalKey**
DELETE	/homes/**{internalKey}**			An operation ID	Trigger a delete on the home with an internal key of **internalKey**. The outcome will be received as a ObjectRemovalEvent
PUT	/homes/**{internalKey}**		A Home		Update an existing home
POST	/homes/**{internalKey}**/testOnDemand			An operation ID	Triggers a test on demand for the home with an internal key of **internalKey**. The result will be received as a HomeTestOnDemandEvent.
GET	/homes/attributes			A contained collection of Attributes	Retrieves the attributes of the home resource.
GET	/homes/homeTerminationTypes			A contained collection of HomeTerminationTypes	Retrieves the home termination types.
GET	/homes/testResult/**{peakInternalKey}**			A HomeTestResult	Retrieves the home test result matching the peak internal key.

3.17. PON test points

Method	Path	Query Parameters	Content	Returned Value	Description
GET	/pon-test-points	**attribute**=*attributeName* **value**=*matchingValue*		A contained collection of PonTestPoints	Retrieves the PON test points whose **attribute** named *attributeName* has a **value** of *matchingValue* Findable attribute names for PON test points are: ponInternalKey, upstreamTestPointInternalKey, centralOfficeInternalKey, name, group, type.

GET	/pon-test-points/all	**offset**=*result offset (for pagination)* **limit**=*the max results count (may be forced to a lower value than requested)*		An EntityList	Retrieves all the PON test points in the system in a paginated manner, returning minimal data about each of them.
POST	/pon-test-points		A contained collection of PonTestPoints		Import PON test points (create or update) **This operation requires the "Upload Data" system permission**
GET	/pon-test-points/**{internalKey}**			A PonTestPoint	Retrieves the PON test point with an internal key of **internalKey**
PUT	/pon-test-points/**{internalKey}**		A PonTestPoint		Update an existing PON test point
DELETE	/pon-test-points/**{internalKey}**				Delete the PON test point with an internal key of **internalKey**
GET	/pon-test-points/attributes			A contained collection of Attributes	Retrieves the attributes of the PON test point resource.

3.18. Alarms

Method	Path	Query Parameters	Content	Returned Value	Description
POST	/alarms/**{internalKey}**/acknowledge				Acknowledges the alarm with an internal key of **internalKey**
POST	/alarms/**{internalKey}**/clear				Clears the alarm with an internal key of **internalKey**
POST	/alarms/**{internalKey}**/unAcknowledge				Unacknowledges the alarm with an internal key of **internalKey**
POST	/alarms/**{internalKey}**/unClear				Unclear the alarm with an internal key of **internalKey**
PUT	/alarms/**{internalKey}**/comment		A plain text		Comment the alarm with an internal key of **internalKey**
POST	/alarms/resendAlarmEvents		An alarm sequence number		Resends alarm events, starting from the alarm sequence number
POST	/alarms/resynchronize				Resynchronizes alarm events for the authenticated user. The client will have to clear it's alarm cache and poll events to complete the resynchronization process.
GET	/alarms/**{internalKey}**/additionalAttributes			AdditionalAttributes	Get the additional attributes for the alarm with an internal key of **internalKey**.
PUT	/alarms/**{internalKey}**/additionalAttributes		AdditionalAttributes		Set the additional attributes for the alarm with an internal key of **internalKey**.
POST	/alarms/events/**{eventInternalKey}**/gpsCoordinates		GpsCoordinates		Update the GPS coordinates for the alarm event with an internal key of **eventInternalKey**.
POST	/alarms/filter		AlarmFilter	Alarms	Retrieve alarms using an alarm filter (with same features as ONMSi alarm viewer).

3.19. Events

Method	Path	Query Parameters	Content	Returned Value	Description
POST	/events			A contained collection of Events	Retrieves events that are new to the authenticated user. `POST` method is used to avoid cache problems, because this function is not idempotent.
GET	/events/maxEventCount			A count	Retrieves the maximum count of events that will be retrieved by `POST /events`. If such call retrieve this count of events, then another call is needed to retrieve remaining events.
PUT	/events/maxEventCount		A count		Sets the maximum count of events that will be retrieved by `POST /events`. If such call retrieve this count of events, then another call is needed to retrieve remaining events.
GET	/events/maxWaitTimeSec			A duration	Retrieves, in seconds, the maximum duration of the `POST /events` call before returning, when no event is available. This parameters ensures that the long delay polling mechanism works well with routers closing connections that last for too long.
PUT	/events/maxWaitTimeSec		A duration		Sets, in seconds, the maximum duration of the `POST /events` call before returning, when no event is available. This parameters ensures that the long delay polling mechanism works well with routers closing connections that last for too long.

Contained Collection: because managing raw collections can be tricky (ex: using JAXB), the REST API wraps the collections in container objects that can be easily (un)serialized.

4. PON cook book

4.20. Running a PON test

This recipe demonstrates how we can find a PON by its name, run a test on that PON, poll for the test progress and display its results.

First, we instanciate the OnmsiRsApiHelper that will ease API calls.

```
String hostName = InetAddress.getLocalHost().getHostName();
OnmsiRsApiHelper helper = new OnmsiRsApiHelper( hostName, "jdoe", "password123", "http" );
```

Then, we look for a PON named 'PON 1234'. Since there might be several matches, the return value is not a single PON. For the sake of simplicity, we only get the first one.

```
// Set the proper MIME type headers
helper.setContentTypeHeader( "application/xml" );
helper.setAcceptHeader( "application/xml" );

// Find a PON named 'PON 1234'
String foundPonsXml = helper.get( "/pons", new NameValuePair[]{
    new NameValuePair( "attribute", "name" ),
    new NameValuePair( "value", "PON 1234" ) } );
Pons foundPons = OnmsiJaxbHelper.convertXmlToObject( foundPonsXml, Pons.class );
Pon pon1234 = foundPons.getPon().get( 0 ); // assume we only want the first result
```

We can then start a test. The return value is a unique operation ID, which will be compared against operation IDs that we will receive through the events service. A word later on the `final` modifier of the operation ID.

```
// Start the test
String operationIdResult = helper.post( "/pons/" + pon1234.getInternalKey() + "/testOnDemand" );
final int operationId = Integer.parseInt( operationIdResult );
```

