

WS-CDL+ Execution Engine User Guide:

Get Started

1 Introduction

The promise of reusability in software systems has become a common theme in commercial application development in recent years. Today, we see it with Web services and the generic concept of Service-Oriented Computing, namely SOC. Web services are autonomous software systems identified by URIs, which can be advertised, located, and accessed through messages encoded according to XML-based standards, and transmitted using Internet protocols. They are designed to support interoperable machine-to-machine interaction over a network, and are now becoming the main building blocks of SOC. SOC is the computing paradigm that utilizes services as fundamental elements for developing applications/solutions. It stands on a higher level, and creates software systems by composing the elementary services, mainly in form of Web services, with service composition technology such as WSFL , XLANG, or BPEL4WS. Because Web services are based on these open and widely-accepted protocols, such as HTTP and SOAP, and provide a uniform and ubiquitous information distributor for a wide range of computing devices and software platforms, they, as well as SOC, constitute the next major step in distributed computing.

The composition technologies, which define an executable process to be enacted by a central orchestration engine, and are a single participant's view of the world, such as WSFL or BPEL4WS. They address one aspect of the SOC, the collaboration technologies, which describe an interaction protocol to be enacted by the peers without any central entity governing the collaboration, and providing a global view of the peers, which address the other aspects of the SOC. WS-CDL is one of the most promising collaboration technologies in the service computing area.

One of the biggest problems going with WS-CDL is that it lacks a real or even prototype execution engine for it to run. Nevertheless, further researches of WS-CDL and its applications do need such an execution engine to experiment on their research conclusion, or test their new properties, or create a B2B message collaboration system. Motivates are then initiated to implement such an engine. However, when building and testing the execution engine, we find it necessary to add extended features to the current WS-CDL specification, so as to enhance the functionalities of WS-CDL itself, and also make it easy and concise for developers to express their collaboration processes. These extensions finally leads to the formation of WS-CDL+. Based upon WS-CDL, WS-CDL+ mainly provides six new functionalities such as user-defined functions, timers, implicit finalization mechanism and so on, and imposes some restrictions to what WS-CDL has not explicitly stated, which increases the co-operability and reduces the arbitrariness from different WS-CDL/WS-CDL+ implementers as well.

Our work aims to advance the current state of the Web service collaboration technology by addressing the following two issues: **a) A prototype implementation of a WS-CDL+ execution engine.** The study of WS-CDL calls for an execution engine; so that we can experiment on the

properties we are working on, and develop applications using WS-CDL. However, today's research mainly focuses on the description, translation and semantic of WS-CDL, leaving no execution engines being built. Implementing a WS-CDL execution engine is quite helpful to the research of WS-CDL itself, as well as its communication and coordination protocols/models. The availability of a WS-CDL execution engine is also crucial to bring WS-CDL into application field, enabling a WS-CDL document to run on servers. Moreover, with the prototype system, researchers and developers are able to simulate their choreography process and get the results of choreography in a single computer without having to interact with the real WS-CDL/WS-CDL+ participants, which makes testing and debugging WS-CDL/WS-CDL+ documents easy. And **b) the extension of the WS-CDL specification into WS-CDL+**. It is no doubt that the idea underlying WS-CDL is fascinating; however, when it comes to the application field, its expressiveness and usability become questionable. To enhance it in this way, we introduce six extended functionalities into the WS-CDL+ execution engine, including the user-defined functions, the interaction model extensions, the implicit finalization mechanism, and etc.

The remainder of this guide is organized as follows: Section 2 introduces how to install the execution engine and verify the installation by running the bundled samples. Section 3 addresses the way to deploy Web services, WSDL files and WS-CDL+ files. Section 4 presents the way to run in the simulation mode. Section 5 provides some tips which are important to make a WS-CDL document run in our WS-CDL+ execution engine.

2 Installing the WS-CDL+ Execution Engine

2.1 Running environments

- ✓ Windows/Unix/Linux
 - Shell scripts are needed if running under Unix/Linux
- ✓ Java 1.5.0
 - Java 6.0 is not recommended

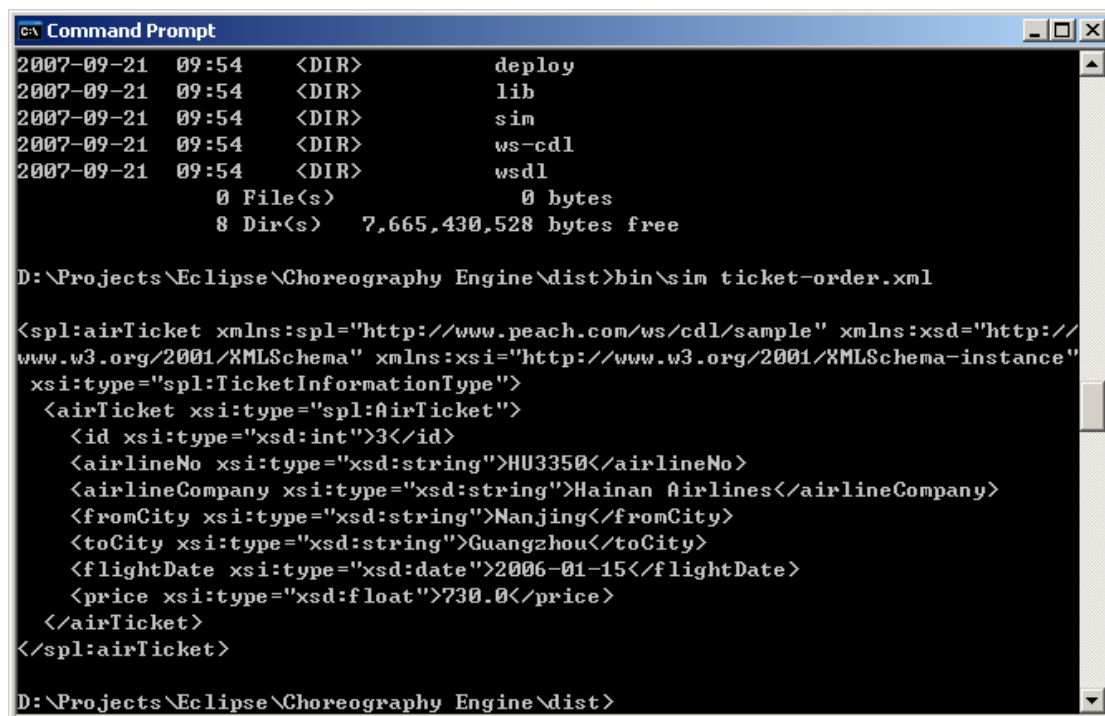
2.2 Installation steps

- If you don't have Java on your system, download it from <http://java.sun.com> website and install it into a proper directory, for example, C:\Java\jdk1.5.0_12
- Download the software package of the WS-CDL+ execution engine and extract it into a certain directory, for example, C:\Java\wscdl-ee-0.1
- Setup two environment variables, JAVA_HOME and WSCDL_HOME, to the proper values based upon your own system settings, here is the example:
 - JAVA_HOME: C:\Java\jdk1.5.0_12
 - WSCDL_HOME: C:\Java\wscdl-ee-0.1

2.3 Running bundled samples in simulation mode

You can make sure that your installation is successful by running the bundled samples within our software package. There are three WS-CDL/WS-CDL+ samples bundled, one of which is written by the researchers of the execution engine, about the service collaboration process among three participants, the travel agency, the ticket seller and the bank. The complete document of this sample is located at `$WSCDL_HOME/ws-cdl/sample.xml`. It is a typical WS-CDL+ document, using most of the extended functionalities provided by WS-CDL+. However, the other is more WS-CDL, modified according to the sample from the official WS-CDL website, located at `$WSCDL_HOME/ws-cdl/consumer-retailer.xml`.

The WS-CDL+ execution engine is intended to support running in both simulation mode and real mode. However, in this prototype system, only simulation mode is provided. To run the first sample in simulation mode, entering `$WSCDL_HOME` and type `bin\sim ticket-order.xml`. You'll see the following output on your console if no exception has ever occurred. And if the following is shown, it means that your installation is successful.



```
Command Prompt
2007-09-21 09:54 <DIR>      deploy
2007-09-21 09:54 <DIR>      lib
2007-09-21 09:54 <DIR>      sim
2007-09-21 09:54 <DIR>      ws-cdl
2007-09-21 09:54 <DIR>      wsd1
           0 File(s)          0 bytes
           8 Dir(s)  7,665,430,528 bytes free

D:\Projects\Eclipse\Choreography Engine\dist>bin\sim ticket-order.xml

<spl:airTicket xmlns:spl="http://www.peach.com/ws/cdl/sample" xmlns:xsd="http://
www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="spl:TicketInformationType">
  <airTicket xsi:type="spl:AirTicket">
    <id xsi:type="xsd:int">3</id>
    <airlineNo xsi:type="xsd:string">HU3350</airlineNo>
    <airlineCompany xsi:type="xsd:string">Hainan Airlines</airlineCompany>
    <fromCity xsi:type="xsd:string">Nanjing</fromCity>
    <toCity xsi:type="xsd:string">Guangzhou</toCity>
    <flightDate xsi:type="xsd:date">2006-01-15</flightDate>
    <price xsi:type="xsd:float">730.0</price>
  </airTicket>
</spl:airTicket>

D:\Projects\Eclipse\Choreography Engine\dist>
```

3 Deploying Web services and WS-CDL+ files

3.1 Deploying Web services

The simulation mode execution requires the invoked Web services be locally accessible, which means that you have to locally deploy a Java class as a Web service. To do so, follow the steps listed below.

- Annotate the Web services with port type names, input/output parameters, and fault names if

there exist. Details on how to annotate these Java classes into proper Web services can be found in the source code of sample Web services, and will not be covered in this guide.

- If the input/output parameters of the deployed Web service are more than the primitive data types of the Java language, annotate them. Details on these annotation will not be covered either, and you can also them on the sample Web services.
- Compile these Java classes and package them into `$WSCDL_HOME/lib/web-services.jar`. Be sure that you are using the correct package name and placing it into a correct directory.
- Write a well-formed WSDL file for the Web service and save it as `$WSCDL_HOME/wsd/filename.xml`.
- Write a deployment description file, in which Java classes that will be deployed as message types and port types are listed, and save it as `$WSCDL_HOME/deploy/web-services.xml`

3.2 Deploying WS-CDL+ files

- Prepare for the WS-CDL+ file you are going to deploy and save it into `$WSCDL_HOME/ws-cdl/filename.xml`.
- Write the WS-CDL+ deployment description file(s) and save it as `$WSCDL_HOME/deploy/*-cdl.xml`. Examples on how to write WS-CDL+ deployment description files can be found at `$WSCDL_HOME/deploy`. This file is composed of four parts, participant, endpoint, ws-cdl and wsdl. The meaning of these four elements are:
 - The participant element determines which participant type the WS-CDL+ container will take.
 - The base attribute of endpoint specifies accessing URL through which the channel types of a certain participant type can be referenced. For simulation mode, you can
 - The ws-cdl element specifies the WS-CDL+ document that we will use.
 - The wsdl elements present the WSDL documents that will be referenced by the specified WS-CDL+ document. You can write multiple lines of the wsdl elements for multiple WSDL documents.

4 Run WS-CDL+ execution engine in simulation mode

To execute a WS-CDL+ document in simulation mode, we need a simulation description file located at `$WSCDL_HOME/sim`. Several examples of the simulation description file are provided altogether with the software package. Each of these files can be used by issuing the command “run *filename*“. Let’s now look at `$WSCDL_HOME/sim/ticket-order.xml` and see how to write a simulation description file.

The container elements are used to specify the WS-CDL+ deployment description files to the execution engine and each file corresponds to a separate WS-CDL+ instance that will be run in a separate WS-CDL+ container, between which message exchanges occur. The initiator attribute of the container element specifies whether or not this instance will be initialized by default when the simulated execution begins. