# Web Services

Planon Software Suite

Version: L105

Planon

Building Connections

# About this Document

## Intended Audience

This document is intended for *Planon Software Suite* users.

## Contacting us

If you have any comments or questions regarding this document, please send them to: support@planonsoftware.com.

## Document Conventions

**Bold**
Names of menus, options, tabs, fields and buttons are displayed in bold type.

*Italic text*
Application names are displayed in italics.

CAPITALS
Names of keys are displayed in upper case.

### Special symbols

| | |
|---|---|
| ℹ | Text preceded by this symbol references additional information or a tip. |
| ❗ | Text preceded by this symbol is intended to alert users about consequences if they carry out a particular action in Planon. |

# Table of Contents

# Web services

The Planon ProCenter software is often deployed in environments with many other enterprise applications. Because of this, customers want to integrate Planon ProCenter into their common business processes allowing other enterprise applications to interface and use data from Planon ProCenter.

This approach shifts the focus of Planon ProCenter as being a ‘stand-alone enterprise application’ to a broader ‘service oriented’ solution in which enterprise applications are loosely coupled through well-defined interfaces.

In order to allow Planon ProCenter to play a role in such a Service Oriented Architecture (SOA), we need to be able to interface with other applications. One of the major approaches to SOA is the use of Web services for tying applications together. Planon ProCenter offers basic support for Web services.

To be able to use Web services, make sure that the appropriate user group is linked to the *PPWS* product definition: *Web Services*. For more information, see product definitions (Accounts).

# Creating a web service

This section describes how to create a web service.

## Adding a web service definition

When you add a web service definition using *Planon ProCenter*, you define which business objects the web service will use. Next, you define which fields of these business objects will be used by the web service. Finally, with the business objects and fields specified, you can generate your new web service.

### Procedure

1. Select Tools > Web Services on the navigation panel.
2. On the action panel, click Add.
3. Specify a unique System name and a Description for your new web service.

   The **Package name** is updated automatically. The package name displays the input to the current web services instance.
4. Click Save.

A new web service definition is added.

## Linking business object definitions to a web service definition

You can link business object definitions to the web services that you have created.

### Procedure

1. In Web services, select a web service definition.
2. On the action panel, click Link business object definitions.

   Select one or more business objects from the **Available** list, and move them to the **In use** list.

The business object definitions listed in the **Link business object definitions** dialog box are the same configurable business objects that are listed in **Field Definer**.

If a business object is non-configurable, or, if it is currently under construction in **Field Definer**, it is not listed in the **Link business object definitions** dialog box.

3. Click OK. The business objects you selected are now linked to the new web service.

## Linking field definitions to a web service BO definition

After creating a web service definition and linking business object definitions to it, you can now specify fields for the business object definition.

### Procedure

1. In Web services, select a web service definition and go to Business object definitions.
2. Select the business object definition for which you want to specify the fields to be used by your new web service.
3. Go to Field definitions.
4. On the action panel, click Link field definitions.

   Select one or more field definitions from the **Available** list and move to the **In use** list.

   **The field definitions are linked to the business object definition.**
5. Click OK.

When the **Link field definitions** dialog box opens, a number of fields are already selected to be 'in use'. These fields are mandatory and if they are removed, you will not be able to generate your web service. In addition, the fields listed in the **Link field definitions** dialog box are the same as those in **Field Definer**, excluding auto-generated fields.

If you update the records via web services, you do not have to enter the mandatory field values again if you are not actually updating them. The field values are inserted automatically.

## Supported fields

This section describes which fields are supported for web services.

### General

A field is supported if the following conditions are met:

- The field is **In use**.
- A field type mapping exists.

- If a field is a *system type* field, it is supported (read-only) for retrieving data from the application. See also System type.

This feature is only available to support filtering. These are read-only fields, it will not be possible to enter/update values of an existing order.

- The field definition is not an association.

## Supported field types

- BigDecimalFieldType
- BooleanFieldType
- IntegerFieldType
- StringFieldType
- LongVarcharFieldType
- NeutralDateFieldType *
- NeutralDateTimeFieldType *
- NeutralTimeFieldType *
- DateTimeFieldType *
- Volume
- Area
- Length
- ObjectRaumFile
- CADViewerSymbol
- AutoCADFile
- Image
- Tariff per unit area (M2Tariff)
- FileRef
- PnNameUserDefined

* For DateTime fields, following formats are supported:

- EEE MMM dd HH:mm:ss zzz yyyy
- yyyy-MM-dd'T'HH:mm:ss
- yyyy-MM-dd'T'HH:mm:ss.000'Z' (including the time zone)

## System type

The *type* of field is sometimes required in order for interfaces to distinguish between similar fields that differ by their type only. (For example to discern a commitment cashflow from a commitment suppletion).

System types are supported except for:

- SysAuthorization
- SysDataSectionRef
- Syscode
- SysUpdateCount

## Linking actions to a web service BO definition

After creating a web service definition and linking business object definitions to it, you can also link actions to it.

### Procedure

1. In Web services, select a web service definition and go to Business object definitions.
2. Select the business object definition for which you want to specify the actions to be used by your new web service.
3. On the action panel, click Link actions.

   Select one or more actions from the **Available** list and move to the **In use** list.

Note that the actions will also include status transitions.

4. Click OK.

**The actions are linked to the business object definition. The actions linked to the web service can be seen in the Web service details selection level.**

The **Web service details** selection level will be active only if a business object is selected in the **Business object definitions** selection level.

## Generating, compiling and deploying a web service

You can generate, compile and deploy a web service definition after adding it to Planon ProCenter.

A .jar file named after the system name of the web service definition is created.

You can generate, compile and deploy a web service by performing the following steps:

## Generating a web service

### Procedure

1. Go to Web Service definitions.
2. In the elements list, select the web service definition you want to generate as a web service.
3. In the Package name field, change the name as required.
   For example, change the Package name as ‘com.planon’ with ‘Person’ as linked BO. By default, the system name is displayed as WS. {System name}.

   The Package name represents the folder structure in which the Web service definition is generated.
   In the above example, com.planon, the ‘planon’ folder is created in the ‘com’ folder.
4. On the action panel, click Generate web service. The web service files are downloaded as a zip file that will be available in your browser's download location.
   The zip file will contain the jars and all the other required files to make local adjustments to the web service. If you want to adjust the web service, you will need to compile it locally outside of Planon
   For this purpose, the zip file contains a build.txt file, which has instructions on how to do this.

The *src* folder contains a folder named in accordance with the respective web service definition. This folder contains subfolders for each of the business objects linked to the web service definition.