Waiting for the end of the test is done by calling `POST /events` repeatedly until we receive an event matching our operation ID. This is the purpose of the `OnmsiApiHelper.waitForEvent()` method created earlier.

```
OnmsiRsEventMatcher< PonTestOnDemandEvent > matcher = new OnmsiRsEventMatcher< PonTestOnDemandEvent >( PonTestOnDemandEvent.class )
{
    private final EnumSet< Outcome > m_endOfTestValues = EnumSet.of( Outcome.FAILURE, Outcome.SUCCESS );

    @Override
    public boolean match( PonTestOnDemandEvent event )
    {
        // compare to the 'final' operation ID stored above
        return (event.getOperationId() == operationId) && m_endOfTestValues.contains( event.getOutcome() );
    }
};
// Then poll events to wait for the end of test, up to 4 minutes
PonTestOnDemandEvent resultEvent = helper.waitForEvent( matcher, 240 );
```

The result of the test contains data that are similar to the PON detail screen

```
// Now read the result
System.out.println( "PON test completed: " + resultEvent.getOutcome() );

// Then get the test result
String ponTestResultXml = helper.get( "/pons/testResult/" + resultEvent.getPonTestInternalKey() );
PonTestResult ponTestResult = OnmsiJaxbHelper.convertXmlToObject( ponTestResultXml, PonTestResult.class );
List< Peak > peaks = ponTestResult.getPeaks();
for ( Peak peak : peaks )
{
    System.out.println( "-----------------------------------" );
    System.out.println( "Peak status: " + peak.getPeakStatus() );
    System.out.println( "Peak type: " + peak.getPeakType() );
    System.out.println( "Peak distance: " + peak.getDistanceM() );
    // ....
}
```

4.21. Download the OTDR trace of a PON test

The OTDR trace of a PON test can be downloaded if required.

The principle is that the client calls the API to generate a **download link** which will be valid for a single download only (and only for a few minutes).

```
stringResponse = helper.post( "/pons/testResult/" + resultEvent.getPonTestInternalKey() + "/trace" );

OtdrTraceDownloadKey downloadKey = OnmsiJaxbHelper.convertXmlToObject( stringResponse, OtdrTraceDownloadKey.class );
String singleShotKey = downloadKey.getKey();

System.out.println( String.format( String.format( "Trace is available at http://onmsi-server/download/trace/{0}", singleShotKey ) ) );
// Perform HTTP GET on http://onmsi-server/download/trace/${singleShotKey} to access the .sor file
```

4.22. Running a Home test

A home test is very similar to a PON test, except that it is triggered from a home. This example demonstrates how to find a home given its home identifier, and then how a test can be run for that Home

```
String hostName = InetAddress.getLocalHost().getHostName();
OnmsiRsApiHelper helper = new OnmsiRsApiHelper( hostName, "jdoe", "password123", "http" );

// Set the proper MIME type headers
helper.setContentTypeHeader( "application/xml" );
helper.setAcceptHeader( "application/xml" );

String homeIdentifier = "1234567890123";
String foundHomesXml = helper.get( "/homes", new NameValuePair[]{
    new NameValuePair( "attribute", "identifier" ),
    new NameValuePair( "value", homeIdentifier ) } );
Homes foundHomes = OnmsiJaxbHelper.convertXmlToObject( foundHomesXml, Homes.class );
Home home = foundHomes.getHome().get( 0 ); // assume we only want the first result

String operationIdResult = helper.post( "/homes/" + home.getInternalKey() + "/testOnDemand" );
final int operationId = Integer.parseInt( operationIdResult );

OnmsiRsEventMatcher< HomeTestOnDemandEvent > matcher = new OnmsiRsEventMatcher< HomeTestOnDemandEvent >( HomeTestOnDemandEvent.class )
{
    private final EnumSet< Outcome > m_endOfTestValues = EnumSet.of( Outcome.FAILURE, Outcome.SUCCESS );

    @Override
    public boolean match( HomeTestOnDemandEvent event )
    {
        // compare to the 'final' operation ID stored above
        return (event.getOperationId() == operationId) && m_endOfTestValues.contains( event.getOutcome() );
    }
};
HomeTestOnDemandEvent resultEvent = helper.waitForEvent( matcher, 240 );

String testResultXml = helper.get( "/homes/testResult/" + resultEvent.getPeakInternalKey() );
HomeTestResult testResult = OnmsiJaxbHelper.convertXmlToObject( testResultXml, HomeTestResult.class );
System.out.println( "Peak status = " + (testResult.getPeak() == null ? "null" : testResult.getPeak().getPeakStatus()) );
```

4.23. Getting the history of the tests on a PON

Retrieving a PON can be done from a home identifier as demonstrated below:

```
String hostName = InetAddress.getLocalHost().getHostName();
OnmsiRsApiHelper helper = new OnmsiRsApiHelper( hostName, "jdoe", "password123", "http" );

// Set the proper MIME type headers
helper.setContentTypeHeader( "application/xml" );
helper.setAcceptHeader( "application/xml" );

String homeIdentifier = "1234567890123";
String foundHomesXml = helper.get( "/homes", new NameValuePair[]{
    new NameValuePair( "attribute", "identifier" ),
    new NameValuePair( "value", homeIdentifier ) } );
Homes foundHomes = OnmsiJaxbHelper.convertXmlToObject( foundHomesXml, Homes.class );
Home home = foundHomes.getHome().get( 0 ); // assume we only want the first result
```

Then, we can retrieve the history of all the tests that were run for that PON.
Notice how the date marshaled back by REST Web Services is not a regular `java.util.Date` and requires some conversion.

```
// Now get the PON test history
String testHistoryXml = helper.get( "/pons/" + home.getPonInternalKey() + "/testHistory" );
PonTestHistories testHistories = OnmsiJaxbHelper.convertXmlToObject( testHistoryXml, PonTestHistories.class );
System.out.println( "PON test history:" );
for ( PonTestHistory testHistory : testHistories.getPonTestHistory() )
{
    XMLGregorianCalendar xmlGregorianCalendar = testHistory.getDate();
    GregorianCalendar gregorianCalendar = xmlGregorianCalendar.toGregorianCalendar();
    Date date = gregorianCalendar.getTime();
    System.out.println( date.toString() );
}
```

4.24. Changing the termination type for a home

The `GET /homes/homeTerminationTypes` allows to fetch all the defined home termination types.

```
String hostName = InetAddress.getLocalHost().getHostName();
OnmsiRsApiHelper helper = new OnmsiRsApiHelper( hostName, "jdoe", "password123", "http" );

helper.setContentTypeHeader( "application/xml" );
helper.setAcceptHeader( "application/xml" );

String homeTerminationTypeXml = helper.get( "/homes/homeTerminationTypes" );
HomeTerminationTypes terminationTypes = OnmsiJaxbHelper.convertXmlToObject( homeTerminationTypeXml, HomeTerminationTypes.class );
for ( HomeTerminationType homeTerminationType : terminationTypes.getHomeTerminationType() )
{
    System.out.println( homeTerminationType.getName() );
}
```

The latest termination type of a home, if any was already assigned, can be read in the `Home` object itself.
If no termination type was ever assign to a home, then its value is `null`.

```
String foundHomesXml = helper.get( "/homes", new NameValuePair[]{
    new NameValuePair( "attribute", "identifier" ),
    new NameValuePair( "value", "1234567890123" ) } );
Homes foundHomes = OnmsiJaxbHelper.convertXmlToObject( foundHomesXml, Homes.class );
Home home = foundHomes.getHome().get( 0 ); // assume we only want the first result

HomeTerminationType homeTerminationType = home.getTerminationType();
System.out.println( "Home termination type: " + (homeTerminationType == null ? "NONE" : homeTerminationType.getName()) );
```

Finally, the termination type of a home can be modified as well.

```
// Assuming that there are at least 3 types, get the 3rd one
HomeTerminationType nextTerminationType = terminationTypes.getHomeTerminationType().get( 2 );
home.setTerminationType( nextTerminationType );
String homeXml = OnmsiJaxbHelper.convertObjectToXml( home, Home.class );
helper.put( "/homes/" + home.getInternalKey(), homeXml );

// Fetch the home again to check that its termination type was updated
homeXml = helper.get( "/homes/" + home.getInternalKey() );
home = OnmsiJaxbHelper.convertXmlToObject( homeXml, Home.class );
assert home.getTerminationType().getName().equals( nextTerminationType.getName() );
```

4.25. Assigning a reference peak to a home

When a home has no reference peak yet, assigning a reference peak to a home also means assigning a home to a peak. This is more what a user perceives when using the PON detail screen.

We first find our home and the PON:

```
String hostName = InetAddress.getLocalHost().getHostName();
OnmsiRsApiHelper helper = new OnmsiRsApiHelper( hostName, "jdoe", "password123", "http" );

// Set the proper MIME type headers
helper.setContentTypeHeader( "application/xml" );
helper.setAcceptHeader( "application/xml" );

String homeIdentifier = "1234567890123";
String foundHomesXml = helper.get( "/homes", new NameValuePair[]{
    new NameValuePair( "attribute", "identifier" ),
    new NameValuePair( "value", homeIdentifier ) } );
Homes foundHomes = OnmsiJaxbHelper.convertXmlToObject( foundHomesXml, Homes.class );
Home home = foundHomes.getHome().get( 0 ); // assume we only want the first result
```

For that PON, we browse the peaks of the latest test and look for a candidate (for the sake of simplicity, we just pick up any peak but in real life, you would be picking a meaningful peak).

```
// Now get the PON test history
String testHistoryXml = helper.get( "/pons/" + home.getPonInternalKey() + "/testHistory" );
PonTestHistories testHistories = OnmsiJaxbHelper.convertXmlToObject( testHistoryXml, PonTestHistories.class );

// ... read the PON tests for the latest (ordered by decreasing dates)...
long ponTestInternalKey = testHistories.getPonTestHistory().get( 0 ).getPonTestInternalKey();

String ponTestResultXml = helper.get( "/pons/testResult/" + ponTestInternalKey );
PonTestResult ponTestResult = OnmsiJaxbHelper.convertXmlToObject( ponTestResultXml, PonTestResult.class );

List< Peak > peaks = ponTestResult.getPeaks();
```

```
// Loop through all non missing peaks which are at 2.2km or further and
// grab the most far away
Peak anotherPeak = null;
for ( Peak peak : peaks )
{
    if ( peak.getDistanceM() > 2200.0 && peak.getPeakStatus() != PeakStatus.MISSING )
    {
        anotherPeak = peak;
    }
}
if ( anotherPeak == null )
{
    throw new IllegalStateException( "No peak found" );
}
```

Assigning a reference peak for that home is done easily:

```
home.setReferencePeak( anotherPeak );

String homeXml = OnmsiJaxbHelper.convertObjectToXml( home, Home.class );
helper.put( "/homes/" + home.getInternalKey(), homeXml );
```

Removing a reference peak for that home is equally simple. This is the same as choosing 'NOT_REFERENCED' for that peak inside the ONMSi application.

```
home.setReferencePeak( null );

homeXml = OnmsiJaxbHelper.convertObjectToXml( home, Home.class );
helper.put( "/homes/" + home.getInternalKey(), homeXml );

homeXml = helper.get( "/homes/" + home.getInternalKey() );
home = OnmsiJaxbHelper.convertXmlToObject( homeXml, Home.class );

assert home.getLatestPeak() == null;
assert home.getReferencePeak() == null;
```

4.26. Changing the state of a peak

In some cases, you will need to update the state of a peak from your programs: change a new peak to be a NOT_HOME or vice-versa is something that cannot be done with the HomeService: the PonService is used for these task, as in the application GUI.

