Dollar Universe Web Services v2.1

User Manual

for Windows and UNIX

EMEA HEADQUARTERS

Tour Franklin
92042
Paris La Défense Cedex
France

+33 [0] 1 47 73 12 12

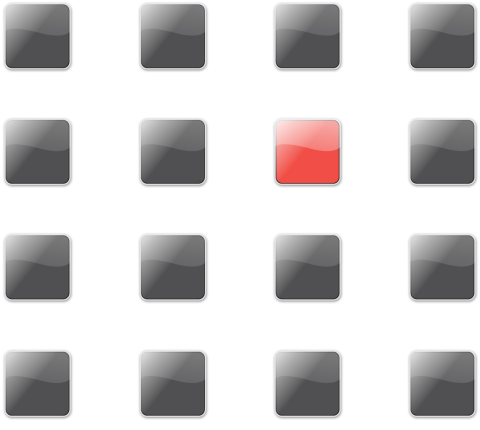
info@orsyp.com
www.orsyp.com

AMERICAS HEADQUARTERS

300 TradeCenter
Suite 5690
Woburn, MA 01801
USA

+1 [781] 569 5730

info_usa@orsyp.com
www.orsyp.com



April 1, 2011

Contents

Figures

1. Introduction

1.1 Who Should Read this Manual

This manual addresses the needs of the person(s) who will install, configure and use Dollar Universe Web Services.

Dollar Universe Web Services are intended for developers and IT consultants who need to provide end-users with customized tools related to their business processes.

1.2 What this Manual Contains

This manual details the installation, configuration and use of Dollar Universe Web Services.

1.3 Related Information Sources

It is assumed the reader is familiar with Dollar Universe and its concepts together with Web servers and HTTP protocol, SSL, XML, WSDL, UDDI, SOAP protocol through Visual Basic, C++, C#, Java programming or any other language chosen for the development of client applications.

Source Name
Dollar Universe Reference Manual

1.3.1 Organizations and Consortia

World Wide Web consortium (W3C): http://www.w3.org/

OASIS consortium: http://www.oasis-open.org/home/index.php/

1.3.2 Web Services Technology

Extensible Markup Language (XML): http://www.w3.org/XML/

Simple Object Access Protocol (SOAP): http://www/w3.org/TR/SOAP/

Web Services Description Language (WSDL) 1.1: http://www/w3.org/TR/wsdl.html/

Universal Description, Discovery and Integration (UDDI): http://uddi.org/specification.html

1.3.3 Java and XML Technology

The Java Language Specification, Second Edition: http://java.sun.com/docs/books/jls/index.html

JSR-000101 Java™ APIs for XML-based RPC:

http://jcp.org/aboutJava/communityprocess/first/jsr101/index.html

JSR 000109: Implementing Enterprise Web Services

JSR 000921: Implementing Enterprise Web Services 1.1

JSR 000151: Java™ 2 Platform, Enterprise Edition 1.4 (J2EE 1.4) Specification

JSR 000154: Java™ Servlet 2.4 Specification

1.3.4 Servlet Containers, Web Container and J2EE Technology

J2EE 1.4 Compatible Implementations: http://java.sun.com/j2ee/compatibility.html/

Java 2 Enterprise Edition (J2EE): http://java.sun.com/j2ee/

1.4 Where to Get Help

1.4.1 Customer Support

You can obtain technical support by using the support page on the ORSYP Website or by contacting our customer support by phone.

1.4.1.1 Technical Support Web Site

You can obtain 24/7 technical support at http://www.orsyp.com/support.

From the ORSYP Web site, you can:

- Report a problem or ask a question
- Upload traces or logs
- Consult the support knowledge base
- Consult support “Tips&Tricks”
- Find the latest information about ORSYP products (supported and recommended product versions)

- Download product documentation and product patches
- Subscribe to email Support Newsletters

1.4.1.2 Telephone Support

- USA: 866 257 7090
- Canada: 877 809 5040
- France: (+33) 1 47 73 02 82
- UK: 0800 731 4208
- Germany: 0800 181 3342
- Italy: 800 129 076
- Spain: (+34) 902 010 760
- Hong Kong: +852 2575 5933
- Singapore: (+65) 64949 258
- Other Countries: (+33) 1 47 73 08 95

1.4.2 ORSYP Forum

http://www.orsypforum.com

This site is available to all users of ORSYP and SYSLOAD Products. The goal is to offer ORSYP customers worldwide functional assistance, thus complementing the services provided by ORSYP Technical Support , and ORSYP Professional Services.

All members are invited to share experience, knowledge, best practices and solutions related to **functional** topics (technical issues must be addressed to ORSYP Technical Support).

1.5 Typographical Convention

Category	Example
Recommendations, Notes and Warnings	**Tip:** Information to facilitate setup and use **Note:** Additional information to better understand **Important:** Information to alert you to situations that can cause problems if you do not follow instructions carefully
Examples	*Examples:* *…* *Here you will find a specific case in point to illustrate what was explained above.*
Buttons, Menu Items, Command Names	Right click and select **Add Filter** **Start > Programs > Sysload > SP Analyst Console > SP Analyst Console** Use the **uxrepadd** command to…
File Names & Extensions	Edit the file <*Sysload*>*Console**sld.ini,* section "EXCD" Select a file with the extension *.pjr* for reports of type "Elements to watch" **Note:** Variable elements in a file path are quoted by < and >.
File Contents and Command Line Help	`[GENERAL]` `Hosts=C:\Program Files\Sysload\Console\Hosts.ini`
Command Line, File Path Syntax & Message Texts	**<path>\<file_name>.<extension>**

2. Overview

2.1 Web Services

2.1.1 Definitions

Web Services make software applications available across the World Wide Web, providing business logic to client applications.

Web Services respect the standard communication mechanisms published by the World Wide Web Consortium (W3C). Such standards allow customer applications to invoke Web Services over Internet or intranet networks whatever the operating systems and platforms of the consumer application and the invoked services. Consult the W3C web site for more details: http://www.w3.org/.

Web Services are defined by a list of public interfaces which provide the functionality and are located by an URI (Uniform Resource Identifier). The description of a Web service is contained in a WSDL file (Web service Description Language) in XML format.

Application servers generally accommodate Web Services that can be discovered with the help of public or private UDDI registries.

Communications between client applications (that can themselves be Web Services) and remote Web services are based on protocols which use XML-based messages; the most common of which is SOAP (Simple Object Access Protocol). SOAP defines a standard for sending and receiving messages over the Internet using the HTTP/HTTPS protocol.

2.1.2 Dollar Universe Web Services

Dollar Universe Web Services is the service-oriented interface to the ORSYP job scheduler Dollar Universe. Dollar Universe Web Services make it possible to easily interconnect Dollar Universe job flow management capabilities to end-user or business processes.

With Dollar Universe Web Services you can create your own applications to monitor and manage IT operations on Dollar Universe instances.

The following figure depicts Dollar Universe Web Services architecture, providing access to Dollar Universe for IT operations monitoring.

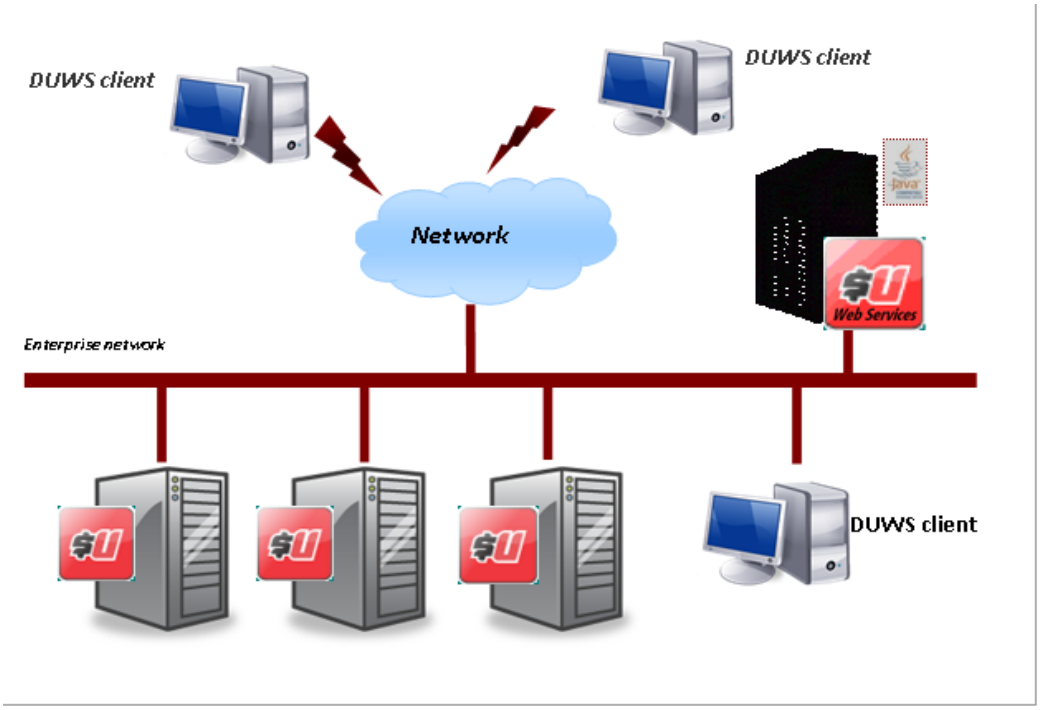


Figure 1: Dollar Universe Web Services for operations monitoring architecture

2.2 Dollar Universe Web Service Components

The components involved in the architecture are:

- An application server running a server-side JAX-RPC runtime, providing full support for both clients and Web Service endpoints (not delivered by ORSYP); for instance, a J2EE 1.4 (or higher) compliant application server
- Dollar Universe Web Services delivered by ORSYP
- Dollar Universe standard installations including I/O servers

A Dollar Universe end user request will follow the route:

- A SOAP message is sent by the client application to an HTTP (or HTTPS) port on the application server hosting Dollar Universe Web Services
- The request is interpreted by the Web server and forwarded to the Web Services container
- The Web service container routes the request to the appropriate Dollar Universe Web Services endpoint
- The Web service contacts the appropriate I/O (data) Server. The action is received by the I/O (data) Server and performed by Dollar Universe components. The invoked Web Service builds the SOAP response with the received results and returns the message to the invoker.

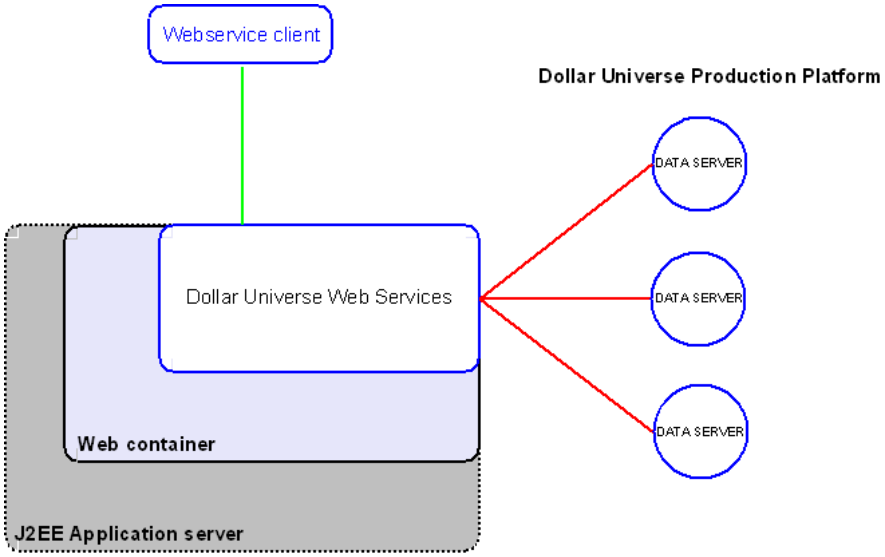


Figure 2: Dollar Universe Web Services installation schema

3. Installation

3.1 Prerequisites

Dollar Universe Web Services requires **Dollar Universe version 5.1** with the latest patch.

It is assumed that a **J2EE 1.4 application server** is already installed and operational.

3.2 Media

Dollar Universe Web Services is provided in the form of an EAR (Enterprise Archive) file: **duws210.ear**.

The Dollar Universe Web Services Enterprise Archive (EAR file) respects the J2EE specification.

3.2.1 Contents of the duws210.ear

The EAR file contains a Web application module (Web archive) with the following contents:

```
duws
duws/meta-inf
duws/meta-inf/Manifest.mf
duws/WEB-INF
duws/WEB-INF/web.xml
duws/WEB-INF/webservices.xml
duws/WEB-INF/lib
duws/WEB-INF/lib/api.jar
duws/WEB-INF/lib/art.jar
duws/WEB-INF/lib/central-core.jar
duws/WEB-INF/lib/comm.jar
duws/WEB-INF/lib/conf.jar
duws/WEB-INF/lib/stdimpl.jar
duws/WEB-INF/lib/util.jar
duws/WEB-INF/lib/wsaccess.jar
duws/WEB-INF/lib/wsduws.jar
duws/WEB-INF/lib/wsutil.jar
duws/WEB-INF/wsdl
duws/WEB-INF/wsdl/DuwsService.wsdl
duws/WEB-INF/wsdl/duwsservice-mapping.xml
```

3.3 Deployment

Dollar Universe Web Services should be deployed with the specific Web Services for J2EE product provider tool.

Follow your IT department's deployment procedures to configure and deploy Dollar Universe Web Services.

The following steps must be performed however:

3.3.1 Configure the duws210.ear

- Copy the **duws210.ear** file to a working directory. This copy of **duws210.ear** will be configured before deployment by the application server.
- Open **duws210.ear** with the deployment tool.
- Update the context parameter **configFile** in the ***web.xml*** file. This parameter contains the absolute path to the *Dollar Universe Web Services configuration file*. Replace the string *ENTER_YOUR_PATH* with the correct path of the configuration file (e.g. **C:/DUWS/duws.conf)**

3.3.1.1 Setting the Configuration File Name

The configuration file should be completed before attempting to start the web service (please refer to section The web service configuration file for details of the configuration file contents).

```
<context-param>
<description>Configuration file</description>
<param-name>configFile</param-name>
<param-value>C:/DUWS/duws.conf</param-value>
</context-param>
```

- Configure the application server's *security policy* to determine which users are authorized to invoke the Web service. By default the Web service can be invoked only by users with the **DollarUniverseUser** role.
- Save the modifications.

The file **duws210.ear** is now ready to be deployed.

3.3.2 Deploy the duws210.ear

With the application server's administration tool, deploy the **duws210.ear** file updated with the deployment tool.

3.3.3 Check Web Service Reachability

Open a Web browser and enter the URL:

http://<host>:<port>/services/duws?WSDL

Where:

- **host** is the hostname of the Web server,
- **port** is the TCP/IP port used by the Web server to receive HTTP requests.

The browser will prompt for a login and a password since BASIC authentication is required. It is assumed that the current user was given permission to use the web service during the *security policy* configuration step.

The Web server responds by displaying the WSDL file (see the file duws/WEB-INF/wsdl/DuwsService.wsdl). However the element:

<soap:address location="…" …>

must contain the effective address location of Dollar Universe Web Services.

Note: The actual URL may differ with the context. For instance, if the underlying web service engine is AXIS and default deployment settings are used, the URL will be:

http://<host>:<port>/axis/services/duws?WSDL

---

4. Configuration

4.1 The Web Service Configuration File

The web service uses a configuration file whose name and address are held in the *context parameter* **configFile**. This file should contain the following properties:

```
com.orsyp.duws.proxy.node    = <proxy node>
com.orsyp.duws.log.file      = <Log file>
com.orsyp.duws.log.level     = <Trace level>
com.orsyp.duws.lic.file      = <Licenses file>
com.orsyp.duws.services.file = <Service mapping file>
com.orsyp.duws.nodes.file    = <Node/hostname mapping file>
com.orsyp.duws.joblog.tempdir = <Job log temporary Directory>
```

Where:

Property name	Values
com.orsyp.duws.proxy.node	The logical (Dollar Universe) node name on which Dollar Universe Web Services was installed. This name must not be more than10 characters long.
com.orsyp.duws.log.file	Path to the Dollar Universe web service log file.
com.orsyp.duws.log.level	The trace level. Permitted values are 0, 1, 2, 3 or 4. 0 = none, 1 = error, 2 = info, 3 = debug, 4 = trace.
com.orsyp.duws.lic.file	The name and address of the license file.
com.orsyp.duws.services.file	The file containing the access port definitions for the I/O servers. This may point to the system's TCP/IP services file (e.g. /etc/services) or to a standalone file.
com.orsyp.duws.nodes.file	Node name to hostname resolution file.
com.orsyp.duws.joblog.tempdir	Directory to store temporary log files while creating attachments. If this property is not defined, the web service will try to create temporary files in the directory specified by the Java machine property java.io.tmpdir. The web service should have permission to write files in this folder.

Figure 3: Description of the configuration file contents

4.1.1 TCP/IP Services File

To connect a web service client to a Dollar Universe Application Server, the web service uses the information provided by the client application: Company name, Node name and Area. These

elements allow the web service to determine both the hostname and port number necessary to connect to a Dollar Universe instance.

The web service retrieves the port number from the file specified by the property ***com.orsyp.duws.services.file***.

Each entry of this file must be specified as follows:

<COMPANY>_<NODE>_IO_<AREA> <PORT NUMBER>/tcp

For instance:

```
UNIV52_PROMETHEUS_IO_X    10642/tcp
UNIV52_PROMETHEUS_IO_S    10643/tcp
UNIV52_PROMETHEUS_IO_I    10644/tcp
UNIV52_PROMETHEUS_IO_A    10645/tcp
```

In this example, “PROMETHEUS” is the logical node name.

4.1.2 Node Name to Hostname Resolution File

The web service retrieves the physical hostname associated with the logical node name in the file specified by the property **com.orsyp.duws.nodes.file**.

This file corresponds to the *uxsrsrv.sck* in Dollar Universe.

Each line maps a logical node name to a physical hostname as follows:

<NODE NAME> <HOST NAME>

For instance:

```
PROMETHEUS saturn
SUPERMAN venus
```

In this example, the node PROMETHEUS is installed on a physical host called Saturn.

4.2 User Configuration

4.2.1 Web Users

The Web service must be deployed with an authentication method. BASIC Authentication is the default. Invoking Dollar Universe Web Services requires that the invoking web user is assigned the **DollarUniverseUser** role.

4.2.2 Aliasing Web Users to Dollar Universe Users

Web users can be aliased to Dollar Universe users via the Dollar Universe Proxy functions (*setproxy, delproxy, lstproxy,…*). Refer to the Dollar Universe Administration Manual for details about proxy management in Dollar Universe.

If proxies are active on the target Dollar Universe Application Server, the proxy user defined on the target node is returned in the invoked method's responses (except when an exception occurs).

If proxies are active on a target DUAS but the Web user is not aliased to a Dollar Universe user, the invoked operation will be refused by Dollar Universe.

If proxies are active on a Dollar Universe node and the Web user is aliased to a Dollar Universe user but the action is not authorized for that Dollar Universe user, the invoked operation will also be refused by Dollar Universe.

If proxies are not active on the target DUAS, the web user must exist in the Dollar Universe user table.

4.3 Licenses

The file specified by the property ***com.orsyp.duws.lic.file*** contains licenses for all nodes addressed by the web service and its client applications. The product code is **WSE** and its version is **2**.

A license is required for each node accessed by the Dollar Universe Web services. For instance:

```
# node      product_code           release exp_date   license_code
PROMETHEUS  DOLLAR_UNIVERSE:WSE    2       20111231   1234567890DFGZUA85
SUPERMAN    DOLLAR_UNIVERSE:WSE    2       20111231   1234567890DFGZUA75
```

The administrator can manually edit the file to add or remove licenses.

5. Dollar Universe Web Services

5.1 Available Operations

Available operations have been classified as follows:

5.1.1 Operations on Administration Objects

- Get the Management Unit list.
- Get the Node list.

5.1.2 Operations on Development Objects

- Get the Uproc list.
- Get the Session list.
- Get the Task list.

5.1.3 Operations on Monitoring Lists

- Get the Launch list.
- Get the Execution list.
- Get the Execution Event list.

5.1.4 Operations to Extract Monitoring Details

- Get the launch characteristics of a Task.
- Get launch characteristics.
- Get execution characteristics.
- Get an execution's job log.
- Get an execution's job log as a SOAP attachment.
- Get an execution's history trace.

- Get an execution's previous launches.
- Get an event's characteristics.

5.1.5 Operations to Perform Actions

- Create a launch from an existing Task.
- Create a launch from an existing Task (providing values for Uproc variables).
- Create a launch.
- Delete a launch.
- Disable a launch.
- Enable a launch.
- Purge an execution.
- Stop an execution.
- Create an event
- Update an event
- Delete an event.

5.2 Service Description

The **WSDL** file describes how to access the web service provided by Dollar Universe Web Services 2.0 and which operations are provided to the client application.

5.2.1 Data Type Definition in the WSDL File

The **Context** sequence contains 3 elements:

- The Dollar Universe environment,
- The error
- The proxy user.

5.2.1.1 The WSDL Context Object

In the WSDL, **Context** is defined as follows:

```
<complexType name="Context">
   <sequence>
      <element name="envir" type="tns:Envir"/>
      <element name="error" type="tns:UniError"/>
```

```
      <element name="proxyUser" type="string"/>
   </sequence>
</complexType>
```

5.2.1.2 The Dollar Universe Environment (envir Element)

This element of the Context sequence contains details of the Dollar Universe environment where the action will be performed by Dollar Universe via the web service. The environment is set by the client invoking the web service. In the WSDL file, the Dollar Universe environment is defined as:

```
<complexType name="Envir">
   <sequence>
      <element name="area" type="string"/>
      <element name="company" type="string"/>
      <element name="node" type="string"/>
   </sequence>
</complexType>
```

Where:

- **area** must be 'A' for Application, 'I' for Integration, 'S' for Simulation or 'X' for Production,
- **company** is the Dollar Universe company name (6 uppercase characters),
- **node** is the Dollar Universe node name (maximum length is 10 characters, case sensitive).

5.2.1.3 Application Errors (error Element)

This element is now obsolete; however it remains for reasons of backwards compatibility.

5.2.1.4 Proxy User (proxyUser Element)

The proxy user is the Dollar Universe user returned by Dollar Universe with each invoked operation. Dollar Universe returns the Dollar Universe user (proxy user) associated with the Web server connection user.

The proxy user should have been already been defined by the Dollar Universe administrator for users connecting through the Dollar Universe Web Services. To identify users from Dollar Universe Web Services use the WEB system code with the **setproxy** command (see Dollar Universe administration manual).

The following table lists proxy user values returned by the web service depending on whether proxies are active on a targeted Dollar Universe node and how proxies are defined for a Web service user.

Proxy activation / Alias definition in Dollar Universe node	Proxy user / Error code returned in the response message
Proxies are not active in Dollar Universe	context.proxyUser = "?"
Proxies are active in Dollar Universe AND the Web service user is	context.proxyUser = <Dollar Universe user>

aliased to a Dollar Universe User.	
Proxies are active in Dollar Universe AND the Web service user is NOT aliased to a Dollar Universe User.	A DuwsException is raised. e.g.: `com.orsyp.duws.stubs.DuwsException: No proxy defined for user - admin, in environment [UNIV53/vmsdmduws1/X]`

Figure 4: Proxy responses to the web service

5.2.2 Filters

Filters are available for operations that retrieve Dollar Universe object lists. The available filters are defined in the WSDL file.

Filter formats are described in the table below:

Member	Format
Status	String. This can be a combination of the characters I, R, O, W, A, E, D, F, L, T and S. See the Appendix for a list of status codes and their meaning.
Node	10 char string. Permitted values: <string_filter>[*] or *.
Mu	10 char string. Permitted values: <string_filter>[*] or *.
Uproc	10 char string. Permitted values: <string_filter>[*] or *.
Session	10 char string. Permitted values: <string_filter>[*] or *.
User	10 char string. Permitted values: <string_filter>[*] or *.
creationDateMin	8 char string. Format is YYYYMMDD.
creationDateMax	8 char string. Format is YYYYMMDD.
creationHourMin	6 char string. Format is HHMMSS.
creationHourMax	6 char string. Format is HHMMSS.
updateDateMin	8 char string. Format is YYYYMMDD.
updateDateMax	8 char string. Format is YYYYMMDD.
updateHourMin	6 char string. Format is HHMMSS.
updateHourMax	6 char string. Format is HHMMSS.
beginDateMin	8 char string. Format is YYYYMMDD.
beginDateMax	8 char string. Format is YYYYMMDD.
beginHourMin	6 char string. Format is HHMMSS.
beginHourMax	6 char string. Format is HHMMSS.
endDateMin	8 char string. Format is YYYYMMDD.
endDateMax	8 char string. Format is YYYYMMDD.
endHourMin	6 char string. Format is HHMMSS.
endHourMax	6 char string. Format is HHMMSS.
processingDate	8 char string. Format is YYYYMMDD. A value of "*" means "All processing dates". A value of "00000000" means "No processing date".

Member	Format
numlancMin	7 numerical char string with leading zeros
numlancMax	7 numerical char string with leading zeros
numsessMin	7 numerical char string with leading zeros
numsessMax	7 numerical char string with leading zeros
numprocMin	7 numerical char string with leading zeros
numprocMax	7 numerical char string with leading zeros

Figure 5: WSDL filter list

The available filters depend on the type of list requested:

5.2.2.1 NodeFilter

```
<complexType name="NodeFilter">
   <sequence>
      <element name="node" type="string"/>
   </sequence>
</complexType>
```

5.2.2.2 MUFilter

```
<complexType name="MUFilter">
   <sequence>
      <element name="mu" type="string"/>
   </sequence>
</complexType>
```

5.2.2.3 UprocFilter

```
<complexType name="UprocFilter">
   <sequence>
      <element name="uproc" type="string"/>
   </sequence>
</complexType>
```

5.2.2.4 SessionFilter

```
<complexType name="SessionFilter">
   <sequence>
      <element name="session" type="string "/>
   </sequence>
</complexType>
```

5.2.2.5 TaskFilter

```
<complexType name="TaskFilter">
   <sequence>
      <element name="mu" type="string"/>
      <element name="session" type="string"/>
      <element name="uproc" type="string"/>
   </sequence>
</complexType>
```

5.2.2.6 LaunchFilter

```
<complexType name="LaunchFilter">
   <sequence>
      <element name="beginDateMax" type="string"/>
      <element name="beginDateMin" type="string"/>
      <element name="beginHourMax" type="string"/>
      <element name="beginHourMin" type="string"/>
      <element name="endDateMax" type="string"/>
      <element name="endDateMin" type="string"/>
      <element name="endHourMax" type="string"/>
      <element name="endHourMin" type="string"/>
      <element name="mu" type="string"/>
      <element name="numlancMax" type="string"/>
      <element name="numlancMin" type="string"/>
      <element name="numprocMax" type="string"/>
      <element name="numprocMin" type="string"/>
      <element name="numsessMax" type="string"/>
      <element name="numsessMin" type="string"/>
      <element name="processingDate" type="string"/>
      <element name="session" type="string"/>
      <element name="status" type="string"/>
      <element name="uproc" type="string"/>
      <element name="user" type="string"/>
   </sequence>
</complexType>
```

5.2.2.7 ExecutionFilter

```
<complexType name="ExecutionFilter">
   <sequence>
      <element name="beginDateMax" type="string"/>
      <element name="beginDateMin" type="string"/>
      <element name="beginHourMax" type="string"/>
      <element name="beginHourMin" type="string"/>
      <element name="endDateMax" type="string"/>
      <element name="endDateMin" type="string"/>
      <element name="endHourMax" type="string"/>
      <element name="endHourMin" type="string"/>
      <element name="mu" type="string"/>
      <element name="numlancMax" type="string"/>
```

```
      <element name="numlancMin" type="string"/>
      <element name="numprocMax" type="string"/>
      <element name="numprocMin" type="string"/>
      <element name="numsessMax" type="string"/>
      <element name="numsessMin" type="string"/>
      <element name="processingDate" type="string"/>
      <element name="relaunched" type="boolean"/>
      <element name="session" type="string"/>
      <element name="status" type="string"/>
      <element name="uproc" type="string"/>
      <element name="user" type="string"/>
   </sequence>
</complexType>
```

5.2.2.8 EventFilter

```
<complexType name="EventFilter">
   <sequence>
      <element name="creationDateMax" type="string"/>
      <element name="creationDateMin" type="string"/>
      <element name="creationHourMax" type="string"/>
      <element name="creationHourMin" type="string"/>
      <element name="mu" type="string"/>
      <element name="numlancMax" type="string"/>
      <element name="numlancMin" type="string"/>
      <element name="numprocMax" type="string"/>
      <element name="numprocMin" type="string"/>
      <element name="numsessMax" type="string"/>
      <element name="numsessMin" type="string"/>
      <element name="processingDate" type="string"/>
      <element name="session" type="string"/>
      <element name="status" type="string"/>
      <element name="updateDateMax" type="string"/>
      <element name="updateDateMin" type="string"/>
      <element name="updateHourMax" type="string"/>
      <element name="updateHourMin" type="string"/>
      <element name="uproc" type="string"/>
      <element name="user" type="string"/>
   </sequence>
</complexType>
```

5.2.3 Identifiers

Identifiers (combined with the Context sequence) are used to select Dollar Universe objects on which an action is to be performed, for example task submission or object deletion. Identifiers are returned in the list for each object. The format of the identifier parts is described in the table below:

Member	Format
node	10 character string.
mu	10 char string.

Member	Format
uproc	10 char string.
uprocVersion	3 char string. Permitted values: "000" … "999".
session	10 char string.
sessionVersion	3 char numerical string. Permitted values: "000" … "999"
numLanc	7 char string. Permitted values: "0000000" … "9999999"
numProc	7 char string. Permitted values: "0000000" … "9999999"
numSess	7 char string. Permitted values: "0000000" … "9999999"
processingDate	8 char string. The format is "YYYYMMDD", where YYYY represents the year, MM the month [01;12] and DD the day [01;31]. If no processing date is provided, the string "00000000" is used.
user	12 char string.

Figure 6: WSDL identifier list

In the WSDL file, the identifiers are defined as follows:

5.2.3.1 Node

```
<complexType name="NodeId">
   <sequence>
      <element name="node" type="string"/>
   </sequence>
</complexType>
```

5.2.3.2 Management Unit

```
<complexType name="MUId">
   <sequence>
      <element name="mu" type="string"/>
   </sequence>
</complexType>
```

5.2.3.3 Uproc

```
<complexType name="UprocId">
   <sequence>
      <element name="uproc" type="string"/>
      <element name="uprocVersion" type="string"/>
   </sequence>
</complexType>
```

5.2.3.4 Session

```
<complexType name="SessionId">
   <sequence>
      <element name="session" type="string"/>
```

```
            <element name="sessionVersion" type="string"/>
         </sequence>
      </complexType>
```

5.2.3.5 Task

```
      <complexType name="TaskId">
         <sequence>
            <element name="mu" type="string"/>
            <element name="session" type="string"/>
            <element name="sessionVersion" type="string"/>
            <element name="template" type="boolean"/>
            <element name="uproc" type="string"/>
            <element name="uprocVersion" type="string"/>
         </sequence>
      </complexType>
```

5.2.3.6 Launch

```
      <complexType name="LaunchId">
         <sequence>
           <element name="mu" type="string"/>
           <element name="numLanc" type="string"/>
           <element name="numProc" type="string"/>
           <element name="numSess" type="string"/>
           <element name="session" type="string"/>
           <element name="sessionVersion" type="string"/>
           <element name="uproc" type="string"/>
           <element name="uprocVersion" type="string"/>
         </sequence>
      </complexType>
```

5.2.3.7 Execution

```
<complexType name="ExecutionId">
         <sequence>
            <element name="mu" type="string"/>
            <element name="numLanc" type="string"/>
            <element name="numProc" type="string"/>
            <element name="numSess" type="string"/>
            <element name="session" type="string"/>
            <element name="sessionVersion" type="string"/>
            <element name="uproc" type="string"/>
            <element name="uprocVersion" type="string"/>
         </sequence>
      </complexType>
```

5.2.3.8 Event

```
      <complexType name="EventId">
```

```
   <sequence>
      <element name="mu" type="string"/>
      <element name="numProc" type="string"/>
      <element name="numSess" type="string"/>
      <element name="processingDate" type="string"/>
      <element name="session" type="string"/>
      <element name="uproc" type="string"/>
      <element name="user" type="string"/>
   </sequence>
</complexType>
```

5.2.4 Request Messages

Request messages contain the parameters of a web service operation. All methods receive a **context** sequence. The **envir** elements **company**, **node**, **area** of the context sequence must be supplied.

Depending on the operation invoked, the request may also contain either a filter or an identifier.

5.2.5 Response Messages

Response messages contain:

- A **context** sequence,
- The results of the action executed by the web service.

The **envir** element of the **context** sequence in the response is simply a copy of the context sequence of the associated request.

The proxy user can be retrieved from the response by accessing the **proxyUse**r element.

5.2.6 Error Message

Errors are returned to the operation invoker as `DuwsException` messages when ***communications-related errors*** occur or ***invalid parameters*** are sent to the web service in the request message. In the WSDL, the message is defined as follows:

```
<complexType name="DuwsException">
   <sequence>
      <element name="message" type="string"/>
   </sequence>
</complexType>
```

where the **message** element is a text giving details of the error. This message can be displayed to an operator.

5.2.7 Request/Response/Fault Messages by Operation

In the **WSDL** file, each operation is defined by an expected request message and a corresponding response message:

Operation	Request message	Response message	Fault message
Get Management Unit list	DuwsSEI_getListMU	DuwsSEI_getListMUResponse	DuwsException
Get node list	DuwsSEI_getListNode	DuwsSEI_getListNodeResponse	DuwsException
Get Uproc list	DuwsSEI_getListUproc	DuwsSEI_getListUprocResponse	DuwsException
Get Session list	DuwsSEI_getListSession	DuwsSEI_getListSessionResponse	DuwsException
Get Task list	DuwsSEI_getListTask	DuwsSEI_getListTaskResponse	DuwsException
Get launch list	DuwsSEI_getListLaunch	DuwsSEI_getListLaunchResponse	DuwsException
Get execution list	DuwsSEI_getListExecution	DuwsSEI_getListExecutionResponse	DuwsException
Get event list	DuwsSEI_getListEvent	DuwsSEI_getListEventResponse	DuwsException
Get an execution's previous launches	DuwsSEI_getPreviousLaunches	DuwsSEI_getPreviousLaunchesResponse	DuwsException
Get an execution's characteristics	DuwsSEI_getExecution	DuwsSEI_getExecutionResponse	DuwsException
Get launch characteristics of a Task	DuwsSEI_getLaunchFromTask	DuwsSEI_getLaunchFromTaskResponse	DuwsException
Get launch characteristics	DuwsSEI_getLaunch	DuwsSEI_getLaunchResponse	DuwsException
Get execution's job log	DuwsSEI_getExecutionLog	DuwsSEI_getExecutionLogResponse	DuwsException
Get execution's job log as a SOAP Attachment	DuwsSEI_getExecutionLogAsAttachment	DuwsSEI_getExecutionLogAsAttachmentResponse	N/A
Get execution's history trace	DuwsSEI_getHistoryTrace	DuwsSEI_getHistoryTraceResponse	DuwsException
Get an event	DuwsSEI_getEvent	DuwsSEI_getEventResponse	DuwsException
Create a launch from an existing task	DuwsSEI_addLaunchFromTask	DuwsSEI_addLaunchFromTaskResponse	DuwsException
Create a launch from an existing task (providing values for Uproc variables)	DuwsSEI_addLaunchFromTask2	DuwsSEI_addLaunchFromTask2Response	DuwsException
Delete a launch	DuwsSEI_deleteLaunch	DuwsSEI_deleteLaunchResponse	DuwsException
Disable a launch	DuwsSEI_disableLaunch	DuwsSEI_disableLaunchResponse	DuwsException

Operation	Request message	Response message	Fault message
Enable a launch	DuwsSEI_enableLaunch	DuwsSEI_enableLaunchResponse	DuwsException
Purge an execution	DuwsSEI_purgeExecution	DuwsSEI_purgeExecutionResponse	DuwsException
Stop an execution	DuwsSEI_stopExecution	DuwsSEI_stopExecutionResponse	DuwsException
Delete an event	DuwsSEI_deleteEvent	DuwsSEI_deleteEventResponse	DuwsException
Create a launch	DuwsSEI_addLaunch	DuwsSEI_addLaunchResponse	DuwsException
Create an event	DuwsSEI_addEvent	DuwsSEI_addEventResponse	DuwsException
Update an event	DuwsSEI_updateEvent	DuwsSEI_updateEventResponse	DuwsException

Figure 7: WSDL request, response and fault message list.

5.3 Operations

The following web service operations are described using:

- **Description**: this text gives a detailed explanation of the function performed by the invoked service.
- **Operation name**: this is the name of the invoked method.
- **Parameters**: the invoked service may receive parameters. Each parameter is specified as an input ([IN]) parameter, an output parameter ([OUT]) or both ([IN/OUT]).

Examples:

*The **context** parameter is always expected by all web service operations. This parameter contains pointers to a Company, a node and an Area. The context also contains the error (a code and a text) if the invoked service could not perform the operation. If the operation is successful, the error code is **1**. In a response message, the **proxyUser** is returned to identify the Web service client as a Dollar Universe User. This parameter is [IN/OUT].*

*The **filter** parameter is used to extract object lists from Dollar Universe. Depending on the type of list, you must use the appropriate filter. This parameter is [IN].*
*The object identifier, **id**, is used to perform an action on a selected object. Object identifiers are provided for each object returned in the lists by the operations. This parameter is [IN].*

- **Input message**: this is the name of the message structure sent to the invoked service. The message carries the parameters passed to the invoked service.
- **Output message**: this is the name of the message structure returned by the invoked service. The message contains the results of the requested operation.
- **Exception**: this is the message returned by the Web service or one of the involved elements between the service customer and the service itself if a fault is detected.

Note: In certain cases, some information is superfluous, for example Session name is sometimes not mandatory. In such a case, consider entering an empty string (“”) rather than a null value, otherwise you might get a NullPointer exception.

5.3.1 Operations on Administration Objects

5.3.1.1 Get the Management Unit List

Description	This method returns the list of Management Units defined in the Management Unit table of the node specified in the context for the selected company. A filter can be applied. The filter is a string formatted as <pattern>[*]. If the operation executes with no error, the returned list contains the Management Units. For each element of the list, the following attributes are available: • Management Unit name, • Label, • Node location, • Development authorization flag, • Production authorization flag. If an error occurs the error code and error text are returned in the context sequence of the response message and the list is empty.
Operation name	getListMU
Parameters	Context: Context [IN/OUT] filter: MUFilter [IN]
Input message	**DuwsSEI_getListMU**: the message contains the parameter context and filter on management units.
Output message	**DuwsSEI_getListMUResponse**: the message contains the list of management units (according to the filter). If an application error occurs, the message also contains the error code and error text in the context element. If the action is successful the Management Unit list is returned by the web service. For each Management Unit, the following XML sequence is sent: `<complexType name="MUItem"><sequence><element name="development" type="boolean"/><element name="ident" type="tns:MUId"/><element name="label" type="string"/><element name="node" type="string"/><element name="production" type="boolean"/></sequence></complexType>` where: **development**: indicates that the MU is authorized in the development universe. **ident**: the management unit identifier. The **mu** element of **ident** is a string of 10 uppercase characters max. **label**: the label text is recorded in Dollar Universe. It is a string of 20 characters max. **node**: the location of the Management Unit. **production**: indicates that the MU is authorized in the production universe.

Figure 8: Operation: getListMU

5.3.1.2 Get the Node List

Description	This operation returns the list of nodes defined in the node table of the Dollar Universe node specified in the context. A filter can be applied. The filter is a string formatted as `<pattern>[*]`. If the operation executes with no error, the list returned contains the nodes. For each element of the list, the following attributes are available: • Node name, • Label, • Central monitoring flag, • Development authorization flag, • Production authorization flag. If an error occurs, the error code and error text are returned in the context sequence of the response message and the list is empty.
Operation name	getListNode
Parameters	context: Context [IN/OUT] filter: NodeFilter [IN]
Input message	**DuwsSEI_getListNode**: the message contains the parameter context and filter on nodes.
Output message	**DuwsSEI_getListNodeResponse**: the message contains the list of nodes (depending on filter). If an application error occurs, the message also contains the error code and error text in the context element. If the action is successful the node list is returned by the web service. For each node, the following XML sequence is sent: `<complexType name="NodeItem"><sequence><element name="centralControl" type="boolean"/><element name="development" type="boolean"/><element name="ident" type="tns:NodeId"/><element name="label" type="string"/><element name="production" type="boolean"/></sequence></complexType>` where: **centralControl**: indicates that the node is a central monitoring node. **development**: indicates that the node is authorized in the development universe. **ident**: the node id. The **node** element of **ident** is a string of 10 characters length and is case sensitive. **label**: the label text is recorded in Dollar Universe. It is a string of 20 characters max. **production**: indicates that the node is authorized in the production universe.

Figure 9: Operation: getListNode

5.3.2 Operations on Development Objects

5.3.2.1 Get the Uproc List

Description	This operation returns the list of Uprocs defined in the Dollar Universe node of the selected Company and Area. A filter can be applied. The filter is a string formatted as `<pattern>[*]`. If the operation performs with no error, the returned list contains the Uprocs. For each element of the list, the following attributes are available: • Uproc name, • Uproc version, • Label, • Application it belongs to, • Domain it belongs to. If an error occurs, the error code and error text are returned in the context sequence of the response message and the list is empty.
Operation name	getListUproc
Parameters	context: Context [IN/OUT] filter: UprocFilter [IN]
Input message	**DuwsSEI_getListUproc**: the message contains the parameter context and filter on uprocs.

Output message	**DuwsSEI_getListUprocResponse:** the message contains the list of Uprocs (depending on the filter). If an error occurs, the message contains also the error code and error text in the context sequence.
	If the action is successful the Uproc list is returned by the web service. For each Uproc, the following XML sequence is sent:
	`<complexType name="UprocItem"><sequence><element name="application" type="string"/><element name="domain" type="string"/><element name="ident" type="tns:UprocId"/><element name="label" type="string"/></sequence></complexType>`
	where:
	application: the application the Uproc belongs to. It is a string of 2 characters.
	domain: the domain the Uproc belongs to. It is a string of 1 character ('A' … 'Z').
	ident: identifier of the Uproc described with the sequence `UprocId`:
	`<complexType name="UprocId"><sequence><element name="uproc" type="string"/><element name="uprocVersion" type="string"/></sequence>`

	</complexType> where: **uproc**: member of **ident** is the uproc name. It is a string of 10 uppercase characters maximum. **uprocVersion**: member of **ident** is the uproc version. It is a string of 3 numerical characters, with leading zero **label**: the label text is recorded into Dollar Universe. It is a string of 60 characters max.

Figure 10: Operation: getListUproc

5.3.2.2 Get the Session List

Description	This operation returns the list of the Sessions defined in the Dollar Universe node of the selected Company and Area. A filter can be applied. The filter is a string formatted as `<pattern>[*]`. If the execution is performed with no error, the returned list contains the Sessions. For each element of the list, the following attributes are available: • Session name, • Session version, • Label, • Session header If an error occurs, the error code and error text are returned in the context sequence of the response message and the list is empty.
Operation name	getListSession
Parameters	context: Context [IN/OUT] filter: SessionFilter [IN]
Input message	**DuwsSEI_getListSession**: the message contains the parameter context and filter on sessions.

Output message	**DuwsSEI_getListSessionResponse**: the message contains the list of Sessions (depending on the filter). If an error occurs, the message also contains the error code and error text in the context element of the response message.
	If the action is successful the Session list is returned by the web service. For each Session, the following XML sequence is sent:
	`<complexType name="SessionItem"><sequence><element name="head" type="string"/><element name="ident" type="tns:SessionId"/><element name="label" type="string"/></sequence></complexType>`
	where:
	head: the uproc header of the session. It is a string of 10 uppercase characters max.
	ident: the identifier of the session specified with the sequence `SessionId`:
	`<complexType name="SessionId"><sequence><element name="session" type="string"/><element name="sessionVersion" type="string"/></sequence></complexType>`
	where:
	session: is the name of the session. It is a string of 10 characters uppercase max.
	sessionVersion: is the version of the session. It is a string of 3 numerical characters, with leading zeroes.
	label: the label text is recorded in Dollar Universe. It is a 40 character string.

