WEB SERVICES DYNAMIC CLIENT GUIDE

USAGE RESTRICTED ACCORDING TO LICENSE AGREEMENT.

Version: 1.1

Last update: 30-Jul-2009.

Authors: Enrico Scagliotti, Giovanni Caire

TABLE OF CONTENTS

1 INTRODUCTION

This document describes the Web Services Dynamic Client (WSDC) add-on that provides support for dynamically invoking web services. Unlike the majority of web service oriented tools in facts WSDC allows invoking web services operations without the need of generating new classes. Furthermore it provides a simple and intuitive API to inspect services described by a WSDL (Web Service Description Language) retrieving for instance available operations as well as names and types of parameters.

Though WSDC belongs to the JADE software suite and requires the Jade library to be in the classpath, developers don't need to be familiar with JADE to use it.

This document only provides an overview of the main usages of the WSDC add-on. Refer to the Javadoc for a complete description of the API.

All bugs, issues, contributions and new feature requirements should be posted to the main JADE bug reporting system, and to the standard JADE mailing lists.

Version 1.0 of the WSDC add-on was developed by the JADE Team and is only guaranteed to work with JADE release 3.7 or later.

1.1 Compliance to standards

The WSDL service description language defines different binding styles (rpc and document) and uses (encoded and literal). WSDC supports the most commonly adopted combinations:

- **rpc/encoded** according to W3C standards (`http://www.w3c.org`)
- **document/literal** according to W3C standards (`http://www.w3c.org`)
- **document/literal wrapped** in compliance to the WS-I basic profile specification (`http://www.ws-i.org`).

All Date fields are encoded according to the ISO-8601 format.

1.2 Compatibility

WSDC 1.0 has been successfully tested with web services developed using AXIS 1.4 (`http://ws.apache.org/axis`) and 2.0 (`http://ws.apache.org/axis2`) and CXF 2.0 (`http://cxf.apache.org`) .

1.3 Requirements

The WSDC add-on requires Java JRE v5.0 (http://java.sun.com/javase/) or later, JADE v3.7 or later and the JADE misc add-on v2.1 or later. Furthermore the WSDC makes use of the following third party libraries **already included in the WSDC distribution**:

- Apache Axis v1.4 (http://ws.apache.org/axis/)
- Apache Commons (http://jakarta.apache.org/commons/)
- WSDL4J v1.6.2 (http://sourceforge.net/projects/wsdl4j)

1.4 Installation

The WSDC add-on does NOT require any installation. To use it from within a Java application it is sufficient to add all jar files included in the `lib` directory of the add-on **and the JADE jar files** to the application classpath.

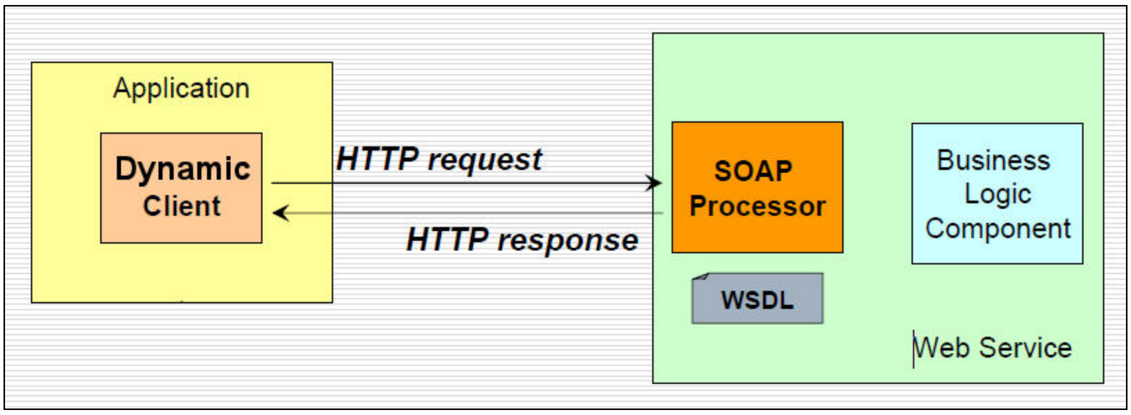
To recompile the add-on, as usual for JADE add-ons, the distribution zip file must be unzipped into the home directory of the JADE installation so that to obtain a directory structure as below.

```
<jade-home>/
    |--add-ons/
           |--dynamicClient/
```

After that the `lib` ANT target can be used to recompile the add-on and re-create its jar file.

2 DYNAMIC WEB SERVICE INVOCATION - BASICS

As mentioned WSDC allows invoking web services without the need of generating new classes such as stubs, and beans representing complex data types. Developers are simply required to create a `jade.webservice.dynamicClient.DynamicClient` object, initialize it specifying the URI of the WSDL describing the target service, and call the `invoke()` method as exemplified in the following sections.



2.1 Initialization

The following code snippet show how to create a `DynamicClient` object and initialize it. The WSDL URI can point to both a resource in the network or a file in the file system (e.g. `file:c:/foo/bar/myWsdl`).

```
DynamicClient dc = new DynamicClient();
dc.initClient(new URI("http://localhost/MathFunctionsPort?wsdl"));
```

It should be noticed that the `initClient()` method requires the indicated URI to be reachable and may take a while (a few seconds) as the WSDL must be retrieved and parsed and

WSDC must initialize all the internal data structures that will be used at invocation time.

In some cases, to properly deal with WSDL adopting particular forms or to force the WSDC to meet specific requirements, it is necessary/possible to modify the WSDC default configuration properties. This can be done by passing a properly configured `DynamicClientPropertis` object to the `DynamicClient` before invoking the `initClient()` method as exemplified below

```
DynamicClientProperties customProps = new DynamicClientProperties();
customProps.setNoWrap(true);
……
dc.setProperties(customProps);
dc.initClient(…..);
```

Parameters that can be configured are described in Appendix I.

2.2 Invocation

The following code snippet shows how to invoke a web service.

```
// Initialize input parameters
WSData input = new WSData();
input.setParameter("firstElement", 5);
input.setParameter("secondElement", 3);

// Invoke the sum operation
WSData output = dc.invoke("sum", input);

// Retrieve output parameters
float sum = output.getParameterFloat("sumReturn");
```

The `invoke()` method gets the name of the operation to be invoked and an instance of the `WSData` class that holds the input parameters if any. The result of the `invoke()` method is another `WSData` object that contains the output parameters.

In case the WSDL contains more that one service and/or port the `DynamicClient` class provides an overloaded version of the `invoke()` method that allows specifying them as shown below.

```
   WSData output = dc.invoke(“myService”, “myPort”, “sum”, null, -1,
input);
```

The above method also allows specifying a timeout for the invocation and an end-point different than that indicated in the WSDL.

2.3 Structured parameters

As shown in previous section, the `WSData` class allows dealing with un-structured parameters such as `String`, `Date, int`, `boolean` and so on directly. Values of complex parameters whose structure is defined by XSD schemas, must be handled using so called Abstract Descriptor classes. Abstract descriptor classes are part of the JADE core distribution and belong to the `jade.content.abs` package. It should be noticed however that WSDC users do not need to be familiar with JADE to use them. Furthermore, while the jade.content.abs package contains several Abstract Descriptor classes WSDC users only need to deal with three of them:

- `AbsConcept` - An Abstract Descriptor representing a complex data whose structure is described by an XSD schema
- `AbsAggregate` - An Abstract Descriptor representing a sequence of values
- `AbsObject` - The common base class for all Abstract Descriptors.

The following code snippet shows how to use Abstract Descriptors to manage values of structured parameters. More details on Abstract Descriptor classes can be found in the "Tutorial on Content Languages and Ontologies" (http://jade.tilab.com/doc/tutorials/CLOntoSupport.pdf) available on the JADE web site.

WSDL input data definition:

```
<xsd:complexType name="complex">
  <xsd:sequence>
    <xsd:element name="real" type="xsd:float"/>
    <xsd:element name="imaginary" type="xsd:float"/>
  </xsd:sequence>
</xsd:complexType>
……
<xsd:element name="sumComplex">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="firstComplexElement" type="impl:complex"/>
      <xsd:element name="secondComplexElement" type="impl:complex"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

DynamicClient code to set input parameters:

```
WSData input = new WSData();
AbsConcept first = new AbsConcept("complex");
```

```
    first.set("real", 4);
    first.set("imaginary", 5);
    input.setParameter("firstComplexElement", first);
    ....
    input.setParameter("secondComplexElement", second);
    ....
    WSData output = dc.invoke("sumComplex", input);
    AbsConcept sum = (AbsConcept) output.getParameter("sumComplexReturn");
    float real = sum.getFloat("real");
    float imaginary = sum.getFloat("imaginary");
```

3 DYNAMIC WEB SERVICE INVOCATION - ADVANCED

3.1 Headers

Besides parameters, a web service operation may include “headers”. Headers are basically additional parameters that are carried inside the header of a SOAP request/response instead of in the body. In general headers are used to specify additional information not strictly related to the semantics of an operation such a as the credentials (username and password) required to invoke it.

The WSData class allows managing parameters and headers homogeneously: while the
`void setParameter(<parameter-name>, < parameter-value>)`
`AbsObject getParameter(<parameter-name>)`
`String getParameterString(<parameter-name>),`
`int getParameterInteger(<parameter-name>),`
`boolean getParameterBoolean(<parameter-name>)`
...
methods are available to manage parameters, the
`void setHeader(<header-name>, <header-value>)`
`AbsObject getHeader(<header-name>)`
`String getHeaderString(<header-name>)`
`integer getHeaderInteger(<header-name>)`
`boolean getHeaderBoolean(<header-name>)`
...
methods are available to manage headers.

3.2 Proxy

In many cases both the access to a WSDL (at DynamicClient initialization time) and the actual web service invocation require passing through an HTTP Proxy. The DynamicClient class provides the following methods to set proxy information.

- `setProxyHost(<host>)`: Set the proxy host (e.g. 163.162.10.12)

- `setProxyPort(<port>)`: Set the proxy port (e.g. 8080)

- `setNonProxyHosts(<listOfAddresses>)`: Set a list of addresses (possibly including '*') that will be accessed without using the proxy. The separator is the '|' character

- `setProxyAuthentication(<user>, <password>)`: Set the credentials (if any) required to access the proxy

The following code snipped provides an example.

```
dc.setProxyHost("10.12.175.14");
dc.setProxyPort("8080");

dc.setNonProxyHosts("163.163.*|*.telecomitalia.it");
dc.setAuthentication("myUser", "myPwd");
dc.initClient(new URI("http://myWSDL"));
```

3.3 Security

Certain web services require HTTP Basic Authentication. The DynamicClient class provides the following methods to set HTTP related information.

- `setDefaultHttpUsername()`: Specifies the http username used in all requests.

- `setDefaultHttpPassword()`: Specifies the http password used in all requests.

The following code snipped provides an example.

```
dc.setDefaultHttpUsername("MyHttpUsername");
dc.setDefaultHttpPassword("MyHttpPassword");
```

If the credential of HTTP Basic Authentication are different in all requests is possible specify them in `invoke(…)` method with `SecurityProperties` object.

Other web services require WS-Security Username Token. The DynamicClient class provides the following methods to set WSS related information.

- `setDefaultWSSUsername()`: Specifies the wss username used in all requests.

- `setDefaultWSSPassword()`: Specifies the wss password used in all requests.

- `setDefaultWSSPasswordType()`: Specifies the wss password type used in all requests (`TEXT` or `DIGEST`, see `SecurityProperties` object).

The following code snipped provides an example.

```
dc.setDefaultWSSUsername(“MyWSSUsername”);
dc.setDefaultWSSPassword(“MyWSSPassword”);
dc.setDefaultWSSPasswordType(SecurityProperties.PW_TEXT);
```

If the credential of WS-Security Username Token are different in all requests is possible specify them in `invoke(…)` method with `SecurityProperties` object.

Other web services require SSL connections with or without certificates. The DynamicClient class provides the following methods to set SSL related information.

- `enableCertificateChecking()`: Enables the certificates checking mechanism. When this mechanism is enabled (the default situation) a trust store holding certificates of trusted remote servers must be indicated (see the `setTrustStore()` method).

- `disableCertificateChecking()`: Disables the certificate checking mechanism.

- `setTrustStore(<file.keystore>)`: Specifies the keystore holding certificates of trusted remote servers

- `setTrustStorePassword(<password>)`: Specifies the password used to protect the keystore of trusted certificates

The following code snipped provides an example.

```
dc.setTrustStore(“C:/myFolder/cert.keystore”);
dc.setTrustStorePassword(“myPassword”);
dc.initClient(new URI("http://myWSDL"));
```

3.4 Caching

Considering that the initialization of a DynamicClient (`initClient()` method) is a long operation that may take some seconds, a good approach is to create a single `DynamicClient` instance for each WSDL and reuse it whenever an operation of a service described in that WSDL must be invoked (note that the `invoke()` methods of the `DynamicClient` class are thread safe and therefore can be called by two or more threads in parallel). In order to facilitate this practice the WSDC provides a class called `DynamicClientCache` that manages all issues related to creation, initialization and caching of `DynamicClient` objects in a thread safe mode. The `DynamicClientCache` class follows the singleton pattern and therefore the first step when using it is to retrieve the singleton `DynamicClientCache` instance by means of the `getInstance()` method.

The following code snippet shows how to use the DynamicClientCache class.

```
DynamicClientCache dcc = DynamicClientCache.getInstance();
DynamicClient client = dcc.get(new URI("http://myWSDL"));
……
WSData output = client.invoke(“sum”, input);
```

The `get()` method of the `DynamicClientCache` class first checks if a `DynamicClient` object was already created to access the given WSDL and returns it in that case. Only if no `DynamicClient` object is already available a new one is created and initialized.

4 READING WSDL INFORMATIONS

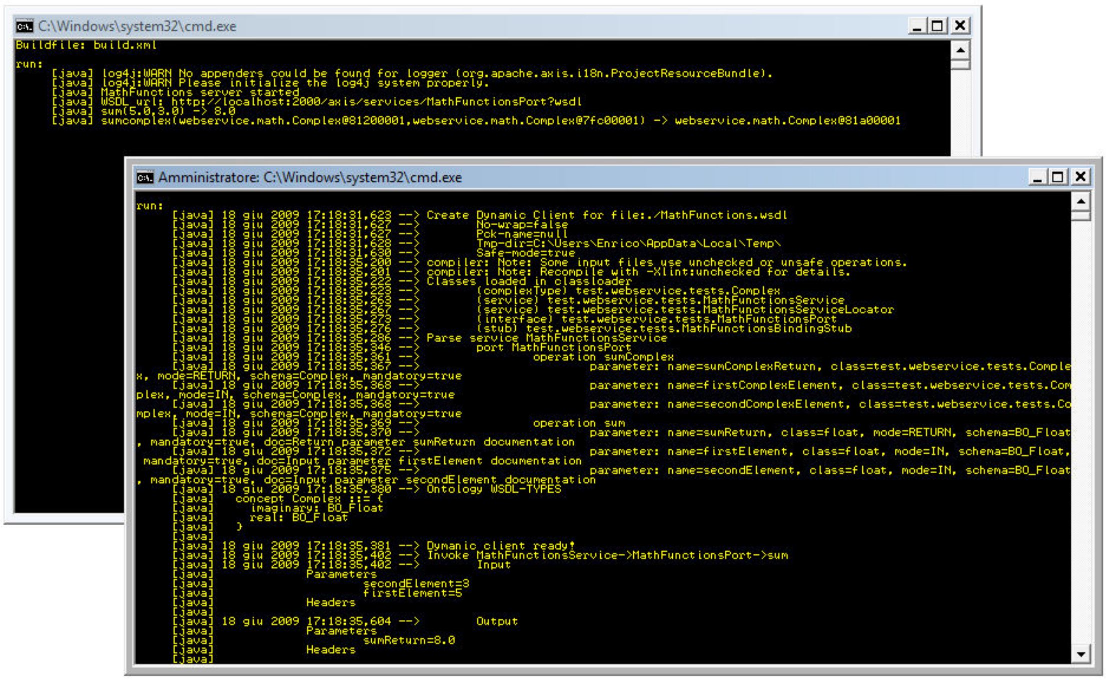
Besides invoking web service operations, the WSDC add-on also provides a simple and intuitive API to read WSDL information such as which operations are there in a service and which parameters they have. The following code snippet shows an example where the names, types and descriptions of the input parameters of the sumComplex operation of the mathService service exposed at port https are retrieved.

```
DynamicClient dc = new DynamicClient();
dc.initClient(new URI("http://localhost/MathFunctionsPort?wsdl"));
ServiceInfo si = dc.getService("mathService");
PortInfo pi = si.getPort("https");
OperationInfo oi = pi.getOperation("sumComplex");
Set<String> parNames = oi.getInputParameterNames();
for (String name : parNames) {
  ParameterInfo par = oi.getParameter(name);
  TermSchema schema = par.getSchema();
  String description = par.getDocumentation();
  System.out.println(name+": type = "+schema.getTypeName()+
                     ", description = "+description);
}
```

It should be noticed that parameter types are described using JADE ontological schemas. Refer to the “Tutorial on Content Languages and Ontologies” (http://jade.tilab.com/doc/tutorials/CLOntoSupport.pdf) available on the JADE web site for details about ontological schemas.

5 APPENDIX I. DESCRIPTION OF THE EXAMPLES

In the **`examples/client`** directory of the WSDC add-on a simple example is available showing how to use the `DynamicClient` class to invoke web services. A “micro-web server” (included in the **`examples/server`** directory) is also provided exposing two mathematical operations that are invoked by the example. In order to try the example it is sufficient to start the server by means of the `startMathServer` script and then to launch the client by means of the `startMathClient` script. Both scripts are included in the **`examples`** directory and are available for Linux (`.sh`) and Windows (`.bat`).


C:\Windows\system32\cmd.exe
Amministratore: C:\Windows\system32\cmd.exe

```
Buildfile: build.xml

run:
     [java] log4j:WARN No appenders could be found for logger (org.apache.axis.i18n.ProjectResourceBundle).
     [java] log4j:WARN Please initialize the log4j system properly.
     [java] MathFunctions server started
     [java] WSDL_url: http://localhost:2000/axis/services/MathFunctionsPort?wsdl
     [java] sum(5.0,3.0) -> 8.0
     [java] sumcomplex(webservice.math.Complex@81200001,webservice.math.Complex@7fc00001) -> webservice.math.Complex@81a00001
```

```
run:
     [java] 18 giu 2009 17:18:31,623 --> Create Dynamic Client for file:./MathFunctions.wsdl
     [java] 18 giu 2009 17:18:31,627 -->         No-wrap=false
     [java] 18 giu 2009 17:18:31,627 -->         Pck-name=null
     [java] 18 giu 2009 17:18:31,628 -->         Tmp-dir=C:\Users\Enrico\AppData\Local\Temp\
     [java] 18 giu 2009 17:18:31,630 -->         Safe-mode=true
     [java] 18 giu 2009 17:18:35,200 --> compiler: Note: Some input files use unchecked or unsafe operations.
     [java] 18 giu 2009 17:18:35,201 --> compiler: Note: Recompile with -Xlint:unchecked for details.
     [java] 18 giu 2009 17:18:35,222 --> Classes loaded in classloader
     [java] 18 giu 2009 17:18:35,223 -->         (complexType) test.webservice.tests.Complex
     [java] 18 giu 2009 17:18:35,263 -->         (service) test.webservice.tests.MathFunctionsService
     [java] 18 giu 2009 17:18:35,267 -->         (service) test.webservice.tests.MathFunctionsServiceLocator
     [java] 18 giu 2009 17:18:35,273 -->         (interface) test.webservice.tests.MathFunctionsPort
     [java] 18 giu 2009 17:18:35,276 -->         (stub) test.webservice.tests.MathFunctionsBindingStub
     [java] 18 giu 2009 17:18:35,286 --> Parse service MathFunctionsService
     [java] 18 giu 2009 17:18:35,346 -->         port MathFunctionsPort
     [java] 18 giu 2009 17:18:35,361 -->                 operation sumComplex
     [java] 18 giu 2009 17:18:35,367 -->                         parameter: name=sumComplexReturn, class=test.webservice.tests.Comple
x, mode=RETURN, schema=Complex, mandatory=true
     [java] 18 giu 2009 17:18:35,368 -->                         parameter: name=firstComplexElement, class=test.webservice.tests.Com
plex, mode=IN, schema=Complex, mandatory=true
     [java] 18 giu 2009 17:18:35,368 -->                         parameter: name=secondComplexElement, class=test.webservice.tests.Co
mplex, mode=IN, schema=Complex, mandatory=true
     [java] 18 giu 2009 17:18:35,369 -->                 operation sum
     [java] 18 giu 2009 17:18:35,370 -->                         parameter: name=sumReturn, class=float, mode=RETURN, schema=BO_Float
, mandatory=true, doc=Return parameter sumReturn documentation
     [java] 18 giu 2009 17:18:35,372 -->                         parameter: name=firstElement, class=float, mode=IN, schema=BO_Float,
 mandatory=true, doc=Input parameter firstElement documentation
     [java] 18 giu 2009 17:18:35,375 -->                         parameter: name=secondElement, class=float, mode=IN, schema=BO_Float
, mandatory=true, doc=Input parameter secondElement documentation
     [java] 18 giu 2009 17:18:35,380 --> Ontology WSDL-TYPES
     [java]   concept Complex ::= {
     [java]     imaginary: BO_Float
     [java]     real: BO_Float
     [java]   }
     [java]
     [java] 18 giu 2009 17:18:35,381 --> Dymanic client ready!
     [java] 18 giu 2009 17:18:35,402 --> Invoke MathFunctionsService->MathFunctionsPort->sum
     [java] 18 giu 2009 17:18:35,402 -->         Input
     [java]                 Parameters
     [java]                         secondElement=3
     [java]                         firstElement=5
     [java]                 Headers
     [java]
     [java] 18 giu 2009 17:18:35,604 -->         Output
     [java]                 Parameters
     [java]                         sumReturn=8.0
     [java]                 Headers
     [java]
```

6 APPENDIX II. WSDC CONFIGURATION PROPERTIES

The following table summarizes the configuration properties that can be passed to a `DynamicClient` object before initialization.

Parameter	Type	Description	Default value
noWrap	boolean	Disables the automatic de-wrapping of parameters.	false
safeMode	boolean	In case a Document Literal WSDL does not appear to comply to the Wrapped convention, the noWrap mode is forced	true
packageName	string	Specify the java package for classes generated under the hood by WSDC and later used in the invocation phase	-
tmpDir	string	WSDC requires a temporary directory with	System-property

		R/W right sto work properly	java.io.tmpdir
classPath	StringBuilder	Specify an ad-hoc classpath for the compilation of the classes generated under the hood by WSDC and later used in the invocation phase	-

7 APPENDIX III. CURRENT LIMITATIONS

Version 1.0 of the WSDC add-on has the following known limitations:

- SOAP attachments are not supported
- XSD complex type restrictions are not supported
- Not all XSD/SOAP types are supported