First, you need to get hold of a peak, either by running a test, or by using the test history. Since we've done that already, we will not go into much details.

```
String hostName = InetAddress.getLocalHost().getHostName();
OnmsiRsApiHelper helper = new OnmsiRsApiHelper( hostName, "jdoe", "password123", "http" );

// Set the proper MIME type headers
helper.setContentTypeHeader( "application/xml" );
helper.setAcceptHeader( "application/xml" );

String httpBufferXml = helper.get( "/homes", new NameValuePair[]{
    new NameValuePair( "attribute", "identifier" ),
    new NameValuePair( "value", "1234567890123" ) } );
Homes foundHomes = OnmsiJaxbHelper.convertXmlToObject( httpBufferXml, Homes.class );
Home home = foundHomes.getHome().get( 0 ); // assume we only want the first result

// Now get the PON test history...
httpBufferXml = helper.get( "/pons/" + home.getPonInternalKey() + "/testHistory" );
PonTestHistories testHistories = OnmsiJaxbHelper.convertXmlToObject( httpBufferXml, PonTestHistories.class );

// ... read the PON tests for the latest (ordered by decreasing dates)...
long ponTestInternalKey = testHistories.getPonTestHistory().get( 0 ).getPonTestInternalKey();
httpBufferXml = helper.get( "/pons/testResult/" + ponTestInternalKey );
PonTestResult ponTestResult = OnmsiJaxbHelper.convertXmlToObject( httpBufferXml, PonTestResult.class );
List< Peak > peaks = ponTestResult.getPeaks();

// ... and finally the latest peak (ordered by increasing distance)
Peak peak = peaks.get( peaks.size() - 1 );
```

Then you update its state and call `PonService.update()`. Notice that the editable values inside the peak need to be kept in a coherent state: ONMSi will not try to guess your intent when invalid values are provided (such as setting NOT_HOME with a not null home internal key).

```
// Assuming peak type is initially a NOT_HOME
// A peak internal key
long referencePeakInternalKey = 12345678L;

// NOT_HOME => HOME
peak.setPeakType( PeakType.HOME );
peak.setHomeInternalKey( home.getInternalKey() );
helper.put( "/pons/peaks/" + peak.getInternalKey(), OnmsiJaxbHelper.convertObjectToXml( peak, Peak.class ) );
httpBufferXml = helper.get( "/pons/peaks/" + peak.getInternalKey() );
peak = OnmsiJaxbHelper.convertXmlToObject( httpBufferXml, Peak.class );

// HOME => NOT_REFERENCED
peak.setPeakType( PeakType.NOT_REFERENCED );
peak.setReferenceInternalKey( null );
peak.setHomeInternalKey( null );
helper.put( "/pons/peaks/" + peak.getInternalKey(), OnmsiJaxbHelper.convertObjectToXml( peak, Peak.class ) );
httpBufferXml = helper.get( "/pons/peaks/" + peak.getInternalKey() );
peak = OnmsiJaxbHelper.convertXmlToObject( httpBufferXml, Peak.class );

// NOT_REFERENCED => NOT_HOME
peak.setPeakType( PeakType.NOT_HOME );
peak.setReferenceInternalKey( referencePeakInternalKey );
helper.put( "/pons/peaks/" + peak.getInternalKey(), OnmsiJaxbHelper.convertObjectToXml( peak, Peak.class ) );
httpBufferXml = helper.get( "/pons/peaks/" + peak.getInternalKey() );
peak = OnmsiJaxbHelper.convertXmlToObject( httpBufferXml, Peak.class );

// NOT_HOME => NOT_REFERENCED
peak.setPeakType( PeakType.HOME );
peak.setReferenceInternalKey( null );
helper.put( "/pons/peaks/" + peak.getInternalKey(), OnmsiJaxbHelper.convertObjectToXml( peak, Peak.class ) );
httpBufferXml = helper.get( "/pons/peaks/" + peak.getInternalKey() );
peak = OnmsiJaxbHelper.convertXmlToObject( httpBufferXml, Peak.class );

// NOT_REFERENCED => HOME
peak.setPeakType( PeakType.HOME );
peak.setReferenceInternalKey( referencePeakInternalKey );
peak.setHomeInternalKey( home.getInternalKey() );
helper.put( "/pons/peaks/" + peak.getInternalKey(), OnmsiJaxbHelper.convertObjectToXml( peak, Peak.class ) );
httpBufferXml = helper.get( "/pons/peaks/" + peak.getInternalKey() );
peak = OnmsiJaxbHelper.convertXmlToObject( httpBufferXml, Peak.class );

// HOME => NOT_HOME
peak.setPeakType( PeakType.NOT_HOME );
peak.setHomeInternalKey( null );
helper.put( "/pons/peaks/" + peak.getInternalKey(), OnmsiJaxbHelper.convertObjectToXml( peak, Peak.class ) );
```

```
httpBufferXml = helper.get( "/pons/peaks/" + peak.getInternalKey() );
peak = OnmsiJaxbHelper.convertXmlToObject( httpBufferXml, Peak.class );
```

4.27. Changing the reference of a peak

Continuing with the previous code example, you may also change the reference of a peak. Typical use case is to select the latest measurement as a reference.

```
// Assuming peak type is initially a NOT_HOME
peak.setReferenceInternalKey( peak.getInternalKey() );
helper.put( "/pons/peaks/" + peak.getInternalKey(), OnmsiJaxbHelper.convertObjectToXml( peak, Peak.class ) );
```

4.28. Creation of PON, Home and Test Point elements into ONMSi

There are a few variables which need to be considered prior to running the import.

```
int baseIdentifier = 98765000;
// Base value for keys in an other system (trouble ticket, GIS, ...)
String baseKey = "XYZ_API_IMPORT";
String ponName = "PON 1234";
String ponExternalKey = "PON_" + baseKey;
String description = "Imported by John Doe through the API";
```

All entities support `String` additional attributes which can be set and stored in the database. This can be used, for instance to store the key of your home in a trouble ticket system.