Figure 11: Operation: getListSession

5.3.2.3 Get the Task List

Description	This operation returns the list of the tasks defined on the Dollar Universe node of the selected Company and Area. A filter can be applied. The filter for a task contains 3 sub-filters: the filters on Uprocs, Sessions and Management Units. The filters are strings formatted as `<pattern>[*]`. If the execution performs with no errors, the returned list contains the Tasks. For each element of the list, the following attributes are available: **Task identifier**: the uproc name and version, the session name and version (if any), the management unit (if it is not a template task) or the template name (if the task is a template), the template task flag marking the task as template or not. • Author code, • Batch queue, **Task type**: 'D' for scheduled, 'P' for provoked, 'S' for optional. If an error occurs, the error code and error text are returned in the context sequence of the response message and the list is empty.
Operation name	getListTask
Parameters	context: Context [IN/OUT] filter: TaskFilter [IN]
Input message	**DuwsSEI_getListTask**: the message contains the parameter context and filter on tasks.

Output message	**DuwsSEI_getListTaskResponse:** the message contains the list of Tasks (depending on the filter). If an error occurs, the message also contains the error code and error text in the context element.
	If the action is performed successfully the Task list is returned by the web service. For each Task, the following XML sequence `TaskItem` is sent:
	`<complexType name="TaskItem"><sequence><element name="authorCode" type="string"/><element name="ident" type="tns:TaskId"/><element name="queue" type="string"/><element name="type" type="string"/></sequence></complexType>`
	where:
	authorCode: the code of the Dollar Universe user who executes the task. It is a string of 3 numerical characters.
	ident: the identifier of the task specified with the sequence `TaskId`:
	`<complexType name="TaskId"><sequence><element name="mu" type="string"/><element name="session" type="string"/><element name="sessionVersion" type="string"/><element name="template" type="boolean"/><element name="uproc" type="string"/><element name="uprocVersion" type="string"/>`

	</sequence> </complexType> where: **mu**: is the name of the Management Unit or the template name. It is a string of 10 uppercase characters max. **session**: is the name of the Session. It is a string of 10 uppercase characters max. This element can be empty if the Task does not launch a Session. **sessionVersion**: is the version of the Session. It is a string of 3 numerical characters, with leading zeroes. This element can be empty if The task does not launch a Session. **template**: indicates if the task is a template or not. **uproc**: is the name of the Uproc header. It is a string of 10 uppercase characters max. **uprocVersion**: is the Uproc version. It is a string of 3 numerical characters, with leading zero **queue**: is the batch queue where the future launch created from this Task is submitted. It is a string of 31 characters max. **type**: is the type of the Task. It is a string of one character. Returned values can be 'D', 'P' or 'S'. See the Appendix for a list of task type codes and their meaning.

Figure 12: Operation: getListTask

5.3.3 Operations on Monitoring Lists

5.3.3.1 Get the Launch List

Description	This operation returns the list of the launches defined on the Dollar Universe node of the selected Company and Area. A filter can be applied. The filter for launches contains the following elements: the filter on Uprocs, the filter on Sessions, the filter on MUs, the filter on launch status, the filter on start date and time (min and max), the filter on end date and time (min and max), the filter on processing date, the filter on launch number (min and max), the filter on Uproc number (min and max), the filter on Session number (min and max). The filters on Uproc, Session, MU and user are strings formatted as <pattern>[*]. The filter on the launch status accepts a string where each character corresponds to a status code. If the filter selects every status, the status filter string will be "SLWO". Status codes are listed in the Appendix with their meanings. The filters on launch number, Session number and Uproc number are strings of 7 numerical characters with leading zeros. The filters on start and end dates are a string of 8 characters with the format YYYYMMDD. The filters on start and end times are a string of 6 characters with the format HHMMSS. The filter on processing date is a string of 8 characters with the format YYYYMMDD. A value of "00000000" means "no processing date", a value of "*" means "All processing dates". If the operation executes with no errors, the returned list will contain the launches. For each element of the list, the following attributes are available: • Launch identifier: the Uproc name and version, the Session name and version (if any), the MU, the launch number, uproc number and session number (if any). • Launch attributes: status, start date and time, end date and time, execution node, priority, processing date, batch queue, user, step. If an error occurs, the error code and error text are returned in the context sequence of the response message and the list is empty.
Operation name	getListLaunch
Parameters	context: Context [IN/OUT] filter: LaunchFilter [IN]
Input message	**DuwsSEI_getListLaunch**: the message contains the parameter context and filter on launches.

Output message	**DuwsSEI_getListLaunchResponse**: the message contains the list of launches (depending on the filter). If an error occurs, the message also contains the error code and error text in the context element.
	`If the action is successful the launch list is returned by the web service. For each launch, the following XML sequence LaunchItem is sent:`
	`<complexType name="LaunchItem">`
	`<sequence>`

```
                <element name="beginDate" type="string"/>
                <element name="beginHour" type="string"/>
                <element name="endDate" type="string"/>
                <element name="endHour" type="string"/>
                <element name="ident"
        type="tns:LaunchId"/>
                <element name="node" type="string"/>
                <element name="priority" type="string"/>
                <element name="processingDate"
        type="string"/>
                <element name="queue" type="string"/>
                <element name="status" type="string"/>
                <element name="step" type="string"/>
                <element name="user" type="string"/>
            </sequence>
         </complexType>
```

where:

beginDate: the begin date of the launch window. This is a string of 8 characters with the format YYYYMMDD.

beginHour: the begin time of the launch window. This is a string of 6 characters with the format HHMMSS.

endDate: the end date of the launch window. This is a string of 8 characters with the format YYYYMMDD.

endHour: the end time of the launch window. This is a string of 6 characters with the format HHMMSS.

ident: this identifier of the launch is given by the sequence LaunchId:

```
 <complexType name="LaunchId">
    <sequence>
       <element name="mu" type="string"/>
       <element name="numLanc" type="string"/>
       <element name="numProc" type="string"/>
       <element name="numSess" type="string"/>
       <element name="session" type="string"/>
       <element name="sessionVersion"
type="string"/>
       <element name="uproc" type="string"/>
       <element name="uprocVersion"
type="string"/>
    </sequence>
 </complexType>
```

where:

mu: is the name of the management unit. It is a string of 10 uppercase characters max.

numLanc: the launch number. A string of 7 numerical characters with leading zeros.

numProc: the uproc number. A string of 7 numerical characters with leading zeros.

numSess: the session number. A string of 7 numerical characters with leading zeros. This element can be empty if the launch concerns a standalone Uproc.

session: the session name. A string of 10 characters uppercase max. This element can be empty if the launch concerns a standalone Uproc.

sessionVersion: the session version number. A string of 3 numerical characters with leading zeros. This element can be empty if the launch concerns a standalone

	Uproc. **uproc**: the Uproc name. A string of 10 characters uppercase max. **uprocVersion**: the Uproc version number. A string of 3 numerical characters, with leading zeros. **node**: the node where the job will run. A string of 10 characters max. The node name is case sensitive. **priority**: the job's execution priority. It is a string of 3 characters (with leading zeros) representing a numerical value from 1 to 255. **processingDate**: the processing date assigned to the job. A string of 8 characters - format YYYYMMDD. If the launch has no processing date, this element is 00000000. **queue**: the batch execution queue. A string of 31 characters max. **status**: the status of the launch item. Status codes are listed in the Appendix with their meanings. **step**: the step number (in the uproc script) from which the job must start running. A string of 2 numerical characters from 00 to 99. **user**: the Dollar Universe user who submits the launch. A string of 12 characters max.

Figure 13: Operation: getListLaunch

5.3.3.2 Get the Execution List

Description	This operation returns the list of executions defined on the Dollar Universe node of the selected Company and Area. A filter can be applied. The filter for executions contains the following elements: the filter on Uprocs, the filter on Sessions, the filter on MUs, the filter on execution status, the filter on the submission user, the filter on start date and time (min and max), the filter on end date and time (min and max), the filter on the processing date, the filter on launch number (min and max), the filter on Uproc number (min and max), the filter on Session number (min and max) and the filter on relaunched executions. The filters on Uproc, Session, MU and user are strings formatted as `<pattern>[*]`. The filter on the execution status accepts a string where each character corresponds to a status code. If the filter selects every status, the status filter string will be "ADEFIORTW". The codes are described in the Appendix If a code is mentioned in the status filter, the service will return executions with the corresponding status. The filters on launch number, Session number and Uproc number are strings of 7 numerical characters, with leading zeros. The filters on start and end dates are a string of 8 characters - format YYYYMMDD. The filters on start and end times are a string of 6 characters - format HHMMSS. The filter on processing date is a string of 8 characters - format YYYYMMDD. A value of "00000000" means "no processing date", a value of "*" means "All processing dates". If the operation executes with no errors, the returned list contains the executions. For each element of the list, the following attributes are available: • Execution identifier: Uproc name and version, Session name and version (if any), MU, launch number, Uproc number and Session number (if any). • Execution attributes: status, start date and time, end date and time, execution node, priority, processing date, batch queue, user, re-launch flag and batch queue entry number. If an error occurs, the error code and error text are returned in the context sequence of the response message and the list is empty.
Operation name	getListExecution
Parameters	context: Context [IN/OUT] filter: ExecutionFilter [IN]
Input message	**DuwsSEI_getListExecution**: the message contains the parameter context and filter on executions.

Output message	**DuwsSEI_getListExecutionResponse**: the message contains the list of executions (depending on the filter). If an error occurs, the message contains also the error code and error text in the context element of the message.
	If the action is successful the execution list is returned by the web service. For each execution, the following XML sequence `ExecutionItem` is sent:
	`<complexType name="ExecutionItem">`
	`<sequence>`
	`<element name="beginDate" type="string"/>`
	`<element name="beginHour" type="string"/>`
	`<element name="endDate" type="string"/>`

```
                <element name="endHour" type="string"/>
                <element name="ident"
        type="tns:ExecutionId"/>
                <element name="node" type="string"/>
                <element name="numEntry" type="string"/>
                <element name="priority" type="string"/>
                <element name="processingDate"
        type="string"/>
                <element name="queue" type="string"/>
                <element name="relaunched"
        type="boolean"/>
                <element name="status" type="string"/>
                <element name="user" type="string"/>
            </sequence>
        </complexType>
```

where:

beginDate: the start date of the execution. This is a string of 8 characters with the format YYYYMMDD.

beginHour: the start time of the execution. The format of this string is HHMMSS.

endDate: the end date of the execution. This is a string of 8 characters with the format YYYYMMDD.

endHour: the end time of the execution. The format of this string is HHMMSS.

ident: this identifier of the execution is given by the sequence `ExecutionId`:

```
 <complexType name="ExecutionId">
    <sequence>
       <element name="mu" type="string"/>
       <element name="numLanc" type="string"/>
       <element name="numProc" type="string"/>
       <element name="numSess" type="string"/>
       <element name="session" type="string"/>
       <element name="sessionVersion"
type="string"/>
       <element name="uproc" type="string"/>
       <element name="uprocVersion"
type="string"/>
    </sequence>
 </complexType>
```

where:

mu: the name of the management unit. A string of 10 uppercase characters max.

numLanc: the launch number. A string of 7 numerical characters with leading zeros.

numProc: the Uproc number. A string of 7 numerical characters with leading zeros.

numSess: the session number. A string of 7 numerical characters, with leading zeros. This element can be empty if the execution concerns a standalone Uproc.

session: is the name of the session. It is a string of 10 uppercase characters max. This element can be empty if the execution concerns a standalone Uproc.

sessionVersion: the version of the session. A string of 3 numerical characters, with leading zeros. This element can be empty if the execution concerns a standalone Uproc.

uproc: the name of the executed Uproc. A string of 10 uppercase characters max.

uprocVersion: the Uproc version. A string of 3 numerical characters, with leading zeros.

	node: the node where the execution ran. A string of 10 characters max. The name is case sensitive. **numEntry**: the batch queue entry number of the execution. A string of 4 numerical characters with leading zeros. **priority**: the priority within the queue. A string of 3 numerical characters from 001 to 255. **processingDate**: the processing date associated with the execution. The format is YYYYMMDD. **queue**: the batch queue where the execution ran. The batch queue name is limited to 31 characters. **relaunched**: this flag tells if the execution was re-launched or not (this flag is meaningless for the getExecution operation). **status**: the status of the execution is a string of 1 character whose value can be I, R, O, W, A, E, D,T or F. Status codes are detailed in the Appendix. .**user**: the submission account associated with the execution. A string of 12 characters max.

Figure 14: Operation: getListExecution

5.3.3.3 Get the Events List

Description	This operation returns the list of the events created on the Dollar Universe node of the selected Company and Area. A filter can be applied. The filter for events contains the following elements: the filter on Uprocs, the filter on Sessions, the filter on MUs, the filter on status, the filter on user, the filter on creation date and time (min and max), the filter on update date and time (min and max), the filter on processing date, the filter on launch number (min and max), the filter on Uproc number (min and max), the filter on Session number (min and max). The filters on Uproc, Session, MU, and user are strings formatted as <pattern>[*]. The filter on the event status accepts a string where each character corresponds to a status code. If the filter selects all statuses, the status filter string will be " TIEAD". The meaning of the codes is detailed in the Appendix The filters on launch number, Session number and Uproc number are strings of 7 numerical characters with leading zeros. The filters on creation and update dates are strings of 8 characters - format YYYYMMDD. The filters on creation and update times are string of 6 characters - format HHMMSS. The filter on processing date is a string of 8 characters with the format YYYYMMDD. A value of "00000000" means "no processing date", a value of "*" means "All processing dates". If the operation executes with no errors, the returned list will contain the events matching the filter. For each element of the list, the following attributes are available: • Event identifier: Uproc name, Session name (if any), MU, launch number, Uproc number, Session number (if any), processing date. • Event attributes: status, creation date and time, last update date and time , author code, step, user. If an error occurs, the error code and error text are returned in the context sequence of the response message and the list is empty.
Operation name	getListEvent
Parameters	context: Context [IN/OUT] filter: EventFilter [IN]
Input message	**DuwsSEI_getListEvent**: the message contains the parameter context and filter on events.

Output message	**DuwsSEI_getListEventResponse**: the message contains the list of events (depending on the filter). If an error occurrs, the message also contains the error code and error text in the context sequence of the response.
	`If the action is successful the event list is returned by the web service. For each event, the following XML sequence EventItem is sent:`
	`<complexType name="EventItem">`
	`<sequence>`
	`<element name="authorCode" type="string"/>`
	`<element name="creationDate" type="string"/>`
	`<element name="creationHour"`

```
        type="string"/>
               <element name="ident" type="tns:EventId"/>
               <element name="numLanc" type="string"/>
               <element name="status" type="string"/>
               <element name="step" type="string"/>
               <element name="updateDate" type="string"/>
               <element name="updateHour" type="string"/>
            </sequence>
         </complexType>
```

where:

authorCode: this is the submission account author code. It is a string of 3 numerical characters.

creationDate: the creation date of the event. This is a string of 8 numerical characters with the format YYYYMMDD.

creationHour: the creation time of the event. The format of this string is HHMMSS.

ident: this is the event identifier described with the sequence EventId:

```
         <complexType name="EventId">
            <sequence>
               <element name="mu" type="string"/>
               <element name="numProc" type="string"/>
               <element name="numSess" type="string"/>
               <element name="processingDate"
       type="string"/>
               <element name="session" type="string"/>
               <element name="uproc" type="string"/>
               <element name="user" type="string"/>
            </sequence>
         </complexType>
```

where:

mu: the name of the management unit. A string of 10 uppercase characters max.

numProc: the uproc number. A string of 7 numerical characters, with leading zeros.

numSess: the session number. A string of 7 numerical characters, with leading zeros. This element can be empty if the execution concerns a standalone Uproc.

processingDate: the processing date associated with this execution of the job. The format is YYYYMMDD.

session: the name of the session. A string of 10 uppercase characters max. This element can be empty if the execution concerns a standalone Uproc.

uproc: the name of the executed Uproc. A string of 10 uppercase characters max.

user: the submission account associated with the event. A string of 12 characters max.

numLanc: the launch number. A string of 7 numerical characters, with leading zeros.

status: the status of the event is a string of 1 character which value can be T, I, E, A, D. Status codes are detailed in the Appendix.

step: the last step number reached by the execution.

updateDate: the date of the last update of the event. A string of 8 numerical characters - format YYYYMMDD.

updateHour: the time the event was last updated. A string of 4 numerical characters - format HHMM.

user: the Dollar Universe user who updated the event. The user can be empty if it was created by the engine. A string of 12 characters max.

Figure 15: Operation: getListEvent

5.3.4 Operations to Extract Monitoring Details

5.3.4.1 Get Execution Characteristics

Description	This operation returns details of an execution on the Dollar Universe node of the selected Company and Area. The information returned in the response are: • Execution identity: Uproc name and version, Session name and version (if any), MU, launch number, Uproc number, Session number (if any), • Execution attributes: variables, current status , start date and time, the end date and time, execution node, priority, batch queue, user, processing date (if any), re-launch flag . If an error occurs, the error code and error text are returned in the context sequence of the response message and the list is empty.
Operation name	getExecution
Parameters	context: Context [IN/OUT] executionId : ExecutionId [IN]
Input message	**DuwsSEI_getExecution**: the message contains the parameter context and the execution identifier defined by the XML sequence `ExecutionId`.

Output message	**DuwsSEI_getExecutionResponse**: the message contains the characteristics of the selected execution. If an error occurs, the message contains the error code and error text in the context element.
	If the action is performed successfully the description of the selected execution is returned by the web service. The response contains the XML sequence `Execution`:
	`<complexType name="Execution">`
	`<sequence>`
	`<element name="data" nillable="true" type="tns1:ExecutionItem"/>`
	`<element name="variables" nillable="true" type="impl:ArrayOf_tns1_Variable"/>`
	`</sequence>`
	`</complexType>`
	where:
	data: this element contains the same information as that returned by the GetListExecution operation.
	variable: this sequence contains the variables and the values used during the execution. The XML sequence shows that the variables are returned in a list of `Variable` objects:
	`<complexType name="Variable">`
	`<sequence>`
	`<element name="name" nillable="true" type="xsd:string"/>`
	`<element name="origin" nillable="true" type="xsd:string"/>`
	`<element name="type" nillable="true" type="xsd:string"/>`

	<element name="value" nillable="true" type="xsd:string"/> </sequence> </complexType> where: **name**: the name of the variable. A string of 20 characters max. **origin**: the origin of the value can be Uproc, Task or launch. Refer to the Appendix for the codes and their meanings. **type**: the type of the variable is a string that can be a text, a date or a numerical value. Refer to the Appendix to find out the codes and their meanings. **value**: this is the value of the variable. It is a string of 255 characters max. whatever **type** it is.

Figure 16: Operation: getExecution

5.3.4.2 Get the Launch Characteristics from a TASK

Description	This operation returns the attributes used for a future launch creation. These attributes are copied from the description of the Task supplied as a parameter on the Dollar Universe node of the selected Company and Area. The Task must exist in the target environment and cannot be a template task. Invoking this service does not create a launch. The information returned in the response is: • Partial launch identifier: Uproc name and version, Session name and version (if any), MU. • Launch attributes: status, variables, automatic restart flag, start date and time, end date and time, bypass condition check flag, central monitoring flag, launch exclusion window, forced execution flag, execution node, priority, processing date, batch queue, user, step. If an error occurs the error code and error text are sent in the context sequence of the response message.
Operation name	getLaunchFromTask
Parameters	context: Context [IN/OUT] taskId: TaskId [IN]
Input message	**DuwsSEI_getLaunchFromTask**: the message contains the parameter context and the task identifier used as a template. The XML sequence of the identifier is `TaskId`: `<complexType name="TaskId"><sequence><element name="mu" type="string"/><element name="session" type="string"/><element name="sessionVersion" type="string"/><element name="template" type="boolean"/><element name="uproc" type="string"/><element name="uprocVersion" type="string"/></sequence></complexType>` where: **mu**: the name of the management unit. A string of 10 uppercase characters max. **session**: the name of the session. A string of 10 uppercase characters max. This element can be empty if the launch concerns a standalone Uproc. **sessionVersion**: the Session version number. A string of 3 numerical characters, with leading zeros. This element can be empty if the launch concerns a standalone Uproc. **template**: should indicate that the task is not template. It should be set to False. **uproc**: is the name of the header Uproc. A string of 10 uppercase characters max. **uprocVersion**: is the Uproc version number. A string of 3 numerical characters, with leading zeros.
Output message	**DuwsSEI_getLaunchFromTaskResponse**: the message contains the characteristics of a future launch. This data may be used to create a new launch. If

	an error occurrs, the message contains the error code and error text in the context sequence. If the action is successful the description of the selected launch is returned by the web service. The response contains the XML sequence `Launch`: `<complexType name="Launch"> <sequence> <element name="autoRestart" type="boolean"/> <element name="beginDate" type="string"/> <element name="beginHour" type="string"/> <element name="bypassCondCheck" type="boolean"/> <element name="centralControl" type="boolean"/> <element name="endDate" type="string"/> <element name="endHour" type="string"/> <element name="exclusionFrom" type="string"/> <element name="exclusionTo" type="string"/> <element name="forcedExecution" type="boolean"/> <element name="ident" type="tns:LaunchId"/> <element name="node" type="string"/> <element name="priority" type="string"/> <element name="processingDate" type="string"/> <element name="queue" type="string"/> <element name="status" type="string"/> <element name="step" type="string"/> <element name="user" type="string"/> <element name="variables" type="tns:ArrayOfVariable"/> </sequence> </complexType>` where: **autorestart**: if this flag is set to TRUE, the launch will be restarteds automatically if the operating system reboots. If it is set to FALSE, the launch will not be restarted automatically by Dollar Universe when the machine restarts. **beginDate**: the launch submission date. A string - format YYYYMMDD. **beginHour**: the launch submission time. A string - format HHMMSS. **bypassCondCheck**: if this flag is set to TRUE, the conditions are not verified by Dollar Universe before submitting the Uproc. If set to FALSE, all Uproc conditions are verified. **centralControl**: if this flag is set to TRUE, job monitoring data is accessible from the central monitoring node. If FALSE, the execution can only be monitored on the node where it was created. **endDate**: the launch window end date. A string - format YYYYMMDD. **endHour**: the launch window end time . A string - format HHMMSS. **exclusionFrom**: the beginning of the exclusion period in which the execution cannot start. The format of this string is HHMMSS. **exclusionTo**: the end of the exclusion period when the execution cannot start. The

	format of this string is HHMMSS. **forcedExecution**: if set to TRUE, a launch in Event Wait will be forced to run at the end of the launch window. If set to FALSE the launch status will be set to "Time Overrun". **ident**: the launch identifier. Refer to the service Get the launch list for a full description of this element. **node**: the node where the execution will run. A string of 10 characters max. **priority**: the execution priority in the batch queue. A string of 3 numerical characters from 001 to 255. **processingDate**: the associated processing date. The processing date format is YYYYMMDD. **queue**: the batch queue name. The batch queue is a string of 31 characters max. **status**: the launch status is a string of 1 character (not used here). **step**: the internal step of the CL script where the job will start running. The format of the step is a string of 2 numerical characters from 00 to 99 (not used here). **user**: the user submission account. The format is a string of 12 characters max. **variables**: this element contains the launch's list of variables. Each variable is described with the XML sequence `Variable`. Please refer to the full description of a variable item defined for the operation Get execution characteristics .

Figure 17: Operation: getLaunchFromTask

5.3.4.3 Get Launch Characteristics

Description	This operation returns the following details for a launch on the Dollar Universe node of the selected Company and Area: • Launch identifier: Uproc name and version, Session name and version (if any), MU, launch number, Uproc number, Session number (if any), • Launch attributes: status, variables, automatic restart flag, begin date and time, end date and time, bypass condition check flag, central monitoring flag, excluded time interval for launch, forced execution flag, execution node, priority, processing date, batch queue, user, step. If an error occurs, the error code and error text are sent in the context sequence of the response message.
Operation name	getLaunch
Parameters	context: Context [IN/OUT] launchId: LaunchId [IN]
Input message	**getLaunchRequest**: the message contains the parameter context and the launch identifier.
Output message	**getLaunchResponse**: the message contains the characteristics of the selected launch. If an error occurrs, the message contains the error code and error text in the context sequence. If the action is successful, the description of the selected launch is returned by the web service. The response contains the XML sequence Launch: <complexType name="Launch"> <sequence> <element name="autoRestart" type="boolean"/> <element name="beginDate" type="string"/> <element name="beginHour" type="string"/> <element name="bypassCondCheck" type="boolean"/> <element name="centralControl" type="boolean"/> <element name="endDate" type="string"/> <element name="endHour" type="string"/> <element name="exclusionFrom" type="string"/> <element name="exclusionTo" type="string"/> <element name="forcedExecution" type="boolean"/> <element name="ident" type="tns:LaunchId"/> <element name="node" type="string"/>

```
                    <element name="priority"
        type="string"/>
                    <element name="processingDate"
        type="string"/>
                    <element name="queue" type="string"/>
                    <element name="status" type="string"/>
                    <element name="step" type="string"/>
                    <element name="user" type="string"/>
                    <element name="variables"
        type="tns:ArrayOfVariable"/>
                  </sequence>
        </complexType>
```

where:

autorestart: if this flag is set to TRUE, the launch will be restarts automatically if the operating system reboots. If it is set to FALSE, the launch will not be restarted automatically by Dollar Universe when the machine restarts.

beginDate: the launch submission date. A string - format YYYYMMDD.

beginHour: the launch submission time. A string - format HHMMSS.

bypassCondCheck: if this flag is set to TRUE, the conditions are not verified by Dollar Universe before submitting the Uproc. If set to FALSE, all Uproc conditions are verified.

centralControl: if this flag is set to TRUE, job monitoring data is accessible from the central monitoring node. If FALSE, the execution can only be monitored on the node where it was created.

endDate: the launch window end date. A string - format YYYYMMDD.

endHour: the launch window end time . A string - format HHMMSS.

exclusionFrom: the beginning of the exclusion period in which the execution cannot start. The format of this string is HHMMSS.

exclusionTo: the end of the exclusion period when the launch cannot start. The format of this string is HHMMSS.

forcedExecution: if set to TRUE, a launch in Event Wait will be forced to run at the end of the launch window. If set to FALSE the launch status will be set to "Time Overrun".

ident: the launch identifier. Refer to the service Get the launch list for a full description of this element.

node: the node where the execution will run. A string of 10 characters max.

priority: the execution priority in the batch queue. A string of 3 numerical characters from 001 to 255.

processingDate: the associated processing date. The processing date format is YYYYMMDD.

queue: the batch queue name. The batch queue is a string of 31 characters max.

status: the launch status is a string of 1 character (not used here).

step: the internal step of the CL script where the job will start running. The format of the step is a string of 2 numerical characters from 00 to 99 (not used here).

user: the user submission account. The format is a string of 12 characters max.

variables: this element contains the launch's list of variables. Each variable is described with the XML sequence `Variable`. Please refer to the full description of a variable item defined for the operation Get execution characteristics .

Figure 18: Operation: getLaunch

5.3.4.4 Get the Job Log File

Description	This operation returns the log file of a selected execution on the Dollar Universe node of the selected Company and Area. If the job log is found for the execution identifier, the response contains: • Execution identifier: Uproc name and version, Session name and version (if any), MU, launch number, Uproc number, Session number (if any), • Job log returned as a text. If no execution log was found for the selected execution, the error and error message is returned in the context sequence of the response message.
Operation name	getExecutionLog
Parameters	context: Context [IN/OUT] executionId: ExecutionId [IN]
Input message	**DuwsSEI_getExecutionLog**: the message contains the parameter context and the execution identifier which is defined by the XML sequence `ExecutionId`.
Output message	**DuwsSEI_getExecutionLogResponse**: the message contains the log contents of the selected execution. If an error occurs, the message contains the error code and error text in the context element. If the action is successful the log file of the selected job is returned by the web service. The response contains the XML sequence `ExecutionLog`: <complexType name="ExecutionLog"> <sequence> <element name="ident" type="tns:ExecutionId"/> <element name="log" type="tns:ArrayOfstring"/> </sequence> </complexType> where: **ident**: this element contains the execution identifier. For a full description, refer to the description of the operation Get the launch list. **log**: the content of the log file is contained in this element which is an array of strings. Each line of the execution log corresponds to an element of this array.

Figure 19: Operation: getExecutionLog

5.3.4.5 Get the Job Log File as SOAP Attachment

Description	This operation returns the log file of a selected execution on the Dollar Universe node of the selected Company and Area. If the job log is found for the execution identifier, the response contains job log returned as a DataHandler. If no execution log was found for the selected execution, the error and error message is returned in the context sequence of the response message.
Operation name	getExecutionLogasAttachment
Parameters	context: Context [IN] executionId: ExecutionId [IN]

	dh : DataHandler [out]
Input message	**DuwsSEI_getExecutionLogAsAttachment**: the message contains the parameter context and the execution identifier which is defined by the XML sequence `ExecutionId`.
Output message	**DuwsSEI_getExecutionLogAsAttachmentResponse**: the message contains the log contents of the selected execution, as DataHanler (SOAP Attachment) . If an error occurs, the message contains the error code and error text in the context element. If the action is successful the log file of the selected job is returned by the web service as a SOAP Attachment.

5.3.4.6 Get the History Trace of an Execution

Description	This operation returns the engine traces for an execution on the Dollar Universe node of the selected Company and Area. The information returned in the response is: • Execution identifier: Uproc name and version, Session name and version (if any), MU name, launch number, Uproc number, Session number (if any), • Uproc label, • Session label, • MU label, • Text containing the traces. If an error occurs, the error code and error text are sent in the context element of the response message.
Operation name	getHistoryTrace
Parameters	context: Context [IN/OUT] executionId: ExecutionId [IN]
Input message	**DuwsSEI_getHistoryTrace**: the message contains the parameter context and the execution identifier.
Output message	**DuwsSEI_getHistoryTraceResponse**: the message contains the traces generated by the engines for the selected execution. If an error occurrs, the message contains the error code and error text in the context element. If the action is successful the trace is returned by the web service. The response contains the XML sequence `HistoryTrace`: `<complexType name="HistoryTrace"><sequence><element name="ident" type="tns:ExecutionId"/><element name="muLabel" type="string"/><element name="sessionLabel" type="string"/><element name="trace" type="tns:ArrayOfstring"/><element name="uprocLabel" type="string"/></sequence></complexType>` where: **ident**: this element recalls the execution identifier. **muLabel**: the execution management unit label. A string of 20 characters max. This information can be useful for building user-friendly interfaces. **sessionLabel**: execution session label. A string of 20 characters max. If no session is involved in the job, this element is an empty string. This information can be useful for building user-friendly interfaces. **trace**: contains the requested trace in an array of strings. Each text line (80 characters max) corresponds to an element of the array. The number of lines

	depends on the execution management. **uprocLabel**: executed Uproc label. A string of 60 characters max. This information can be useful for building user-friendly interfaces.

Figure 20: Operation: getHistoryTrace

5.3.4.7 Get an Execution's Previous Launches

Description	This operation returns the previous launches for the selected execution on the Dollar Universe node of the selected Company and Area. The information returned in the response is a list of previous launch attempts common to the same execution. For each previous launch the data provided by this operation is: • Launch date and time, • Uproc execution number, • Uproc name, • Initiator of the launch action: 'R' for the operator, 'E' for the Launcher. If an error occurs, the error code and error text are sent in the context element of the response message. The execution identifier must correspond to an existing execution in the targeted environment.
Operation name	getPreviousLaunches
Parameters	context: Context [IN/OUT] executionId: ExecutionId [IN]
Input message	**DuwsSEI_getPreviousLaunches**: the message contains the parameter context and the execution identifier.
Output message	**DuwsSEI_getPreviousLaunchesResponse**: the message contains the previous launches of the execution selected in the request message. If an error occurs, this message contains the error code and error text in the context element. If the action is successful the previous launches of the selected execution are returned by the web service. In the response, the previous launches of the selected execution are described by the XML sequence `PreviousLaunch`: `<complexType name="PreviousLaunch"><sequence><element name="launchDate" type="string"/><element name="launchHour" type="string"/><element name="numProc" type="string"/><element name="type" type="string"/><element name="uproc" type="string"/></sequence></complexType>` where: **launchDate**: the date of the launch. The format of this string is YYYMMDD. **launchHour**: the time of the launch. The format of this string is HHMMSS. **numProc**: for the current launch element, this is the uproc execution number. The number is stored as a string of 7 numerical characters with leading zeros. **type**: this flag signals if the job has been re-launched by the operator or by the Launcher. This 1 character string can be `'R'` for the operator or `'E'` for the launch

	engine. **uproc**: the name of the Uproc executed by the job.

Figure 21: Operation: getPreviousLaunches

5.3.4.8 Get Event Characteristics

Description	This operation returns the attributes of the selected event on the Dollar Universe node of the selected Company and Area. The information returned in the response is: • Event identifier: Uproc name, Session name (if any), MU, Uproc number, Session number (if any), processing date, user, • Creation date and time, last update date and time of the event, job status, and step. If an error occurs, the error code and error text are sent in the context element of the response message.
Operation name	getEvent
Parameters	context: Context [IN/OUT] eventId: EventId [IN]
Input message	**DuwsSEI_getEvent**: the message contains the parameter context and the event identifier to be retrieved. The XML sequence of the event identifier is `EventId`.

Output message	**DuwsSEI_getEventResponse**: the message contains the attributes of the event selected in the request message. If an error occurs, this message contains the error code and error text in the context element.
	If the action is performed successfully the attributes of the selected event are returned by the Web service. In the response, the selected event is described by the XML sequence `Event`:
	`<complexType name="Event">`
	`<sequence>`
	`<element name="creationDate" type="string"/>`
	`<element name="creationHour" type="string"/>`
	`<element name="ident" type="tns:EventId"/>`
	`<element name="status" type="string"/>`
	`<element name="step" type="string"/>`
	`<element name="updateDate" type="string"/>`
	`<element name="updateHour" type="string"/>`
	`</sequence>`
	`</complexType>`
	where:
	creationDate: the occurrence date of the event. This format of this string is YYYYMMDD.
	creationHour: the occurrence time of the event. This format of this string is HHMMSS.

	ident: the event identifier. Refer to the description of the operation GetList Events for more details of this field. **status**: job status of the event. A string of 1 character which can be T (Completed), I (Aborted), E (Running), A (Pending) or D (Started). See the Appendix for the meaning of the status codes. **step**: designates the execution step corresponding to the event. The format is a string of 2 numerical characters from 00 to 99. **updateDate**: the date of the last update action on the event. The string format is YYYYMMDD. **updateHour**: the time of the last update action on the event. The string format is HHMMSS.

Figure 22: Operation: getEvent

5.3.5 Operations that Perform Actions on IT Operations

5.3.5.1 Create a Launch

Description	This operation creates a launch on the Dollar Universe node of the selected Company and Area. If the operation runs without errors, the service returns the new launch identifier. The information returned in the response is: • Uproc name and version, • Session name and version (if any), • MU, • Launch number, • Uproc number, • Session number (if any). If an error occurs, the error code and error text are sent in the context element of the response message.
Operation name	addLaunch
Parameters	context: Context [IN/OUT] launch: Launch [IN]
Input message	**DuwsSEI_addLaunch**: the message contains the parameter context and the task identifier used as template. The XML sequence of the launch characteristics is `Launch`:

```
<complexType name="Launch">
        <sequence>
          <element name="autoRestart"
type="boolean"/>
          <element name="beginDate"
type="string"/>
          <element name="beginHour"
type="string"/>
          <element name="bypassCondCheck"
type="boolean"/>
          <element name="centralControl"
type="boolean"/>
          <element name="endDate" type="string"/>
          <element name="endHour" type="string"/>
          <element name="exclusionFrom"
type="string"/>
          <element name="exclusionTo"
type="string"/>
          <element name="forcedExecution"
type="boolean"/>
          <element name="ident"
type="tns:LaunchId"/>
          <element name="node" type="string"/>
          <element name="priority"
```

	type="string"/> <element name="processingDate" type="string"/> <element name="queue" type="string"/> <element name="status" type="string"/> <element name="step" type="string"/> <element name="user" type="string"/> <element name="variables" type="tns:ArrayOfVariable"/> </sequence> </complexType> where: **autorestart**: if this flag is set to TRUE, the launch will be restarts automatically if the operating system reboots. If it is set to FALSE, the launch will not be restarted automatically by Dollar Universe when the machine restarts. **beginDate**: the launch submission date. A string - format YYYYMMDD. **beginHour**: the launch submission time. A string - format HHMMSS. **bypassCondCheck**: if this flag is set to TRUE, the conditions are not verified by Dollar Universe before submitting the Uproc. If set to FALSE, all Uproc conditions are verified. **centralControl**: if this flag is set to TRUE, job monitoring data is accessible from the central monitoring node. If FALSE, the execution can only be monitored on the node where it was created. **endDate**: the launch window end date. A string - format YYYYMMDD. **endHour**: the launch window end time . A string - format HHMMSS. **exclusionFrom**: the beginning of the exclusion period in which the execution cannot start. The format of this string is HHMMSS. **exclusionTo**: the end of the exclusion period when the launch cannot start. The format of this string is HHMMSS. **forcedExecution**: if set to TRUE, a launch in Event Wait will be forced to run at the end of the launch window. If set to FALSE the launch status will be set to "Time Overrun". **ident**: the launch identifier. Refer to the service Get the launch list for a full description of this element. **node**: the node where the execution will run. A string of 10 characters max. **priority**: the execution priority in the batch queue. A string of 3 numerical characters from 001 to 255. **processingDate**: the associated processing date. The processing date format is YYYYMMDD. **queue**: the batch queue name. The batch queue is a string of 31 characters max. **status**: the launch status is a string of 1 character (not used here). **step**: the internal step of the CL script where the job will start running. The format of the step is a string of 2 numerical characters from 00 to 99 (not used here). **user**: the user submission account. The format is a string of 12 characters max. **variables**: this element contains the launch's list of variables. Each variable is described with the XML sequence `Variable`. Please refer to the full description of a variable item defined for the operation Get execution characteristics . Note: If the variable list contains variables that are not defined in the Uproc, these variables are ignored during the launch creation.
Output message	**DuwsSEI_addLaunchResponse**: If the action is successful the message contains the identifier of the new launch created. If an error occurs, this message contains the error code and error text in the context element.

	In the response, the launch identifier is described by the XML sequence `LaunchId`. For more details about the attributes returned by new launch, refer to the operation Get the launch list.

Figure 23: Operation: addLaunch

5.3.5.2 Create a Launch from a Task

Description	This operation submits a task on the Dollar Universe node of the selected Company and Area. If the submission is correctly performed, the service returns the identity of the new launch generated by the task submission. The information returned in the response is: • Uproc name and version, • Session name and version (if any), • MU, • Launch number, • Uproc number, • Session number (if any). If an error occurs, the error code and error text are sent in the context element of the response message.
Operation name	addLaunchFromTask
Parameters	context: Context [IN/OUT] taskId: TaskId [IN] In the request the task is described by the XML sequence `TaskId`.
Input message	**DuwsSEI_addLaunchFromTask**: the message contains the parameter context and the task identifier.
Output message	**DuwsSEI_addLaunchFromTaskResponse**: If the action is successful the message contains the identifier of the new launch. If an error occurs, this message contains the error code and error text in the context element. In the response, the launch is described by the XML sequence `LaunchId`. For more details about the attributes returned by new launch, refer to the operation Get the launch list.

Figure 24: Operation: addLaunchFromTask

5.3.5.3 Create a Launch from a Task (Providing Uproc Variables)

Description	This operation submits a task on the Dollar Universe node of the selected Company and Area. This operation executes the same action as the **addLaunchFromTask** operation but allows the invoker to override existing variable values with the values supplied as parameters. If the submission is correctly performed, the operation returns the new launch identity. The information returned in the response is: • Uproc name and version, • Session name and version (if any), • MU, • Launch number, • Uproc number, • Session number (if any). If an error occurs, the error code and error text are sent in the context element of the response message.
Operation name	addLaunchFromTask2
Parameters	context: Context [IN/OUT] taskId: TaskId [IN] variableArray: ArrayOfVariables [IN] In the request, the task is described by the XML sequence `TaskId`, and the variable list is described by the XML type `ArrayOfVariable`. The XML sequence of the variable list characteristics is `Variables`: `<complexType name="Variable"><sequence><element name="name" nillable="true" type="xsd:string"/><element name="origin" nillable="true" type="xsd:string"/><element name="type" nillable="true" type="xsd:string"/><element name="value" nillable="true" type="xsd:string"/></sequence></complexType>` where: **name**: the name of the variable. A string of 20 characters max. **origin**: the origin of the value can be Uproc, Task or launch. Refer to the Appendix for the codes and their meanings. to find out the codes and their meanings. **type**: the type of the variable is a string that can be a text, a date or a numerical value. Refer to the Appendix for the codes and their meanings **value**: the value of the variable. A string of 255 characters max. Whatever the **type**.
Input message	**DuwsSEI_addLaunchFromTask2**: the message contains the parameter context, the task identifier and the variable list.
Output message	**DuwsSEI_addLaunchFromTask2Response**: If the action is successful the message contains the new launch identifier. If an error occurs, this message

	contains the error code and error text in the context element. In the response, the launch is described by the XML sequence `LaunchId`. For more details about the attributes returned by a new launch, refer to the operation Get the launch list.

Figure 25: Operation: addLaunchFromTask2

5.3.5.4 Delete a Launch

Description	This operation deletes a launch from the Dollar Universe node of the selected Company and Area. The launch identifier is sent in the request message and contains: • Uproc name and version, • Session name and version (if any), • MU, • Launch number, • Uproc number, • Session number (if any). If an error occurs, the error code and error text are sent in the context element of the response message and no launch is deleted.
Operation name	deleteLaunch
Parameters	context: Context [IN/OUT] launch: LaunchId [IN] In the request the launch is described by the XML sequence `LaunchId`.
Input message	**DuwsSEI_deleteLaunch**: the message contains the parameter context and the identifier of the launch to delete.
Output message	**DuwsSEI_deleteLaunchResponse**: if an error occurs, this message contains the error code and error text in the context sequence (elements `errorCode` and `message` of the sequence `error`). If the action is successful, the `errorCode` is equal to 1 and the `message` element is empty.

Figure 26: Operation: deleteLaunch

5.3.5.5 Disable a Launch

Description	This operation suspends a launch on the Dollar Universe node of the selected Company and Area. The launch identifier is sent in the request message and contains: • Uproc name and version, • Session name and version (if any), • MU, • Launch number, • Uproc number, • Session number (if any). If an error occurs, the error code and error text are sent in the context element of the response message and no launch is suspended.
Operation name	disableLaunch
Parameters	context: Context [IN/OUT] launchId: LaunchId [IN] In the request the launch is described by the XML sequence `LaunchId`.
Input message	**DuwsSEI_disableLaunch**: the message contains the parameter context and the identifier of the launch to suspend.
Output message	**DuwsSEI_disableLaunchResponse**: if an error occurs, this message contains the error code and error text in the `context` sequence (elements `errorCode` and `message` of the sequence `error`). If the action is successful, the `errorCode` is equal to 1 and the `message` member is empty.

Figure 27: Operation: disableLaunch

5.3.5.6 Enable a Launch

Description	This operation enables (releases) a launch on the Dollar Universe node of the selected Company and Area. The launch identifier is sent in the request message: • Uproc name and version, • Session name and version (if any), • MU, • Launch number, • Uproc number, • Session number (if any). If an error occurs, the error code and error text are sent in the context element of the response message and no launch is released.
Operation name	enableLaunch
Parameters	context: Context [IN/OUT] launchId: LaunchId [IN] In the request the launch is described by the XML sequence `LaunchId`.
Input message	**DuwsSEI_enableLaunch**: the message contains the parameter context and the identifier of the launch to enable.
Output message	**DuwsSEI_enableLaunchResponse**: if an error occurs, this message contains the error code and error text in the `context` sequence (elements `errorCode` and `message` of the sequence `error`). If the action is successful, the `errorCode` is equal to 1 and the `message` member is empty.

Figure 28: Operation: enableLaunch

5.3.5.7 Purge an Execution

Description	This operation purges the selected execution on the Dollar Universe node of the selected Company and Area. The execution identifier is sent in the request message: • Uproc name and version, • Session name and version (if any), • MU, • Launch number, • Uproc number, • Session number (if any). If an error occurs, the error code and error text are sent in the context element of the response message and no job is deleted.
Operation name	purgeExecution
Parameters	context: Context [IN/OUT] executionId: ExecutionId [IN]
Input message	**DuwsSEI_purgeExecution**: the message contains the parameter context and the identifier of the execution to delete.
Output message	**DuwsSEI_purgeExecutionResponse**: if an error occurs, this message contains the error code and error text in the `context` sequence (elements `errorCode` and `message` of the sequence `error`). If the action is successful, the `errorCode` is equal to 1 and the `message` member is empty.

Figure 29: Operation: purgeExecution

5.3.5.8 Stop an Execution

Description	This operation aborts the selected execution on the Dollar Universe node of the selected Company and Area. The execution identifier is sent in the request message: • Uproc name and version, • Session name and version (if any), • MU, • Launch number, • Uproc number, • Session number (if any). An amount of time (delay parameter) is also supplied as a parameter to delay the abort action. The delay is an integer value (6 digits) expressed in seconds. If an error occurs, the error code and error text are sent in the context element of the response message and no execution is stopped.
Operation name	stopExecution
Parameters	Context: Context [IN/OUT] executionId: ExecutionId [IN] delay: integer [IN]
Input message	**DuwsSEI_stopExecution**: the message contains the parameter context, the identifier of the job to stop and the delay.
Output message	**DuwsSEI_stopExecutionResponse**: if an error occurs, this message contains the error code and error text in the `context` sequence (elements `errorCode` and `message` of the sequence `error`). If the action is successful, the `errorCode` is equal to 1 and the `message` member is empty.

Figure 30: Operation: stopExecution

5.3.5.9 Create an Event

Description	This operation creates an event on the Dollar Universe node of the selected Company and Area. The event identifier and characteristics are sent in the request message: The identifier is given by: • Uproc name, • Session name (if any) : this member can be an empty string, • MU, • Processing date, • Session number (if any). • User The attributes are: • Status, • Step number (optional). The event status is a string of 1 character whose value can be T, I, E, A or D. Refer to the Appendix for the meaning of the codes. The event step is a string of 2 numerical characters, (padded). If the action is successful, no error and no additional information is returned. A new event is created by Dollar Universe. If an error occurs, the error code and error text are sent in the context element of the response message and no event is created.
Operation name	addEvent
Parameters	context: Context [IN/OUT] eventId: Event [IN]
Input message	**DuwsSEI_addEvent**: the message contains the parameter context, the event identifier to create.
Output message	**DuwsSEI_addEventResponse**: If an error occurs, this message contains the error code and error text in the `context` sequence (elements `errorCode` and `message` of the sequence `error`). If the action is successful, the `errorCode` is equal to 1 and the `message` member is empty. No data is returned by this operation.

Figure 31: Operation: addEvent

5.3.5.10 Update an Event

Description	This operation updates the selected event on the Dollar Universe node of the selected Company and Area. The event identifier is sent in the request message: • Uproc name, • Session name (if any), • MU, • Processing date, • Uproc number, • Session number (if any), • User. Only the status and step number of an event can be updated. The new event status and step are also supplied as parameters. The new event status is a 1 character string whose value can be T, I, E, A or D. Refer to the Appendix on page 88 to find out the meaning of the codes. The new event step is a 2 numerical character string with leading zeros. If an error occurs, the error code and error text are sent in the context element of the response message and no event is updated.
Operation name	updateEvent
Parameters	context: Context [IN/OUT] eventId: EventId [IN]
Input message	**DuwsSEI_updateEvent**: the message contains the parameter context and the event identifier to update.
Output message	**DuwsSEI_updateEventResponse**: If an error occurs, this message contains the error code and error text in the `context` sequence (elements `errorCode` and `message` of the sequence `error`). If the action is successful, the `errorCode` is equal to 1 and the `message` member is empty.

Figure 32: Operation: updateEvent

5.3.5.11 Delete an Event

Description	This operation deletes the selected event on the Dollar Universe node of the selected Company and Area. The event identifier is sent in the request message: • Uproc name, • Session name (if any), • MU, • Processing date, • Uproc number, • Session number (if any), • User. If an error occurs, the error code and error text are sent in the context element of the response message and no event is deleted.
Operation name	deleteEvent
Parameters	context: Context [IN/OUT] eventId: EventId [IN]
Input message	**DuwsSEI_deleteEvent**: the message contains the parameter context and the event identifier to delete.
Output message	**DuwsSEI_deleteEventResponse**: If an error occurs, this message contains the error code and error text in the `context` sequence (elements `errorCode` and `message` of the sequence `error`). If the action is successful, the `errorCode` is equal to 1 and the `message` member is empty.

Figure 33: Operation: delete an Event

6. Deployment example 1: Tomcat and Axis

This section resumes the procedure for deploying Dollar Universe Web Services 2.0 using Apache Tomcat (as a container and server) and Apache Axis.

6.1 Software Pre-requisites

- Tomcat 5.5 or higher
- Axis 1: 1.4
- JRE 1.5
- Archive manipulation software

6.2 First Installation Steps

Install Apache Tomcat and Apache Axis as described in their respective user manuals.

6.3 Installation Overview

The Dollar Universe Web Services enterprise archive: **duws210.ear** cannot be directly deployed within Axis. The procedure can be divided into three main steps:

- **Active code installation**: Extract the executable code from the archive and make it accessible to Axis.
- **Method registration**: Available Dollar Universe Web Services functions have to be declared as operations of the web service.
- **Configuration**: Dollar Universe Web Services requires a little more information to start working (log file declaration, TCP/IP service mapping, etc).

The procedures described below should be run on the same machine as Axis and Tomcat.

6.3.1 Active Code Installation

Bear in mind that JAR, WAR and EAR archives are ZIP archives. Use your favorite compression utility to extract content from ***duws210.ear***.

At the first level is a *meta-inf* directory and a *duws20.war* file. Extract all files from the *duws.war* archive.

A meta-inf and a *web-inf* directory will be created. Check that the *web-inf* directory contains all files listed in section Contents of the duws210.ear. The web service is contained in the JAR files. Copy JAR files to the Axis lib directory (usually *webapps/axis/WEB-INF/lib*).

6.3.2 Method Registration

A number of environment variables must be declared.

6.3.2.1 Axis Variable Declaration on UNIX

On UNIX, save the following instructions in a script, for example *setenv_axis.ksh,* and customize the AXIS_HOME variable. Run the script *setenv_axis.ksh* from the command line:

```
AXIS_HOME=<path to Axis>
AXIS_LIB=$AXIS_HOME/lib
AXISCLASSPATH=$AXIS_LIB/axis.jar:
AXISCLASSPATH=$AXISCLASSPATH:$AXIS_LIB/commons-discovery-0.2.jar
AXISCLASSPATH=$AXISCLASSPATH:$AXIS_LIB/commons-logging-1.0.4.jar
AXISCLASSPATH=$AXISCLASSPATH:$AXIS_LIB/jaxrpc.jar
AXISCLASSPATH=$AXISCLASSPATH:$AXIS_LIB/saaj.jar
AXISCLASSPATH=$AXISCLASSPATH:$AXIS_LIB/log4j-1.2.8.jar
AXISCLASSPATH=$AXISCLASSPATH:$AXIS_LIB/xml-apis.jar
AXISCLASSPATH=$AXISCLASSPATH:$AXIS_LIB/xercesImpl.jar

export AXIS_HOME
export AXIS_LIB
export AXISCLASSPATH
```

6.3.2.2 Axis Variable Declaration on Windows

On Windows, save the following commands in a file, for example *setenv_axis.cmd*, and customize the AXIS_HOME variable. Run the script *setenv_axis.cmd* from a DOS command window:

```
set AXIS_HOME=<path to Axis>
set AXIS_LIB=%AXIS_HOME%\lib
set AXISCLASSPATH=%AXIS_LIB%\axis.jar
set AXISCLASSPATH=%AXISCLASSPATH%;%AXIS_LIB%\commons-discovery-0.2.jar
set AXISCLASSPATH=%AXISCLASSPATH%;%AXIS_LIB%\commons-logging-1.0.4.jar
set AXISCLASSPATH=%AXISCLASSPATH%;%AXIS_LIB%\jaxrpc.jar
set AXISCLASSPATH=%AXISCLASSPATH%;%AXIS_LIB%\saaj.jar;
set AXISCLASSPATH=%AXISCLASSPATH%;%AXIS_LIB%\log4j-1.2.8.jar
set AXISCLASSPATH=%AXISCLASSPATH%;%AXIS_LIB%\xml-apis.jar
```

```
set AXISCLASSPATH=%AXISCLASSPATH%;%AXIS_LIB%\xercesImpl.jar
```

6.3.2.3 Declare Exposed Methods to Axis

Next, declare the list of methods to be exposed by Axis. Save the following lines in a file, for example *deploy.wsdd*.

In this example, the Axis deployment descriptor uses the RPC/encoded style which is not WS-I Basic Profile compliant.

Note: For further details about WSDD file and how to deploy and undeploy a web service with Axis, refer to the Axis reference manual.

```
<deployment
    xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
    xmlns:ns1="http://xml.apache.org/xml-soap">
  <service name="DuwsService" provider="java:RPC" style="rpc"
use="encoded">
    <parameter name="className" value="com.orsyp.duws.DuwsImpl"/>
      <parameter name="wsdlTargetNamespace"
value="http://duws.orsyp.com"/>
      <parameter name="wsdlServiceElement" value="DuwsService"/>
      <parameter name="wsdlServicePort" value="DuwsSEIPort"/>
      <parameter name="className" value="com.orsyp.duws.DuwsImpl"/>
      <parameter name="wsdlPortType" value="DuwsSEI"/>
         <parameter name="cacheAttachments"
locked="false">true</parameter>
        <parameter name="sizeThreshold" locked="false">40000</parameter>
       <parameter name="enableSwA" locked="false">true</parameter>
       <parameter name ="dh" value="ns1:DataHandler"/>

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:Context"
        type="java:com.orsyp.duws.Context"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:Envir"
        type="java:com.orsyp.duws.Envir"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
```

```
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:UniError"
        type="java:com.orsyp.duws.UniError"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:TaskFilter"
        type="java:com.orsyp.duws.TaskFilter"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:TaskId"
        type="java:com.orsyp.duws.TaskId"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:TaskItem"
        type="java:com.orsyp.duws.TaskItem"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:DuwsException"
        type="java:com.orsyp.duws.DuwsException"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:LaunchId"
        type="java:com.orsyp.duws.LaunchId"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
```

```
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:LaunchFilter"
        type="java:com.orsyp.duws.LaunchFilter"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:LaunchItem"
        type="java:com.orsyp.duws.LaunchItem"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:UprocId"
        type="java:com.orsyp.duws.UprocId"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:UprocFilter"
        type="java:com.orsyp.duws.UprocFilter"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:UprocItem"
        type="java:com.orsyp.duws.UprocItem"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
```

```
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:SessionId"
        type="java:com.orsyp.duws.SessionId"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:SessionFilter"
        type="java:com.orsyp.duws.SessionFilter"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:SessionItem"
        type="java:com.orsyp.duws.SessionItem"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:MUnitId"
        type="java:com.orsyp.duws.MUId"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:MUnitFilter"
        type="java:com.orsyp.duws.MUFilter"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:MUnitItem"
        type="java:com.orsyp.duws.MUItem"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
```

```
deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:EventId"
        type="java:com.orsyp.duws.EventId"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:EventFilter"
        type="java:com.orsyp.duws.EventFilter"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:EventItem"
        type="java:com.orsyp.duws.EventItem"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:ExecutionId"
        type="java:com.orsyp.duws.ExecutionId"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:ExecutionFilter"
        type="java:com.orsyp.duws.ExecutionFilter"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
```

```
      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:ExecutionItem"
        type="java:com.orsyp.duws.ExecutionItem"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:HistoryTrace"
        type="java:com.orsyp.duws.HistoryTrace"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:Variable"
        type="java:com.orsyp.duws.Variable"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:Execution"
        type="java:com.orsyp.duws.Execution"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:Launch"
        type="java:com.orsyp.duws.Launch"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:NodeFilter"
```

```
        type="java:com.orsyp.duws.NodeFilter"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:NodeId"
        type="java:com.orsyp.duws.NodeId"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:NodeItem"
        type="java:com.orsyp.duws.NodeItem"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />

      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:ExecutionLog"
        type="java:com.orsyp.duws.ExecutionLog"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
       <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:ExecutionLogAttach"
        type="java:com.orsyp.duws.ExecutionLogAttach"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:PreviousLaunch"
        type="java:com.orsyp.duws.PreviousLaunch"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      />
```

```
      <typeMapping
        xmlns:ns="http://duws.orsyp.com"
        qname="ns:Event"
        type="java:com.orsyp.duws.Event"
        serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"

deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       />
      <typeMapping
           qname="ns1:DataHandler"
           type="java:javax.activation.DataHandler"

    serializer="org.apache.axis.encoding.ser.JAFDataHandlerSerializerFact
ory"

deserializer="org.apache.axis.encoding.ser.JAFDataHandlerDeserializerFact
ory"
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  </service>
</deployment>
```

Figure 34: AXIS deployment descriptor file

6.3.3 Deploy the Dollar Universe Web Service

When the WSDD file is ready for deployment, execute the following command line to deploy ***duws20*** as a web service.

On UNIX, execute the following command:

```
java -cp "$AXISCLASSPATH" org.apache.axis.client.AdminClient -
lhttp://localhost:8080/axis/services/AdminService deploy.wsdd
```

On Windows, execute the following command:

```
java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient -
lhttp://localhost:8080/axis/services/AdminService deploy.wsdd
```

If the web service deployment is successful, the following output should be displayed:

```
C:\axis>java -cp %AXISCLASSPATH% org.apache.axis.client.AdminClient -
lhttp://localhost:8080/axis/services/AdminService deploy.wsdd

log4j:WARN No appenders could be found for logger
(org.apache.axis.components.threadpool.TaskManagerFactory).
log4j:WARN Please initialize the log4j system properly.
Processing file deploy.wsdd
<Admin>Done processing</Admin>
```

The list of available functions will be displayed at http://localhost:8080/axis/servlet/AxisServlet

6.3.4 In Case of Problems

Go to the URL http://localhost:8080/axis/servlet/AxisServlet.

If ***DuwsService*** is missing from the list, check the “Axis Happiness Page” at http://localhost:8080/axis/happyaxis.jsp

This should indicate clearly if Axis is in running order. Look for major errors such as:

```
Error: could not find class javax.activation.DataHandler from file
activation.jar
Axis will not work.
See http://java.sun.com/products/javabeans/glasgow/jaf.html
```

Note: The Sun Java website link above should provide clear instructions for solving any errors displayed on the Happiness Page. In this particular case, it was simply necessary to download the activation jar and copy it to the Axis lib directory, then reload the Axis Web application.

Once all the problems have been fixed, rerun the deployment command (Deploy the Dollar Universe web service).

The following web services should be displayed at http://localhost:8080/axis/servlet/AxisServlet:

6.3.5 List of DUWS Declared Operation

And now... Some Services

- DuwsService *(wsdl)*
  - getEvent
  - getListNode
  - getListMU
  - getListUproc
  - getListSession
  - getListTask
  - getLaunchFromTask
  - addLaunchFromTask
  - addLaunchFromTask2
  - getListLaunch
  - getLaunch
  - deleteLaunch
  - disableLaunch
  - enableLaunch
  - addLaunch
  - getListExecution
  - purgeExecution
  - stopExecution
  - getHistoryTrace
  - getExecution
  - getExecutionLog
  - getPreviousLaunches
  - getListEvent
  - deleteEvent
  - addEvent
  - updateEvent
- AdminService *(wsdl)*
  - AdminService
- Version *(wsdl)*
  - getVersion

6.4 Tomcat and Axis Configuration

A minimum amount of configuration is required before the web service cab function correctly.

- First the Dollar Universe Web Services configuration file must be prepared and declared. See section 0 The web service configuration file.
- Then a security profile must be defined for Dollar Universe Web Services. Declaration of the user role in Tomcat is detailed below.

6.4.1 Tomcat Users and Role

In the Tomcat user file, declare the ***DollarUniverseUser*** role and associate a Tomcat user to this role.

The Tomcat user file is ***conf/tomcat-users.xml*** under the Tomcat installation directory:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>
  <role rolename="admin"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="admin" password="admin" roles="admin,manager"/>
</tomcat-users>
```

To add the *DollarUniverseUser* role and give it to the Tomcat user, add the following XML element as a « **tomcat-users** » XML child:

```
<role rolename="DollarUniverseUser"/>
```

Modify the « roles » attribute corresponding to the username « tomcat » so that it becomes:

```
roles="tomcat,DollarUniverseUser"
```

The resulting file should look like this:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>
  <role rolename="admin"/>
  <role rolename="DollarUniverseUser"/>
  <user username="tomcat" password="tomcat"
roles="tomcat,DollarUniverseUser"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
```

```
  <user username="role1" password="tomcat" roles="role1"/>
  <user username="admin" password="admin" roles="admin,manager"/>
</tomcat-users>
```

6.4.2 Axis web.xml Configuration

Add the following XML elements concerning Dollar Universe Web Services as "**web-app**" XML element children in the Axis *web.xml* file.

Set the parameter *configFile* to a valid file name. This file contains the Dollar Universe Web Services configuration. Replace the keyword `ENTER_YOUR_PATH` with the full path name of the web service configuration file.

```
<context-param>
   <description>Configuration file</description>
   <param-name>configFile</param-name>
   <param-value>ENTER_YOUR_PATH</param-value>
</context-param>
<context-param>
   <param-name>debug</param-name>
   <param-value>0</param-value>
</context-param>
<servlet>
   <display-name>DuwsImpl</display-name>
   <servlet-name>DuwsImpl</servlet-name>
   <servlet-class>com.orsyp.duws.DuwsImpl</servlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
   <servlet-name>DuwsImpl</servlet-name>
   <url-pattern>/duws</url-pattern>
</servlet-mapping>

<jsp-config/>

<security-constraint>
   <display-name>SecurityConstraint</display-name>
   <web-resource-collection>
      <web-resource-name>Dollar Universe Web Services</web-resource-name>
      <url-pattern> /services/duws/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>DollarUniverseUser</role-name>
   </auth-constraint>
   <user-data-constraint>
      <transport-guarantee>NONE</transport-guarantee>
   </user-data-constraint>
</security-constraint>

<login-config>
   <auth-method>BASIC</auth-method>
   <realm-name>DollarUniverse</realm-name>
```

```
</login-config>

<security-role>
   <description>Dollar Universe Web User</description>
   <role-name>DollarUniverseUser</role-name>
</security-role>
```

Once the Axis web.xml file is updated, reload the Axis Web application with the Tomcat Manager application.

6.4.2.1 Security Profile Check

To check whether the security profile is correctly taken into account by a simple WSDL retrieval, in your Web navigator key in http://host:port/axis/services/duws?WSDL

If the server prompts for a *user/password* enter the user/password associated with the ***DollarUniverseUser*** role. If access is granted, the WSDL of Dollar Universe Web Services is displayed. This shows that the Basic authentication is correctly set.

6.4.3 Using getExecutionLogAsAttachment with Axis

Dollar Universe Web Services uses SOAP with Attachments Java API to send the execution log files requested through this method. The execution log file is added as an attachment to the response message.

First, Dollar Universe Web Services retrieves the execution log file from the targeted Dollar Universe node and stores it as a temporary file on the machine where it is running. Then the SOAP attachment is created using this temporary local file.

It is possible to configure the web service to select the location of these temporary files by adding the property **com.orsyp.duws.joblog.tempdir** to the DUWS configuration file. For example, on Windows:

```
com.orsyp.duws.joblog.tempdir=C://Temp//
```

If this property is not declared in the DUWS configuration file, the temporary local file is stored by default in the Java temporary directory specified by the Java property **java.io.tempdir**.

The sample client code below retrieves the execution log as an attachment and writes the log into a local file.

```
DataHandler dh =
         service.getExecutionLogasAttachment(holder,getExecutionId());

FileOutputStream fout = new FileOutputStream("Extractedlog.txt");
dh.writeout(fout);
```

7. Deployment example 2: JBoss on Windows

7.1 Software Pre-requisites

- JBoss 4.2.2 GA
- JDK 1.5 and JVM 1.5
- ZIP archive manipulation software

7.2 Conventions

Please note that windows shell variables will be used throughout this section to refer to particular values. These variables are:

- %JBOSS_INSTALL_DIR% : path to the JBoss installation directory
- %JDK_INSTALL_DIR% : path to the JDK installation directory

Moreover, it is assumed that all path values are entered without a trailing '\' character. Furthermore, it is considered that the web service will be deployed in the default server configuration (%JBOSS_INSTALL_DIR%\server\default)

7.3 First Installation Steps

Install JBoss and JDK 1.5 as described in their respective user manuals.

Edit the ***run.bat*** script which can be found at %JBOSS_INSTALL_DIR%\bin directory to add the path to the bin JDK 1.5 directory to %PATH%:

```
Set PATH=%JDK_INSTALL_DIR%\bin
```

7.4 duws.war Modification

This section is specific to JBoss. For further details, refer to the section *Security on JBoss* in the JBoss user guide.

Several configuration files will be added to the **duws.war** archive in order to tell JBoss about the security settings regarding Dollar Universe Web Services.

This section explains:

- How to declare JBoss users
- How to assign them the role DollarUniverseUser.

Use your favorite compression utility to extract content from ***duws210.ear***.

At the first level is a *meta-inf* directory and a ***duws.war*** file.

Extract all files from the ***duws.war*** archive.

A *meta-inf* and a *web-inf* directory will be created. Check that the *web-inf* directory contains all files listed in section 0 Contents of the duws210.ear.

Create and add the following *jboss-web.xml* file to the *duws* war's WEB-INF directory:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE jboss-web
    PUBLIC "-//JBoss//DTD Web Application 2.3V2//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-web_3_2.dtd">
<jboss-web>
    <security-domain>java:/jaas/duws</security-domain>
</jboss-web>
```

Create a directory named **classes** in the duws.war WEB-INF.

Create a file named ***duws-users.properties*** to the directory ***classes***. This file declares those JBoss users allowed to access Dollar Universe Web Services. Declare JBoss users using the following file format:

```
user=password
```

In our example, we define a user *hermione* using *secret* as password.

```
hermione=secret
```

Create a file named ***duws-roles.properties*** in the same ***classes*** directory. This file defines JBoss user/role relationships. Declare roles assigned to JBoss users using the following file format:

```
user=role
```

In our example, the user *hermione* is assigned the role *DollarUniverseUser*.

```
hermione=DollarUniverseUser
```

However, JBoss is not yet able to use these files. This will be adjusted later in the installation procedure.

Finally indicate the path to the Dollar Universe Web Services configuration file:

- Edit the *web.xml* file of the *duws.war* archive.
- Replace the **ENTER_YOUR_PATH** keyword value by the actual file path.

As soon as the modification is effective, the duws210.ear has to be rebuilt. See section Configure the duws210.ear.

7.5 JBoss Security Domain Definition

This section explains how to associate a *security domain* to Dollar Universe Web Services. This example uses the JBoss login module *UsersRoleLoginModule* which supports users and user roles loaded from Java properties files.

Edit the ***%JBOSS_INSTALL_DIR%\server\default\conf\login-config.xml***. The file login-config.xml defines the JBoss security domains for a server configuration.

Add the following XML element as a **policy** element child.

```
<application-policy name="duws">
  <authentication>
   <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
          flag="required">
      <module-option name="usersProperties">duws-
users.properties</module-option>
      <module-option name="rolesProperties">duws-
roles.properties</module-option>
   </login-module>
 </authentication>
</application-policy>
```

7.6 Deploy Dollar Universe Web Services in JBoss

Copy the modified duws210.*ear* to the **%JBOSS_INSTALL_DIR%\server\default\deploy** directory.

Run the ***%JBOSS_INSTALL_DIR%\bin\run.bat*** batch file to launch the JBoss application server. Should you wish to make the server reachable from a machine other than the localhost, use the `-b 0.0.0.0` argument.

The WSDL should be accessible at “//<host>:<port>/services/duws?WSDL”. The browser should display an authentication prompt.

8. Troubleshooting

The Dollar Universe web service has been successfully deployed, but all requests still fail. Several points can be checked to solve the problems:

- Deployment settings and application server configuration
- Dollar Universe Web Services configuration file

8.1 Increase the Trace Level

Several reasons may prevent Dollar Universe Web Services from running successfully. To find out what happens during the execution of the web service, activate the trace level (com.orsyp.duws.log.level = 4), undeploy and then redeploy Dollar Universe Web Services.

Trace messages will be written by the web service to the file specified by the property ***com.orsyp.duws.log.file***.

8.2 Web Service Deployment

To check whether Dollar Universe Web Services has been correctly deployed, retrieve the WSDL (see section Deploy the duws210.ear). Check the JEE application server log file for errors during the deployment phase.

8.3 User Authentication Refused by the Web Server

Check whether the authentication scheme is correctly set or not. If you cannot retrieve the WSDL file despite the fact that the web service is correctly deployed, security options may not be correctly configured.

Check if the user is known to the application server and if he is allowed to access Dollar Universe Web Services.

By default the user must be assigned the role **DollarUniverseUser** (as specified in the *web.xml* descriptor of the web service). If the *web.xml* descriptor refers to another role, it must be assigned to each user who is supposed to use Dollar Universe Web Services.

```
  <security-constraint>
```

```
    <web-resource-collection>
      <web-resource-name>Dollar Universe Web Services</web-resource-name>
      <url-pattern>/duws</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>DollarUniverseUser</role-name>
    </auth-constraint>
  </security-constraint>
```

8.4 Dollar Universe Web Services Configuration Issue

If WSDL is correctly retrieved, you may find a similar message in the application server log:

```
19:48:39,552 ERROR [STDERR] Cannot read config fileENTER_YOUR_PATH (The
system cannot find the file specified)
19:48:39,583 ERROR [SOAPFaultHelperJAXRPC] SOAP request exception
```

Check whether the Dollar Universe Web Services configuration file path specified in the ***web.xml*** descriptor is valid and accessible by the web service.

The parameter *configFile* is not properly set in the following example:

```
<context-param>
    <description>Configuration file</description>
    <param-name>configFile</param-name>
    <param-value>ENTER_YOUR_PATH</param-value>
</context-param>
```

Replace the value **`ENTER_YOUR_PATH`** with a valid file name.

8.5 Invalid License for Dollar Universe Web Services

If the configuration is correctly entered, the issue may be related to the license file:

- Check whether the license file path specified by the property *com.orsyp.duws.lic.file* is correct (the file must exist).
- Check also that it can be opened for reading by Dollar Universe Web Services.
- Finally check the validity of the license (code WSE, version is 2) for each Dollar Universe node accessed by the web service. If the file exists but no license WSE/2 is found for the Dollar Universe node, the web service client will get a ***DuwsException*** with the "**No license**" message.

8.6 Node Mapping

Check whether the node name to hostname mapping file specified by the property **com.orsyp.duws.nodes.file** is accessible by the web service and has a valid format.

8.7 Dollar Universe Service Declarations

Undeclared port numbers may lead to the following exception:

```
com.orsyp.duws.stubs.DuwsException: IO Connection problem - Cannot find
port [UNIV53/vmsdmduws1/X]
```

Check whether they have been correctly declared in the TCP/IP services mapping file or if the value ***com.orsyp.duws.services.file*** is incorrect.

8.8 User Authentication Refused by Dollar Universe

The connection from Dollar Universe Web Services to the Dollar Universe I/O Server will fail if proxies are activated and the user does not have any access rights on the node. In this case, the web service client will get the following exception:

```
com.orsyp.duws.stubs.DuwsException: No proxy defined for user - admin, in
environment [UNIV53/vmsdmduws1/X]
```

8.9 How To Suppress SocketTimeout Java Exception (with Axis 1.4)

Transferring large execution log files from the web service to the web service client may take several seconds or minutes. During the transfer with Axis 1.4, the Java exception

```
java.net.SocketTimeoutException: Read timed out
```

can be thrown by the client application. To avoid this error, it is recommended to add the following line in the client source code to increase the socket timeout value for the requests from this client.

```
service.settimeout(2400000);
```

Where `service` is the object of the stub service class generated from the WSDL. The timeout value is given in milliseconds. Recompile the client source code and run the client again.

9. Appendix

9.1 Status Codes

The status codes for launches, executions and events are listed in the table:

Code	Meaning
I	Aborted
R	Refused
O	Time overrun
W	Event wait
A	Pending
E	Running
D	Started
T	Completed
F	Completion in progress
L	Launch wait
S	Disabled

9.2 Task Type Codes

The codes and meaning for task types are listed in the following table:

Code for Task type	Meaning
'D'	Scheduled Task.
'P'	Provoked Task
'S'	Optional Task.

9.3 Date and Time Formats

Dates and times formats are:

- YYYY represents the year.
- MM represents the month [01;12].
- DD represents the day in the month [01;31].
- HH is for hours [00;23].
- MM is for minutes [00;59].
- SS is for seconds [00;59].

9.4 Origin and Type Codes for Uproc Variables

The variables source values are:

Code for origin	Meaning
'1'	Value comes from the Uproc definition.
'2'	Value comes from the Task definition.
'3'	Value comes from the Launch definition.

The variables type values are:

Code for type	Meaning
'T'	The variable is text.
'D'	The variable is a date.
'Q'	The variable is numeric.

10. Release Notes

10.1 Release Notes 2.0.01

10.1.1 Corrections

Problem	Description
pb098934 - bug15231	DUWS 2.0 was not compatible with Dollar Universe 5.6.
pb093501 - bug14407	The values returned by the operation getListLaunch for the launch status were incorrect. The expected values are 'L' (Launch wait), 'S' (Disabled), 'O' (Time overrun) and 'W' (Event wait).
pb098975 - bug15258	The code for the status "Time Overrun" returned by the getListExecution operation of the web service was incorrect. It should be "O" instead of "»".

10.2 Release Notes 2.1.00

10.2.1 New Operation getExecutionLogAsAttachment

A new operation 'getExecutionLogAsAttachment' is added to the web service to transfer the entire execution log as an SOAP attachment. The job log is returned as DataHandler (SOAP attachment type). This new operation allows the web service client to get huge logs. Since a new operation is added to the web service, the WSDL file is modified. The developer of the web service client has to generate a new set of stubs to use the new operation. The operation 'getExecutionLog' has not been modified. This new operation runs successfully with Axis 1.4.

11. Index

L

M

N

O

P

R

S

T

U

W

Email: support@orsyp.com
WEB site: http://www.orsyp.com
User Forum: http://www.orsypforum.com