<source directory>

```
|- src/
   |- META-INF
   |   - services.xml
   |- ws/
          |- <wsdefinition name>/
      |- <BusinessObjectname >/
        |- <BusinessObject.java>
                 |- <BusinessObjectFilter.java>
                 |- <BusinessObjectService.java>
             |- PlanonSessionService.java
```

```
- test/
      |- ws/
 |- <wsdefinition name>/
     |- < BusinessObject name >
```

```
        |- <BusinessObject Client.java>
        |- <BusinessObject ClientStub.java>


    |- build.xml|
```

The *META-INF/services.xml* file has the required service definition for Axis2. It contains the information required to expose your business object as a web service.

Note that this file does not contain the WSDL of your web service.

The *src/ws/<wsdefinition name>* folder contains all the necessary web service source files which are placed in their respective folders (the folder name is same as that of the BO).

For example, if you generated a web service definition for Person BO:

- the *Person.java* file represents the actual business object (as POJO),

- the *PersonFilter.java* file is used for filtering on persons and

- the *PersonService.java* file provides the actual Web service: it defines the basic CRUD operations for adding, reading, deleting and updating persons in *Planon ProCenter.*

The PlanonSessionService.java file is required for logging on and off from the Planon ProCenter framework using Web services.

The *test/ ws/<wsdefinition name>* folder contains a sample client for your web service, for example,*PersonClient.java*, which is placed in its respective folder (the folder name is same as that of the BO), together with a utility class *PersonClientStub.java.*

You can use these sources to make a quick test for your web service using Java.

## Compiling a web service

After having generated the web service, you have to compile it.

### Procedure

1. Go to Web service definitions.
2. On the action panel, click Compile web service.

A jar file containing the compiled web service is downloaded to your browser's download location.

## Deploying a web service

After having compiled the web service, you have to deploy it. Deploying the web service needs to be done via the Axis2 web interface. The interface will allow inserting and updating of web services. In the Planon Cloud there is a designated WebDAV location where the web service files can also be uploaded to load / update a web service.

- Make sure the Web application setting **Enable web service console** is set to **Yes**. See also Web application fields.
- The default URL for the Axis2/Nyx interface is: http://hostname:port/nyx. When logging in, a Welcome page listing three links is displayed. After clicking the **Services** and **Administration** links, you are required to log in using the specific account credentials for Web Services. For Cloud, these credentials can be obtained via the Environment management gadget. When clicking the **Validate** link, an empty page is displayed.
- You can create multiple web service definitions but can deploy only a single web service definition at a time on a single instance of the web server.
- If you want to extend a web service you need to update the definition and compile a new .JAR-file. In Axis you need to deactivate and remove the old web service before you upload the new one.

## Verifying the deployed web services

Shortly after you have deployed your Web service, you can verify if it is up and running by using the path below: http://{servername or IP address}:{port number}/nyx/services/listServices.

Two new web services should be listed, for example, 'Person', along with a 'PlanonSession'. Also listed is the EPR (Endpoint Resource) for accessing your Web service from clients. Click on the service name-links to see the WSDL for that web service.

When you update the web services to a new version, you must compile, generate and deploy the web services once again. In case of a problem, contact your application manager.

Whenever changes are made in any of the involved business objects in Planon ProCenter, the web services must also be regenerated, compiled and deployed again to affect the changes via web services as well.

Whenever the web services APIs change, the web services need to be regenerated, recompiled and redeployed.

## Configuring axis2.xml

For each protocol (HTTP and/or HTTPS), an `AxisServletListener` instance must be declared in `axis2.xml`.

The `axis2.xmlfile` is found at the location:

`...\Server\tomcat-*\webapps\nyx\WEB-INF\conf`

If only a single protocol is used, no further configuration is required.
For example, if only HTTP is used, the following declaration must be present in `axis2.xml`:

```
<transportReceiver name="http" class="org.apache.axis2.transport.http.
```

```
        AxisServletListener"/>
```

If both HTTP and HTTPS are used, `AxisServlet` must know the ports used by HTTP and HTTPS to expose WSDLs with correct endpoint URLs. Typically, the servlet API doesn't allow a Web application to discover all the configured protocols. It only provides information about the protocol, host name and port for the current request.

If only a single `AxisServletListener` is configured, then this information is enough to let `AxisServlet` auto-detect the port number.

If both HTTP and HTTPS are used (or if WSDLs are retrieved through transports other than `AxisServlet`), then `AxisServlet` has no way of knowing the port numbers until it has processed at least one request for each protocol. To make WSDL generation predictable in this scenario, it is necessary to explicitly configure the port numbers in `axis2.xml`, such as in the following example:

```
<transportReceiver name="http" class="org.apache.axis2.transport.http.

        AxisServletListener">

<parameter name="port">8080</parameter>

</transportReceiver>

<transportReceiver name="https" class="org.apache.axis2.transport.http.

        AxisServletListener">

<parameter name="port">8443</parameter>

</transportReceiver>
```

# Developing a Web service client

Now your Web service is up and running, you can start developing clients that make use of the Web service. For an easy start, take a look at the files located in the test/ directory of your generated sources. The files provide a basic client for accessing a Web service from Java.

For other languages, or other frameworks, you need the WSDL of your Web service as a start for your client. For more information, refer to Verifying the deployed web services.

## Session management

This section points out a number of common specifications for web services.

### General

Web services can either access the application using a sessionID or an access key.

### sessionID

To reduce storing data in the PLN_SESSIONDATA table and improve performance, (stateless) in-memory sessionIDs are used for SOAP web services. Functionally, this means that sessions used by web services will no longer be visible in the **Session data** TSI.

General practice for web services is to ensure your own login and logout calls. This is needed because the web service must have a Planon session(ID) in order to work.

The standard use case is:

- Log in
- Execute work (for example, adding a number or persons to Planon).
- Log out

Instead of using a sessionID, you can also use an access key.

#### Session resources

When using a sessionID, the following resource restrictions/specifications need to be taken into account:

- Session timeout: to prevent web services from retaining too many resources, there is a timeout for the web service session when it has not been used for 24 hours (default).
- Limited BOValue cache on web services sessions to 16 entries
- Close SessionData upon logging out in web service
- Created a **SOAP session timeout** in System Settings > Web application TSI.

For more information, see Web application.

### Access key

Instead of executing a login SOAP request to create a sessionID and pass that as parameter to each request, you can also pass an access key.

This makes it possible to directly call SOAP web services by using an access key. It is then no longer required to call login / logout to fetch and release a sessionID.

- Using an access key could be slightly less efficient than using a sessionID.
- For more information, see Access keys.