Finally, the PON can be created, added to a list because the API is design to import several PONs (up to 50) in one shot, and the list is imported.
The central office internal key can be found by through the central office service.
You can also automatically associate the PON to an OTU port by setting its internal key (see the OTU service to retrieve OTU ports).

```
Pon pon = new Pon();
pon.setName( ponName );
pon.setDescription( description );
// The CO internal key can be retrieved through the central office service
pon.setCentralOfficeInternalKey( INTERNAL_KEY_OF_AN_EXISTING_CO );
// uncomment the following line to automatically associate the PON to an existing OTU port
// pon.setPortInternalKey( INTERNAL_KEY_OF_A_PORT_BELONGING_TO_AN_OTU_ASSOCIATED_TO_THE_CO );

Entry externalKeyAdditionalAttribute = new Entry();
// set as key the name of the PON additional attribute used as external key
externalKeyAdditionalAttribute.setKey( "_externalKey" );
externalKeyAdditionalAttribute.setValue( ponExternalKey );
pon.setAdditionalAttributes( new AdditionalAttributes() );
pon.getAdditionalAttributes().getEntry().add( externalKeyAdditionalAttribute );
Pons pons = new Pons();
pons.getPon().add( pon );

System.out.println( "Generating PON: '" + pon.getName() + "'" );

String hostName = InetAddress.getLocalHost().getHostName();
OnmsiRsApiHelper helper = new OnmsiRsApiHelper( hostName, "jdoe", "password123", "http" );

// Set the proper MIME type headers
helper.setContentTypeHeader( "application/xml" );
helper.setAcceptHeader( "application/xml" );

String ponsXml = OnmsiJaxbHelper.convertObjectToXml( pons, Pons.class );
helper.post( "/pons", ponsXml );

// Now find the PON again to get check that it was
// imported and to get its internal key
ponsXml = helper.get( "/pons", new NameValuePair[]{
    new NameValuePair( "attribute", "name" ),
    new NameValuePair( "value", ponName ) } );
pons = OnmsiJaxbHelper.convertXmlToObject( ponsXml, Pons.class );
pon = pons.getPon().get( 0 );
```

Similarly, the homes can be created and imported (this time, the example shows how several homes are imported in one go).

```
Homes homes = new Homes();
for ( int i = 0; i < 4; i++ )
{
    Home home = new Home();
    String identifier = Long.toString( baseIdentifier + i );
    home.setIdentifier( identifier );
    home.setName( "Home " + identifier );
    home.setDescription( description );
    home.setPonInternalKey( pon.getInternalKey() );
    home.setAdditionalAttributes( new AdditionalAttributes() );
    externalKeyAdditionalAttribute = new Entry();
    // set as key the name of the home additional attribute used as external key
    externalKeyAdditionalAttribute.setKey( "_externalKey" );
    externalKeyAdditionalAttribute.setValue( "HOME_" + i + "_" + baseKey );
    home.getAdditionalAttributes().getEntry().add( externalKeyAdditionalAttribute );
    homes.getHome().add( home );
    System.out.println( "Generating HOME: '" + home.getName() + "'" );
}
String homesXml = OnmsiJaxbHelper.convertObjectToXml( homes, Homes.class );
helper.post( "/homes", homesXml );
```

Similarly, the test points can be created and imported.

> **WARNING**: upstream test points have to be created ahead of their downstream test points..

```
PonTestPoints ponTestPoints = new PonTestPoints();

// create the splitter test point
PonTestPoint splitterPonTestPoint = new PonTestPoint();
String identifier = Long.toString( baseIdentifier );
splitterPonTestPoint.setName( "TP " + identifier );
splitterPonTestPoint.setPonInternalKey( pon.getInternalKey() );
splitterPonTestPoint.setInsertionLossThresholdFromOriginDb( SPLITTER_INSERTION_LOSS_THRESHOLD_FROM_ORIGIN_DB );
splitterPonTestPoint.setType( "SPLITTER" );
ponTestPoints.getPonTestPoint().add( splitterPonTestPoint );
System.out.println( "Generating splitter test point: '" + splitterPonTestPoint.getName() + "'" );
String ponTestPointsXml = OnmsiJaxbHelper.convertObjectToXml( ponTestPoints, PonTestPoints.class );
helper.post( "/pon-test-points", ponTestPointsXml );

// fetch the created splitter test point to get its internal key
// (because the PON is new, it only has a single test point: the splitter test point)
ponTestPointsXml = helper.get( "/pon-test-points", new NameValuePair[] {
    new NameValuePair( "attribute", "ponInternalKey" ),
    new NameValuePair( "value", String.valueOf( pon.getInternalKey() ) ) } );
```

```
        ponTestPoints = OnmsiJaxbHelper.convertXmlToObject( ponTestPointsXml, PonTestPoints.class );
        splitterPonTestPoint = ponTestPoints.getPonTestPoint().get( 0 );

        // create the connection box test points
        ponTestPoints.getPonTestPoint().clear();
        for ( int i = 1; i <= 16; i++ )
        {
            PonTestPoint ponTestPoint = new PonTestPoint();
            ponTestPoint.setName( String.format( "TP %s-%d", identifier, i ) );
            ponTestPoint.setPonInternalKey( pon.getInternalKey() );
            ponTestPoint.setUpstreamTestPointInternalKey( splitterPonTestPoint.getInternalKey() );
            ponTestPoint.setInsertionLossThresholdFromOriginDb( CONNECTION_BOX_INSERTION_LOSS_THRESHOLD_FROM_ORIGIN_DB );
            ponTestPoint.setType( "CONNECTION_BOX" );
            ponTestPoints.getPonTestPoint().add( ponTestPoint );
            System.out.println( "Generating test point: '" + ponTestPoint.getName() + "'" );
        }
        ponTestPointsXml = OnmsiJaxbHelper.convertObjectToXml( ponTestPoints, PonTestPoints.class );
        helper.post( "/pon-test-points", ponTestPointsXml );
```

4.29. Running a PON calibration

This recipe demonstrates how we can find a PON by its name, run a calibration on that PON, poll for the calibration progress and display its results.

First, we instanciate the OnmsiRsApiHelper that will ease API calls.

```
        String hostName = InetAddress.getLocalHost().getHostName();
        OnmsiRsApiHelper helper = new OnmsiRsApiHelper( hostName, "jdoe", "password123", "http" );
```

Then, we look for a PON named 'PON 1234'. Since there might be several matches, the return value is not a single PON. For the sake of simplicity, we only get the first one.

```
        // Set the proper MIME type headers
        helper.setContentTypeHeader( "application/xml" );
        helper.setAcceptHeader( "application/xml" );

        // Find a PON named 'PON 1234'
        String foundPonsXml = helper.get( "/pons", new NameValuePair[]{
            new NameValuePair( "attribute", "name" ),
            new NameValuePair( "value", "PON 1234" ) } );
        Pons foundPons = OnmsiJaxbHelper.convertXmlToObject( foundPonsXml, Pons.class );
        Pon pon1234 = foundPons.getPon().get( 0 ); // assume we only want the first result
```

We can then start a calibration. The return value is a unique operation ID, which will be compared against operation IDs that we will receive through the events service. A word later on the `final` modifier of the operation ID.

```
        // Setup the calibration configuration
        PonCalibrationConfig ponCalibrationConfig = new PonCalibrationConfig();
        ponCalibrationConfig.setForceAcquisition( true );
        ponCalibrationConfig.setReferenceZoneBeginDistanceM( 12.3 );
        ponCalibrationConfig.setReferenceZoneEndDistanceM( 23.4 );
        String ponCalibrationConfigXml = OnmsiJaxbHelper.convertObjectToXml( ponCalibrationConfig, PonCalibrationConfig.class );

        // Start the calibration
        String operationIdResult = helper.post( "/pons/" + pon1234.getInternalKey() + "/calibration", ponCalibrationConfigXml );
        final int operationId = Integer.parseInt( operationIdResult );
```

Waiting for the end of the calibration is done by calling `POST /events` repeatedly until we receive an event matching our operation ID. This is the purpose of the `OnmsiApiHelper.waitForEvent()` method created earlier.

```
        OnmsiRsEventMatcher< PonCalibrationEvent > matcher = new OnmsiRsEventMatcher< PonCalibrationEvent >( PonCalibrationEvent.class )
        {
            private final EnumSet< Outcome > m_endOfTestValues = EnumSet.of( Outcome.FAILURE, Outcome.SUCCESS );

            @Override
            public boolean match( PonCalibrationEvent event )
            {
                // compare to the 'final' operation ID stored above
                return (event.getOperationId() == operationId) && m_endOfTestValues.contains( event.getOutcome() );
            }
        };
        // Then poll events to wait for the end of test, up to 4 minutes
        PonCalibrationEvent calibrationEvent = helper.waitForEvent( matcher, 240 );
```

The result of the calibration contains data that are similar to the OTU port association screen

```
        // Now read the result
        System.out.println( "PON calibration completed: " + calibrationEvent.getOutcome() );

        // Then get the calibration result
        String ponCalibrationResultXml = helper.get( "/pons/" + calibrationEvent.getInternalKey() + "/calibration" );
        PonCalibrationResult ponCalibrationResult = OnmsiJaxbHelper.convertXmlToObject( ponCalibrationResultXml, PonCalibrationResult.class );

        // Process calibration result
        System.out.println( "-------------------------------------" );
        System.out.println( "Reference zone: " + ponCalibrationResult.getReferenceZoneBeginDistanceM() + " - " + ponCalibrationResult.getReferenceZoneEndDistanceM()
);
        System.out.println( "Calibration : " + ponCalibrationResult.isCalibrated() );
        System.out.println( "- Level : " + ponCalibrationResult.isLevelValid() + " / " + ponCalibrationResult.getLevelDb() );
        System.out.println( "- Deviation : " + ponCalibrationResult.isDeviationValid() + " / " + ponCalibrationResult.getDeviationDb() );
```

5. Point to point cook book

5.30. Enable/disable link monitoring

Disabling the monitoring for a link is done by a fetch-modify-update cycle.

```
        String hostName = InetAddress.getLocalHost().getHostName();
        OnmsiRsApiHelper helper = new OnmsiRsApiHelper( hostName, "jdoe", "password123", "http" );

        // Set the proper MIME type headers
        helper.setContentTypeHeader( "application/xml" );
        helper.setAcceptHeader( "application/xml" );

        String linksXml = helper.get( "/links", new NameValuePair[]{
            new NameValuePair( "attribute", "name" ),
            new NameValuePair( "value", "Link 1234") } );
        Links links = OnmsiJaxbHelper.convertXmlToObject( linksXml, Links.class );
        Link link = links.getLink().get( 0 ); // assume one and only one result

        // Disable
```

```
link.setMeasureEnabled( false );
String linkXml = OnmsiJaxbHelper.convertObjectToXml( link, Link.class );
helper.put( "/links/" + link.getInternalKey(), linkXml );

// Make sure we see the updated value
linkXml = helper.get( "/links/" + link.getInternalKey() );
link = OnmsiJaxbHelper.convertXmlToObject( linkXml, Link.class );
assert link.isMeasureEnabled() == false;
```

Enabling monitoring is very similar.

```
// Enable
link.setMeasureEnabled( true );
linkXml = OnmsiJaxbHelper.convertObjectToXml( link, Link.class );
helper.put( "/links/" + link.getInternalKey(), linkXml );

// Make sure we see the updated value
linkXml = helper.get( "/links/" + link.getInternalKey() );
link = OnmsiJaxbHelper.convertXmlToObject( linkXml, Link.class );
assert link.isMeasureEnabled() == true;
```

5.31. Start a test on demand on a link

Starting the monitoring test for a link requires finding the test to be run.

```
String hostName = InetAddress.getLocalHost().getHostName();
OnmsiRsApiHelper helper = new OnmsiRsApiHelper( hostName, "jdoe", "password123", "http" );

// Set the proper MIME type headers
helper.setContentTypeHeader( "application/xml" );
helper.setAcceptHeader( "application/xml" );

String linksXml = helper.get( "/links", new NameValuePair[] {
    new NameValuePair( "attribute", "name" ),
    new NameValuePair( "value", "Link 1234") } );
Links links = OnmsiJaxbHelper.convertXmlToObject( linksXml, Links.class );
Link link = links.getLink().get( 0 ); // assume one and only one result

// Get all tests for the link
String testsXml = helper.get( "/monitoringTests", new NameValuePair[] {
    new NameValuePair( "attribute", "linkInternalKey" ),
    new NameValuePair("value", String.valueOf( link.getInternalKey() ) ) } );
MonitoringTests tests = OnmsiJaxbHelper.convertXmlToObject( testsXml, MonitoringTests.class );
MonitoringTest monitoringTest = tests.getMonitoringTest().get( 0 ); // grab the first test
```

Then, a test can be started. Notice how this only returns an operation ID and how the actual progress of the test will be notified through the event service, similarly to PON or Home tests. We therefore create an event matcher for the operation ID of the started test.

```
String operationIdResult = helper.post( "/monitoringTests/" + monitoringTest.getInternalKey() + "/testOnDemand" );
final int operationId = Integer.parseInt( operationIdResult );

OnmsiRsEventMatcher< MonitoringTestOnDemandEvent > matcher = new OnmsiRsEventMatcher< MonitoringTestOnDemandEvent >( MonitoringTestOnDemandEvent.class )
{
    private final EnumSet< Outcome > m_endOfTestValues = EnumSet.of( Outcome.FAILURE, Outcome.SUCCESS );

    @Override
    public boolean match( MonitoringTestOnDemandEvent event )
    {
        // compare to the 'final' operation ID stored above
        return (event.getOperationId() == operationId) && m_endOfTestValues.contains( event.getOutcome() );
    }
};
```

With the event matcher defined above, we can then wait for the event, just as in PON or Home tests. There is one difference though: monitoring tests never return a test result (i.e. you cannot determine whether there was a defect on the monitored link or not): instead in case of test failure, an alarm is raised, just as if the test had been run by the normal test sequencer. But the outcome in the test event will tell you whether the test was properly performed or not.

```
// Wait up to 4 minutes for the event specified by the matcher above
MonitoringTestOnDemandEvent resultEvent = helper.waitForEvent( matcher, 240 );

// The outcome only tells if the test was properly run: if a failure occurred, an alarm will be raised
System.out.println( "Test on demand ended with outcome: " + resultEvent.getOutcome().toString() );
```

5.32. Changing alarm states

Changing the CLEAR and ACKNOWLEDGE statuses from any alarm can be accomplished by `POST /alarms/{id}/action` methods.

Following example shows how to wait for a new equipment alarm (up to 10min) and then change its state automatically.

```
String hostName = InetAddress.getLocalHost().getHostName();
OnmsiRsApiHelper helper = new OnmsiRsApiHelper( hostName, "jdoe", "password123", "http" );
// setup an event matcher to catch new alarm events of equipment alarms
OnmsiRsEventMatcher< SequentialAlarmEvent > matcher = new OnmsiRsEventMatcher< SequentialAlarmEvent >( SequentialAlarmEvent.class )
{
    @Override
    public boolean match( SequentialAlarmEvent event )
    {
        AbstractAlarmEvent alarmEvent = event.getAlarmEvent();
        AlarmType alarmType = alarmEvent.getAlarm().getAlarmType();
        return alarmEvent instanceof NewAlarmEvent && alarmType == AlarmType.EQUIPMENT_ALARM;
    }
};

// Wait up to 10 minutes for a new equipment alarm
int timeoutSec = 10 * 60;
SequentialAlarmEvent sequentialEvent = helper.waitForEvent( matcher, timeoutSec );
// retrieve the alarm internal key
long alarmInternalKey = sequentialEvent.getAlarmEvent().getAlarm().getInternalKey();
```

Then, any action can be taken: whether acknowledge

```
helper.post( "/alarms/" + alarmInternalKey + "/acknowledge" );
```

... then un-acknowledge...

```
helper.post( "/alarms/" + alarmInternalKey + "/unAcknowledge" );
```

... or clear...

```
        helper.post( "/alarms/" + alarmInternalKey + "/clear" );
```

... then unclear...

```
        helper.post( "/alarms/" + alarmInternalKey + "/unClear" );
```

6. Misc cook book

6.33. Running a lightsource measure

Starting lightsource measure on a port requires finding the OTU run it, then the OTDR module and port to use.

```
        String hostName = InetAddress.getLocalHost().getHostName();
        OnmsiRsApiHelper helper = new OnmsiRsApiHelper( hostName, "jdoe", "password123", "http" );

        // Set the proper MIME type headers
        helper.setContentTypeHeader( "application/xml" );
        helper.setAcceptHeader( "application/xml" );

        // find the OTU
        String otusXml = helper.get( "/otus", new NameValuePair[] {
            new NameValuePair( "attribute", "name" ),
            new NameValuePair( "value", "OTU LS") } );
        Otus otus = OnmsiJaxbHelper.convertXmlToObject( otusXml, Otus.class );
        Otu otu = otus.getOtu().get( 0 ); // assume one and only one result
        Otdr otdr = otu.getOtdrs().get( 0 ); // use the first OTDR

        // Get all ports for the OTU
        String portsXml = helper.get( String.format( "/otus/%d/ports", otu.getInternalKey() ) );
        Ports ports = OnmsiJaxbHelper.convertXmlToObject( portsXml, Ports.class );
        Port port = ports.getPort().get( 0 ); // use the first port
```

Then, the measure can be started. Notice how this only returns an operation ID and how the actual progress of the measure will be notified through the event service, similarly to PON or Home tests. We therefore create an event matcher for the operation ID of the started measure.

```
        // build the measure parameters
        LightsourceMeasureParameters parameters = new LightsourceMeasureParameters();
        parameters.setOtuInternalKey( otu.getInternalKey() );
        parameters.setOtdrInternalKey( otdr.getInternalKey() );
        parameters.setPortInternalKey( port.getInternalKey() );
        String[] wavelengths = otdr.getWavelengths().split( "," ); // wavelength list is comma separated
        parameters.setWavelengthNm( Integer.parseInt( wavelengths[0] ) ); // use the first wavelength
        parameters.setToneHz( otdr.getLightsourceTonesHz().get( 0 ) ); // use the first tone (and empty tone list would mean the OTDR is not able to perform
lightsource measure)
        parameters.setDurationSec( 30 );

        // start the measure
        String parametersXml = OnmsiJaxbHelper.convertObjectToXml( parameters, LightsourceMeasureParameters.class );
        String operationIdResult = helper.post( String.format( "/otus/%d/ports/%d/lightsource-measure", otu.getInternalKey(), port.getInternalKey() ), parametersXml
);
        final int operationId = Integer.parseInt( operationIdResult );

        // build a matcher for LightsourceMeasureOnDemandEvent
        OnmsiRsEventMatcher< LightsourceMeasureOnDemandEvent > matcher = new OnmsiRsEventMatcher< LightsourceMeasureOnDemandEvent >(
LightsourceMeasureOnDemandEvent.class )
        {
            private final EnumSet< Outcome > m_endOfTestValues = EnumSet.of( Outcome.FAILURE, Outcome.SUCCESS );

            @Override
            public boolean match( LightsourceMeasureOnDemandEvent event )
            {
                // compare to the 'final' operation ID stored above
                return (event.getOperationId() == operationId) && m_endOfTestValues.contains( event.getOutcome() );
            }
        };
```

With the event matcher defined above, we can then wait for the event, just as in PON or Home tests. There is one difference though: lightsource measure never return a result. But the outcome in the measure event will tell you whether the measure was properly performed or not.

```
        // Wait up to 4 minutes for the event specified by the matcher above
        LightsourceMeasureOnDemandEvent resultEvent = helper.waitForEvent( matcher, 240 );

        // The outcome only tells if the test was properly run: if a failure occurred, an alarm will be raised
        System.out.println( "Lightsource ended with outcome: " + resultEvent.getOutcome().toString() );
```

6.34. Handling API session with Spring Boot

Managing the API_SESSION cookie using Spring Boot. to maintain the API session on backend side. This example shows how to maintain the API session on backend using the HTTP header cookie. That technique could be easily translate to some others technologies.

Please read the API Authentication to understand the API ssession Lifecycle.

```
@Component
public class ExOnmsiApi
{
    private final static String API_SESSION_COOKIE_KEY = "API_SESSION";

    /** Local Spring Boot REST template */
    private RestTemplate restTemplate;

    /** Local cached API session */
    private HttpCookie apiSessionCookie;

    private RestTemplate getRestTemplate()
    {
        if ( this.restTemplate == null )
        {
            // REST template for authenticated user
            this.restTemplate = new RestTemplate( clientHttpRequestFactory() );

            this.restTemplate.getInterceptors().add( new OaOnmsiHttpRequestInterceptor() );
        }
        return this.restTemplate;
    }

    private class OaOnmsiHttpRequestInterceptor
        implements
        ClientHttpRequestInterceptor
    {
        private void injectApiSessionCookieToHeaders( HttpHeaders headers )
        {
```

```
            if ( ExOnmsiApi.this.apiSessionCookie != null )
            {
                String apiSessionCookieString = ExOnmsiApi.this.apiSessionCookie.getName() + "=" + ExOnmsiApi.this.apiSessionCookie.getValue() + ";";
                headers.add( HttpHeaders.COOKIE, apiSessionCookieString );
            }
        }

        private void extractApiSessionCookieFromHeaders( HttpHeaders headers )
        {
            List< String > respCookieStrings = headers.get( HttpHeaders.SET_COOKIE );
            if ( respCookieStrings != null && !respCookieStrings.isEmpty() )
            {
                respCookieStrings.stream()
                    .map( respCookieString -> HttpCookie.parse( respCookieString ) )
                    .forEachOrdered( respCookies ->
                        {
                            respCookies.forEach( respCookie ->
                                {
                                    if ( respCookie.getName().equalsIgnoreCase( API_SESSION_COOKIE_KEY ) )
                                    {
                                        ExOnmsiApi.this.apiSessionCookie = respCookie;
                                    }
                                });
                        });
            }
        }

        @Override
        public ClientHttpResponse intercept( HttpRequest request, byte[] body, ClientHttpRequestExecution execution )
            throws IOException
        {
            HttpRequestWrapper requestWrapper = new HttpRequestWrapper( request );

            // Accept
            List< MediaType > medias = new ArrayList<>();
            medias.add( MediaType.APPLICATION_XML );
            medias.add( MediaType.TEXT_PLAIN );
            requestWrapper.getHeaders().setAccept( medias );

            // Content
            requestWrapper.getHeaders().setContentType( MediaType.APPLICATION_XML );

            // Authorization token
            String authorizaton = "Basic " +
                new String( Base64.encodeBase64(
                    String.format( "%s:%s", "jdoe", "password123" ).getBytes() ) );
            requestWrapper.getHeaders().add( "Authorization", authorizaton );

            // Inject the API session cookie
            injectApiSessionCookieToHeaders( requestWrapper.getHeaders() );

            // Execute
            ClientHttpResponse response = execution.execute( requestWrapper, body );

            // Store the API sesion cookie
            extractApiSessionCookieFromHeaders( response.getHeaders() );

            // Return response
            return response;
        }
    }
}
```

GetEvents example using Spring Boot REST template, with the use of 'interceptor' defined above.

```
@Component
public class ExOnmsiApi
{
    private String getUrl( String path )
    {
        String baseUrl = String.format( "%s://%s:%d/rs",
            "http",
            "myFavoriteHost",
            80 );
        return baseUrl + path;
    }

    public List< Event > getEvents()
        throws RestClientException
    {
        List< Event > result = new ArrayList<>();

        String resourceUrl = getUrl( "/events" );

        URI uri = UriComponentsBuilder
            .fromUriString( resourceUrl )
            .build()
            .encode()
            .toUri();

        // @formatter:off
        ResponseEntity< Events > response = getRestTemplate().exchange(
            uri,
            HttpMethod.POST,
            null,
            Events.class );
        // @formatter:on

        if ( response.getStatusCode().is2xxSuccessful() )
        {
            result = response.getBody().getEvent();
        }

        return result;
    }
}
```

Table of content

Back to the WS page