## SessionID vs Access key

Whenever you want to set up a session using web services, a proper identification needs to be established.

This can either be done via creating a sessionID or via an access key.

This article describes how to do that for both ways of authentication.

### sessionID

When using a sessionID, you need to start by calling the following two endpoints:

- ../nyx/services/PlanonSession.PlanonSessionHttpsSoap11Endpoint/
- ../nyx/services/PlanonSession.PlanonSessionHttpsSoap12Endpoint/

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:per="http://
Person.ws">

  <soap:Header/>

  <soap:Body>
```

```
    <per:login>

      <!--Optional:-->

      <per:args0>[USERNAME]</per:args0>

      <!--Optional:-->

      <per:args1>[PASSWORD]</per:args1>

    </per:login>

  </soap:Body>

</soap:Envelope>
```

This will return:

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">

  <soapenv:Header/>

  <soapenv:Body>

    <ns:loginResponse xmlns:ns="http://Person.ws">

      <ns:return>[SESSIONID]</ns:return>

    </ns:loginResponse>

  </soapenv:Body>

</soapenv:Envelope>
```

Once you have retrieved the **sessionID** you can add that to the actual request, in this case the Read-BOM on the Person business object:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:per="http://
person.Person.ws">

  <soap:Header/>

  <soap:Body>

    <per:read>

      <!--Optional:-->

      <per:args0>[SESSIONID]</per:args0>

      <per:args1>[SYSCODE]</per:args1>

    </per:read>
  </soap:Body>

</soap:Envelope>
```

When using the sessionID, all requests are performed in the same session. When done, the session should be properly closed.

### Closing session

When using the **sessionID**, it is good practice to close the session by using the following request:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:per="http://
Person.ws">

  <soap:Header/>

  <soap:Body>

    <per:logout>

      <!--Optional:-->

      <per:args0>[SESSIONID]</per:args0>

    </per:logout>

  </soap:Body>

</soap:Envelope>
```

## Access key

If you want to use use an **access key**, simply replace the **sessionID** in the earlier request with the **access key**.

When using an access key, unlike with sessionID, a session is used per request - it is automatically opened and closed.

# Typical web service client operation

### Procedure

A typical sequence of operations performed by your web service client are:

1. Connect to the PlanonSession web service.

2. Log in with a valid user name and password to obtain a session identifier.

To be able to use Web services, make sure that the appropriate user group is linked to the *PPWS* product definition: *Web Services*. For more information, see product definitions (Accounts).

3. Connect to the actual web service.
4. Perform operations on this web service.
5. Connect to the PlanonSession web service.
6. Log out from Planon ProCenter.

- It is mandatory to log into the Planon ProCenter framework. It is the only way to obtain a valid session identifier.
- Each login session to Planon ProCenter via web services must be followed by a logout when the goal is complete. Doing so will prevent unnecessary memory usage by inactive web services as the session data will be cleaned up automatically after logout.
- Planon will trigger a session time-out if a web service is inactive for more than 24 hours.

## Searching/filtering for instances of business objects

It is possible to search for particular business objects with certain characteristics. For example, it is possible to look for all persons having a last name of 'Peters', or having the initials 'J.A'.

For this, a filter class is available, for example, PersonFilter. This class has setter methods for each field you can filter on. It is possible to filter on multiple fields at the same time. Searching on multiple fields is always done using ‘in’ and ‘like’ operators. For example, searching for all persons with initials 'J.A' and last name 'Peters' will only match the persons with the same initials and the same last name.

The setter methods in your filter class accept two parameters, the first parameter is the filter-operator to use, and the second parameter is the value to filter on. The following are the supported filter operators:

**Filters on exact matching values**

- "equals": the field value should be exactly equal to the filter value.
- "notEquals": the field value should not be equal to the filter value.
- "notEqualsOrEmpty": the field value should not be equal to the filter value or should be empty.

**Filters that contain a value**

- "contains": the field value contains the filter value.
- "notContains": the field value does not contain the filter value.
- "containsAValue": the field value contains any value.

**Filters that match the beginning of values**

- "startsWith": field value should start with the filter value.
- "notStartsWith": field value should not start with the filter value.

**Filters that match the end of values**

- "endsWith": the field value should end with the filter value.
- "notendsWith": the field value should not end with the filter value.

**Filter on values that are strictly greater than a value**

- "greater": the field value should be greater than the filter value.

**Filter on values that are greater than or equal to a value**

- "greaterEqual": the field value should be greater than, or equal to the filter value.

**Filter on values that are strictly less than a value**

- "less": the field value should be less than the filter field value.

**Filter on values that are less than or equal to a value**

- "lessEqual": the field value should be less than or equal to the filter field value.

**Filter on on empty values**

- notContainsAValue": the field value is null (empty).

Note that the operators are case-sensitive. For convenience, a Java class is defined with constants for these operators. This class is called ‘nl.planon.nyx.common.filter.FilterOperator'. It is preferred to use the constants of this class instead of the strings mentioned above.

However, it is not always possible to use all filter operators on all fields. For example, filtering on date fields starting with '12' will not work. In this case, an exception will be thrown to your Web service client.

## Filtering on field names

**You can use the actual field names while specifying the filtering criteria.**

For example, the **ParentRef** field on the Property business object in the Web Service definition.

However, for using this field for filtering, you must first add the field to the layout and put the field settings, **Inselection=true** and **Simple selection=true** in the FieldDefiner TSI.

The following example explains the settings for the ParentRef field:

```
PropertyFilter myFilter = new PropertyFilter();

    FieldFilter[] fieldFilters = new FieldFilter[1];

    fieldFilters[0]= new FieldFilter("","","");

    fieldFilters[0].setFieldName("ParentRef");

    fieldFilters[0].setFilterValue("301");

    fieldFilters[0].setOperator("equals");

    myFilter.setFieldFilters(fieldFilters);
```

## Filtering DisplaytypeRef fields:

You can use the **DisplayTypeRef** fields in the filtering as explained in the following example:

Add the **IsArchived** field to **Inselection=true** and ensure that it is added in the layout before you generate the webservices.

```
  FieldFilter[] fieldFilters = new FieldFilter[2];

    fieldFilters[0]= new FieldFilter();

    fieldFilters[1]= new FieldFilter();

    fieldFilters[0].setFieldName("ParentRef");

    fieldFilters[0].setFilterValue("301");

    fieldFilters[0].setOperator("equals");

    fieldFilters[1].setFieldName("IsArchived");

    fieldFilters[1].setFilterValue("true");

    fieldFilters[1].setOperator("equals");

    myFilter.setFieldFilters(fieldFilters);
```

## Java examples for using filters in webservice client:

```
// find all persons whose last name ends width “eters”:
PersonFilter filter1 = new PersonFilter();
```

```
filter1.setFilterLastName(FilterOperator.ENDS_WITH,
 “eters”);
```

```
// find all persons whose working address equals
// to the address with primary key (syscode) 25:
PersonFilter filter2 = new PersonFilter();
filter2.setFilterAddressRef(FilterOperator.EQUALS, 25);
```

```
// find all addresses whose free field 7 is less or equal to 141:
// (assuming free field 7 represents an integer field)
AddressFilter filter3 = new AddressFilter();
Filter3.setFilterFreeField7(FilterOperator.LESS_EQUAL, 141);
```

```
instance.find(final String aSessionId, Filter3);
```

### Java example for using the 'find method' in webservice client

```
// find all addresses whose free field 7 is less or equal to 141:
// (assuming free field 7 represents an integer field)
AddressFilter filter1 = new AddressFilter();
Filter1.setFilterFreeField7(FilterOperator.LESS_EQUAL, 141);

instance.find(final String aSessionId, filter1);
```

## Handling date and date-time fields

### Date fields

The current version of Axis2 cannot properly handle date fields that also contain a reference to time. Planon therefore recommends to only include the date in date fields.

Incorrect:

```
<xsd:beginDate>2017-07-24T14:42:14.000+02:00</xsd:beginDate>
```

Correct:

```
<xsd:beginDate>2017-07-24</xsd:beginDate>
```

### Date-time fields

To handle the time zones correctly **Property date-time** (DateTimeProperty) and **Neutral date-time** (DateTimeNeutral) fields values should be handled as string type.

When retrieving, populating or searching on dates, the date-time field should be in one of following formats:

- EEE MMM dd HH:mm:ss zzz yyyy
- yyyy-MM-dd'T'HH:mm:ss
- yyyy-MM-dd'T'HH:mm:ss.000'Z'

For more information on web services and time zones, see Web services and time zones.

## Modification date-time

For the majority of business objects, the **SysMutationDateTime** field is available and mandatory within Web Services.

For business objects for which **SysMutationDateTime** is not available through Web Service, you can use **SysChangeDateTime**.

If one or both of these fields are available for a business object, they can be used to find records based on the modification date-time of a business object record.

## Referring to other business objects from a web service

In many use cases for Web services, references need to be made from one business object to other business objects. In Planon ProCenter, support is added for reference fields in Web services.

References to other business objects are in most cases recognizable by the suffix “Ref” in field names, for example, “AddressRef”. Planon ProCenter has several representations of references, of which the largest part consists of integer references. Such an integer always points to the primary key of the referred business object.

For example, if a Person has a reference to working address 25, this actually means that Person refers to the business object Address with primary key 25.

All business objects in Planon ProCenter have a primary key by which they can be uniquely identified. When creating a new, unsaved, business object, the primary key is initially not set. It is assigned as soon as the business object is stored in Planon ProCenter.

A Java-example showing how to set the working address of a person to a newly created address is shown here:

```
// Create a new address
AddressClientStub addressClient = new AddressClientStub();
Address myAddr = addressClient.create( mySessionId );
// fill values of myAddr with relevant data (not shown)
myAddr.setAddress( “my Address” );
// store the newly created address
myAddr = addressClient.save( mySessionId, myAddr );

// Create a new person whose working address is ‘my Address’
PersonClientStub personClient = new PersonClientStub();
Person myPers = personClient.create( mySessionId );
// fill values of person with relevant data (not shown)
// set the working address of this person to the new address
myPers.setAddressRef( myAddr.getPrimaryKey() );
myPers = personClient.save( mySessionId, myPers );
```

If you want to link an existing business object, you need to find it first to obtain its primary key. It is common practice to not rely on primary keys as constant values in your code. For example, always search for the address to obtain its primary key, instead of relying on that its primary key is always 25.

You can create a new business object (dependent business object), which has a composite relationship with an existing primary business object (whole business object) using the method “ > create<USERBOTYPE>part”.

To understand more about Composite Business Objects, see Working with Composite BOs using web services .

## Reading and changing the status of a business object

The getState & setState methods are used to get and set the status of a business object, that is, these methods set the status transitions of the business objects. These new methods replace the previous set of methods whereby one method was used for each status transition. The sample code below shows you how to use the GetState and SetState methods to check the current status of any business object in Planon ProCenter.

```
final UsrMoveOrderClientStub instance = new UsrMoveOrderClientStub();

final String sessionId = instance.login( aArgs[0], aArgs[1] );

// Read the Order  using the primary key got
```

```
UsrMoveOrder aOrder = instance.read(sessionId,orderId[0]);

//get the current state

instance.getState(final aSessionId, aOrder);

// Set a State

String aTargetState= “UsrTechnicallyCompleted” ;

// UsrTechnicallyCompleted::System name of the target state

instance.setState(sessionId,aOrder,aTargetState);
```

## Status transition for orders based on a standard order

A standard order does not have a status itself, it has a **Status** field. Consequently, you cannot apply the getState and setState directly, but you will have to work around it.

For standard orders a **getStandardBOUserStatePnName** method is available that allows you to retrieve the standard order user status **PnName** of the status as set in the **RefBOStateUserDefined** field on the standard order.

This PnName can subsequently be used to set the status of an order created via web services to the desired state of the standard order, using the regular **setState** method.

**Example**

1. Create and save a new order.
2. Read all values from the standard order and set those values on the new order.
3. Read the standard order status using the new method.
4. Use getState on the order to retrieve the current status PnName.
5. Use getStandardBOUserStatePnName on the standard order to get PnName of desired status.

This method will only be added to web service when the **Read** action is linked on the standard order.

6. Use setState (current order status PnName, desired standard order status PnName).
7. Save the order.

# Adding time schedule to Maintenance Activity Definition in web services

You can set a schedule for time based Maintenance Activity Definitions through the Web services client. You can add recurring hourly, daily, weekly, monthly, yearly schedules using the following methods:

You must add the date in the 'Tue Nov 09 12:29:45 2010' format. If not an error message `<faultstring> Unparseable date: "Tuesday Nov 09 12:29:45 2010"</faultstring>` is displayed.

```
//Hourly Time schedule

void add_timeScheduleIterateHourly(final String aSessionId,

final MaintenanceActivityDefinition aPOJO, final Calendar aBeginDateTime,

final Calendar aEndDateTime, final int aInterval)

//Daily Time schedule

void add_timeScheduleIterateDaily(final String aSessionId,

final MaintenanceActivityDefinition aPOJO, final Calendar aBeginDateTime,

final Calendar aEndDateTime, final int aInterval)

//Monthly Time schedule

void add_timeScheduleIterateMonthly(final String aSessionId,

final MaintenanceActivityDefinition aPOJO, final Calendar aBeginDateTime,

final Calendar aEndDateTime, final int aInterval, final int aDayOfMonth)

//Yearly Time schedule

void add_timeScheduleIterateYearly(final String aSessionId,

final MaintenanceActivityDefinition aPOJO, final Calendar aBeginDateTime,

final Calendar aEndDateTime, final int aInterval)

//Monthly Weekdays Time schedule

void add_timeScheduleIterateMonthlyOnWeekDay(final String aSessionId,

final MaintenanceActivityDefinition aPOJO, final Calendar aBeginDateTime,

final Calendar aEndDateTime,

final int aInterval, final int aOccuranceInMonth,

final boolean aOnMonday, final boolean aOnTuesday, final boolean aOnWednesday,

final boolean aOnThursday, final boolean aOnFriday,

final boolean aOnSaturday, final boolean aOnSunday)

//Weekly Time schedule
```

```
void add_timeScheduleIterateWeekly(final String aSessionId,

final MaintenanceActivityDefinition aPOJO, final Calendar aBeginDateTime,

final Calendar aEndDateTime, final int aInterval, final boolean aOnMonday,

final boolean aOnTuesday, final boolean aOnWednesday,

final boolean aOnThursday,final boolean aOnFriday,

final boolean aOnSaturday, final boolean aOnSunday).
```

For a description of the parameters used in the methods, refer to Parameters.

## Adding person type reference field

In Planon ProCenter, 10 person type references are available. Each person type reference corresponds to a digit from 0 to 9.You can add a Person type reference field in the Person BO by using the

```
setPersonTypeRef(final String aNewValue)method.
final PersonClientStub instance = new PersonClientStub();

final String sessionId = instance.login( aArgs[0], aArgs[1] );

// TODO add your business logic here...

Person person = instance.createPerson(sessionId);

person.setCode("test1");

person.setFacilityNetUsername("test1");

// valid input

person.setPersonTypeRef("1 3 5");
```

In the above example of setting person type reference as”1 3 5”, empty spaces are provided between the digits. The empty spaces correspond to the missing digits thus enabling you to display only the person type references of the digits mentioned here. That is, if 1, 2 and 3 digits correspond to Coordinator intern, Uitvoerder intern and Melder person types respectively, according to the above example, only the Coordinator intern and Melder are displayed while Uitvoerder intern becomes invisible as 2 is represented by an empty space.

While assigning person type references in the method, if continuous numbers are not used and if empty spaces are not provided between the digits, incorrect person type reference fields will be displayed.

# Working with Composite BOs using web services

Composite Business Objects are those that either have a dependent (part) or a primary (whole) relationship with another BO.

A primary – dependent relationship is a relationship where the instance of the dependent business object cannot exist without the existence of an instance of the primary object.

In Planon ProCenter, in **Field definer**, the **Technical** information tab for any business object informs the user if the business object is composite or not.
To create an instance of a BO which is dependent in a composite relationship using web services in Planon ProCenter, the web services exposes a method to create the instance of the dependent BO from the instance of the primary BO.

For example, to create a work order which is a dependent in the whole part relationship with the base order, you must use this method and pass the primary key of the base order type, the session id and the partBusinessObject instance.

```
/**

 * Creates a new, unsaved UsrWerkorderalgemeen part instance.

 */

public UsrWerkorderalgemeen createUsrWerkorderalgemeenPart

     (final String aSessionId, final int aBaseOrderPrimaryKey,

      final UsrWerkorderalgemeen aNewPartPOJO)throws AxisFault {
```

# Filtering on Person type

The following is true for the **Person type** field which supports 'contains' operator:

The field PersonType supports 'contains' operator. User must specify the right 'String' for the person type. Person Type being a special field, gets stored in the database as string made of numbers 0,1,2,3,4,5,6,7,8,9 where each digit specifies the selection of the particular type.
That is, if Type 1 and 3 are selected, '1 3' get stored in database. However. you must ensure that you specify the same string in the filter parameter.

For example, if you like to search the Person type A and C in the database which will be stored in a pattern like(1 3), then ABCD will be 1234.
The following example gives you all the person types having person type selected A and C:

```
<xsd:fieldFilters>

    <!--Optional:-->

    <xsd1:fieldName>PersonTypeRef</xsd1:fieldName>

    <!--Optional:-->

    <xsd1:filterValue>1 3</xsd1:filterValue>

    <!--Optional:-->

    <xsd1:operator>contains</xsd1:operator>

</xsd:fieldFilters>
```

# Supported Methods in web services

Web services support the **C**reate, **R**ead, **U**pdate, **D**elete (CRUD) actions.

When a user generates a web service, the <BOName>Service.java file is generated. This file lists all the methods that are supported by web services for this BO.

The generated methods also depend on the user authorization done on the BO.

The following table lists the methods supported by the web services when a BO is exported from Planon ProCenter:

| Method in WSDL | Action in Planon Procenter |
|---|---|
| `Create<BOName>`<br><br>**Example: public Person** createPerson**(final String**aSessionId**)** | BomAdd |
| `create<BOName>Part`<br><br>**Example: public UsrWerkorderalgemeen** createUsrWerkorderalgemeenPart**(final String** aSessionId**, final int** aBaseOrderPrimaryKey**, final UsrWerkorderalgemeen** aNewPartPOJO**)** | BomAddPart<br><br>This method is used to create a business object which is a dependent one in a composite relationship using web services in Planon ProCenter. |
| `create<BOName>( <parameter set>)`<br><br>**Example:** public CommunicationLog createCommunicationLogWithArguments (final String aSessionId, final Integer aSyscode, final String aBOType)<br><br>**Example:** public UsrReservering createUsrReserveringWithArguments (final String aSessionId, final Date aBeginDateTime, final Date aEndDateTime, final Integer aReservationUnitRef, final Integer aPersonCount, final Integer aDeskConfigurationRef) | BomAdd<br><br>You can also use the create methods, with Arguments.<br><br>Only the BOs that require parameters passed as part of the create (ADD BOM) must use Create method(With Arguments). If the parameters are not passed correctly while executing this method, an error message occurs and the system shuts down. |

| Method in WSDL | Action in Planon Procenter |
|---|---|
| Return type: Object.<br>**Input parameters:** String SessionId, Integer aSyscode, String aBOType | |
| Read<br>**Example: public** Person read(**final String** aSessionId, **final int** aPrimaryKey) | BomRead |
| Save<br>**Example: public** Person save(**final String** aSessionId, **final Person** aPOJO) | BomSave |
| Delete<br>**Example:public Boolean** delete(**final String** aSessionId**, final int** aPrimaryKey) | BomDelete |
| Find<br>**Example:** public int[] find(final String aSessionId, final PersonFilter aFilter) | Quick search in Planon ProCenter.<br>For more information, see Java example for using the 'find method' in webservice client |
| getState<br>**Example:** public String getState(final String aSessionId, final Person aPOJO) | Gets the current state |
| setState<br>**Example:** public Person setState(final String aSessionId, final Person aPOJO, final String aTargetState) | Sets the status of an instance of business object from its current state to its target state. It can set system statuses as well as user statuses depending on the business object type. |
| connectTo<Business Object> | Creates M-to-N relationships between two business objects. |

| Method in WSDL | Action in Planon Procenter |
|---|---|
| **Return type:** void<br>**Input parameters:** Session ID, BOPrimaryKey, BOPrimaryKeyN. | |
| `disconnectFrom<Business Object>`<br>**Return type:** void<br>**Input parameters:** Session ID, BOPrimaryKey, BOPrimaryKeyN. | Disconnects M-to-N relationships between two business objects. |
| `connectToBom<Link.SystemName>`<br>**Example:** public Boolean connectToUsrSkills6897(final String aSessionId, final int aPrimaryKey, final int aPrimaryKeyN) | Marks all the M-to-N link actions for a business object.<br>The Link.SystemName is mentioned in the business object's **Links** tab in **Layouts** TSI. |
| `disconnectfromBom<Link.System-Name>`<br>**Example:** public Boolean DisconnectFromUsrSkills6897(final String aSessionId, final int aPrimaryKey, final int aPrimaryKeyN) | Unlinks the M-to-N actions on a business object. |
| `setSessionDataSection`<br>**Return type:** Boolean.<br>**Input parameters:** String SessionId, String DataSection. | Changes the property set for a user through web services. |
| `setReferenceDate`<br>**Return type:** Boolean<br>**Input parameters:** String SessionId , java.util.Calendar. | Sets the reference date of a reference date aware business object. This method is available on PlanonSessionService.java. |

| Method in WSDL | Action in Planon Procenter |
|---|---|
| `logout`<br><br>**Return type:** Boolean<br><br>**Input parameters:** String SessionId | Logs out a Planon session. |
| `createSubInventoryItem`<br><br>**Return type:** InventoryItem.<br><br>**Input parameters:** String SessionId, Object InventoryItem. | Creates a sub-asset on the asset indicated by the parameter. |
| `archive`<br><br>**Example: public Boolean** archive(**final String** aSessionId**, final int** aPrimaryKey) | Archives a business object. |
| `dearchive`<br><br>**Example: public Boolean** dearchive(**final String** aSessionId**, final int** aPrimaryKey) | Dearchives a business object. |
| `addTimeAwareMToN` | Additional method that only applies to business objects with time-aware M:n links in Service Providers solution mode!<br><br>Adds a time-aware M:n record (link to another business object). |
| `endTimeAwareMToN` | Additional method that only applies to business objects with time-aware M:n links in Service Providers solution mode!<br><br>Sets an end date for a time-aware M:n record (link to another business object). |

BaseOrder BO has **Save** and **Delete** methods.

# Web services and time zones

When time zones are **not** enabled, any date-time entered with a web service is processed as if the date-time is entered in the time zone of the web server. Any time zone specified in the input date-time field is stripped/ignored.

When times zones **are** enabled, any date-time entered with a web service is processed in the time zone of the logged in user (the time zone that is registered for that user in Planon). Any time zones specified in the input date-time field are stripped/ignored. The web/application server time zone is irrelevant, the database time zone, as specified in Planon **System Settings**, is used to determine how the values are stored in the database.

**Examples**

There are three different date-time types:

- Property date-time
- Transaction date-time
- Neutral date-time

| Type | Base Value | Time Zone |
|---|---|---|
| Property | Integer | Property |
| Transaction | Date | System |
| Neutral | Integer | Not specified |

**Configuration**

| Time Zone | Location | Acronym | Offset |
|---|---|---|---|
| Property | USA / New York | EST | UTC -05 (UTC -04 with DST) |
| User 1 | Pacific/Tahiti | TAHT | UTC -10 |
| User 2 | London | UTC | UTC + 0 |

Legend:

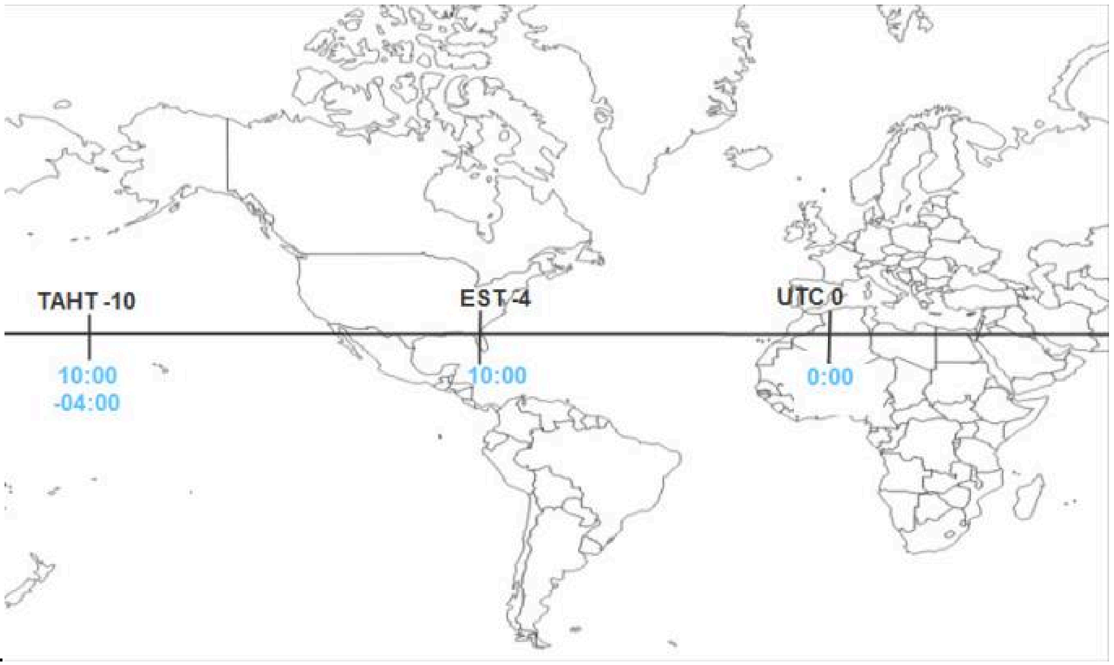
- EST: Eastern Standard Time
- TAHT: Tahiti Standard Time
- UTC: Coordinated Universal Time

- DST: Daylight Saving Time

## Property date-time

- Property is in the EST time zone.
- User 1 is used to log in with web services, to create the data and then check the user interface for the result.
- User 2 is used to read the data (in the user interface / web services).

The date is always returned in the property time zone (-04 is property time zone USA/ New York)



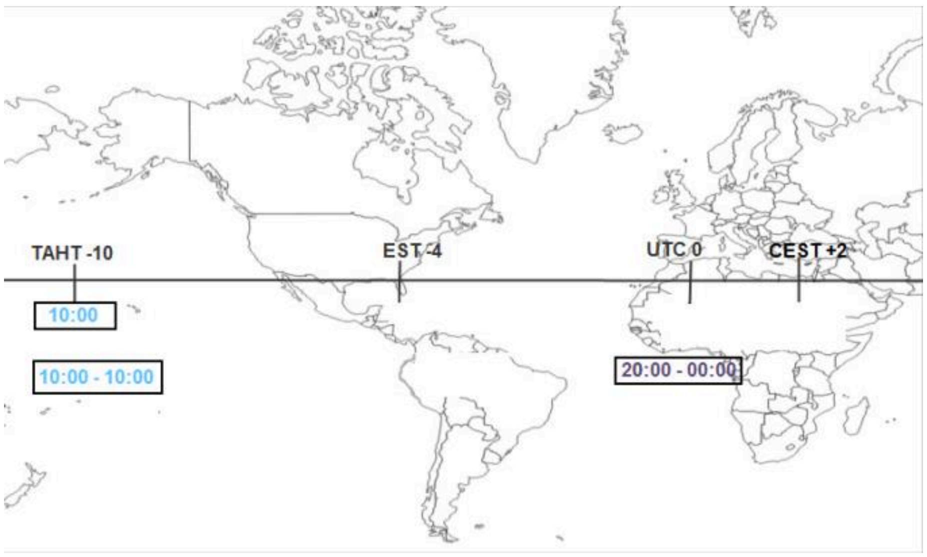
.

| | |
|---|---|
| Web service input user 1 | 2015-10-21T**10:00**:00 |
| Web service reply to user 1 | 2015-10-21T **10:00**:00.000 **-04:00** (the date is returned in the property's time zone) |
| User interface displays to user 1 and user 2 | 2015-10-21 10:00:00.000 |

In the user interface, the property date-time is always the same.

## Transaction date-time

Date is exported in time zone of the user (-10 is user time zone TAHT).

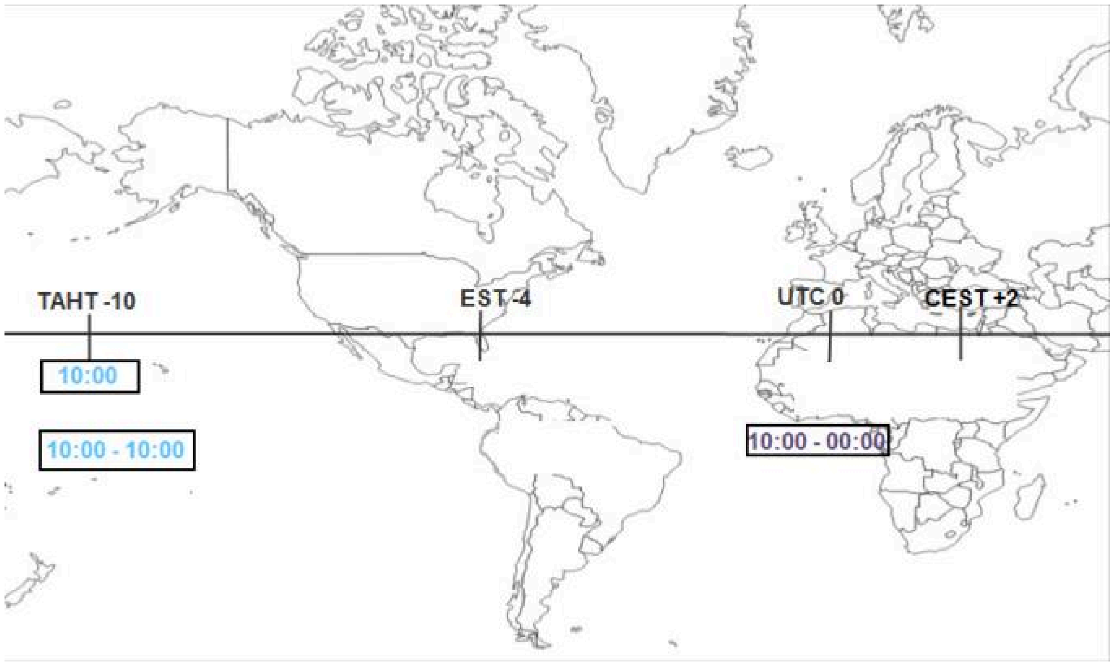


| | |
|---|---|
| Web service input user 1 | 2015-10-21T**10:00**:00 |
| Web service reply to user 1 | 2015-10-21T**10:00**:00.000 **-10:00** (the date is returned in the user's time zone) |
| User interface displays to user 1 | 2015-10-21 **10:00**:00 |
| User interface displays to user 2 | 2015-10-21 **20:00**:00 |

## Neutral date-time

The date is exported with the time zone of the user.

The time is always the same, the time zone is different (based on the user time zone).



| | |
|---|---|
| Web service input user 1 | 2015-10-21T**10:00**:00 |
| Web service reply user 1 | 2015-10-21T**10:00**:00.000 **-10:00** (The date is returned in the user's time zone) |
| User interface displays to user 1 and user 2 | 2015-10-21 10:00 |

# BO types and their methods

The table below lists some examples of Business objects for specific BO types with their typical methods:

| BOType | connectTo | create | createPart | delete | getState |
|---|---|---|---|---|---|
| | | | **Method** | | |
| System BO without subtypes without states | | | | | |
| Department | Y | Y | Y | Y | - |
| PSSActivityInventory-ItemSubject | Y | Y | - | Y | - |
| System BO with states without subtypes | | | | | |
| Visitor | Y | Y | Y | Y | Y |
| System BO with subtypes without states | | | | | |
| BasePSSActivitySubject | Y | - | - | Y | - |
| System BO with userDefined subtype with states | | | | | |
| Person | Y | Y | - | Y | Y |
| TransactionPlan | Y | Y | Y | Y | Y |
| UsrProject | Y | Y | Y | Y | Y |
| InventoryItem | Y | Y | Y | Y | Y |
| UserDefined BO with states | | | | | |
| UsrPerson | Y | Y | - | Y | Y |
| UsrInventoryItem | Y | Y | Y | Y | Y |
| BaseType BO | | | | | |
| Teams | - | - | - | Y | - |

| BOType | connectTo | create | createPart | delete | getState |
|---|---|---|---|---|---|
| System BO that is subtype of a BaseType BO and no userDefined subtypes | | | | | |
| ProjectTeams | Y | Y | Y | Y | Y |
| System BO with states with subtypes | | | | | |
| BaseProjects | - | - | - | Y | Y |
| System BO that is subtype of a BaseType BO with userDefined subtypes | | | | | |
| Project | - | Y | Y | Y | Y |
| **Orders** | | | | | |
| BaseOrder | | | | | |
| BaseOrder | - | - | - | Y | - |
| System BO that is subtype of a BaseOrder | | | | | |
| WorkOrder | - | - | - | Y | Y |
| UserDefined BO that is subtype of a subOrder | | | | | |
| UsrWerkorder algemeen | Y | Y | Y | Y | Y |

| BOType | read | save | setState | setState-<ToState> | Disconnect-From |
|---|---|---|---|---|---|
| **Method** | | | | | |
| System BO without subtypes without states | | | | | |
| Department | Y | Y | - | - | Y |
| PSSActivityInventory-ItemSubject | Y | Y | - | - | Y |
| System BO with states without subtypes | | | | | |

| BOType | read | save | setState | setState-<ToState> | Disconnect-From |
|---|---|---|---|---|---|
| Visitor | Y | Y | Y | - | v |
| System BO with subtypes without states | | | | | |
| BasePSSActivity-Subject | Y | Y | - | - | - |
| System BO with userDefined subtype with states | | | | | |
| Person | Y | Y | Y | - | Y |
| TransactionPlan | Y | Y | Y | D | Y |
| UsrProject | Y | Y | Y | - | Y |
| InventoryItem | Y | Y | Y | D | Y |
| UserDefined BO with states | | | | | |
| UsrPerson | Y | Y | Y | - | Y |
| UsrInventoryItem | Y | Y | Y | - | Y |
| BaseType BO | | | | | |
| Teams | v | - | - | - | - |
| System BO that is subtype of a BaseType BO and no userDefined subtypes | | | | | |
| ProjectTeams | Y | Y | Y | - | Y |
| System BO with states with subtypes | | | | | |
| BaseProjects | Y | Y | - | - | - |
| System BO that is subtype of a BaseType BO with userDefined subtypes | | | | | |
| Project | Y | Y | Y | - | - |

**Orders**

| BOType | read | save | setState | setState-<ToState> | Disconnect-From |
|---|---|---|---|---|---|
| BaseOrder | | | | | |
| BaseOrder | Y | Y | - | - | - |
| System BO that is subtype of a BaseOrder | | | | | |
| WorkOrder | Y | Y | - | - | - |
| UserDefined BO that is subtype of a subOrder | | | | | |
| UsrWerkorder algemeen | Y | Y | Y | - | Y |

## Method Parameter Explanation

The following list provides an overview and description of the available parameters that can be used.

| Parameter | Description |
|---|---|
| aSessionId | Indicates the session id for the logged in user session. |
| MaintenanceActivityDefinition | Provides an instance of a Maintenance Activity Definition. |
| aBeginDateTime | Begin date & time of the schedule. This parameter also accepts 'null' as value. |
| aEndDateTime | End date & time of the schedule. This parameter also accepts 'null' as value. |
| aInterval | Indicates the number of times the scheduler is called during the schedules. |
| aDayOfMonth | Indicates the day in a month when the scheduler should be executed. |
| aOccuranceInMonth | Indicates the occurrences in a month. |
| aOnMonday | This value should be true if you want to schedule the occurrence on a Monday. |

| Parameter | Description |
|---|---|
| aOnTuesday | This value should be true if you want to schedule the occurrence on a Tuesday. |
| aOnWednesday | This value should be true if you want to schedule the occurrence on a Wednesday. |
| aOnThursday | This value should be true if you want to schedule the occurrence on a Thursday. |
| aOnFriday | This value should be true if you want to schedule the occurrence on a Friday. |
| aOnSaturday | This value should be true if you want to schedule the occurrence on a Saturday. |

# Frequently asked questions

**Q**: The Web server complains about not being able to find the class ‘nl/planon/util/pnlogging/PnLogger.class' or ‘nl/planon/hades/valueobject/IBOValue.class'.

**A**: Make sure the Axis2 directory in ...\Server\tomcat-*\webapps is removed prior to deploying a new Axis2.war file. If you do not do this, your updated WAR-file will not be redeployed and thus your changes are not visible.

**Q**: I'm working with Axis2 and Java, but my Web service crashes without much information. How can I get more debug information?

**A**: Update the ‘log4j.properties' in your Axis2 installation directory. Replace the line "log4j.rootCategory = INFO, CONSOLE" with "log4j.rootCategory = DEBUG, CONSOLE" an restart your client to see verbose information about what it is doing.

**Q**: I try to run my Axis2, but it keeps complaining about classes not being found! What should I do?

**A**: If you want to run your Java Web service client using Axis2, you need to add (almost) all Axis2 JAR-files to the classpath. For your convenience, you can use the following batch-file:

-- snip --

```
@echo off

set AXIS2_HOME=c:\path\to\axis2

set JAVA_HOME=c:\path\to\java

set WS_CLIENT_JAR=c:\path\to\myclient.jar

set WS_MAIN_CLASS=nl.test.ws.MyClient

setlocal EnableDelayedExpansion

%JAVA_HOME%\bin\java" -cp "%WS_CLIENT_JAR%;%AXIS2_HOME%*"
 %WS_MAIN_CLASS% %1 %2
```

-- snip --

Save the above snippet into a file ending with the `.bat' extension, for example, ‘run.bat', and adjust the first three set-commands to your situation. Note that lines ending with ‘#’ are continued on the following line, and should result in a **single** line in your batch file;

**Q**: I want to make Web service calls from a client, how should I do this?

**A**: Take a look at the sample code generated; you can use the Stub-named class directly to talk to your Web service. You do not need to instantiate the Service-classes yourself! This is done by the Web service container, and should not be done at the client.

# References

- Apache Axis2 -
- Java JDK -
- Apache Axis2 Web administration guide -
- Apache ANT -

# Index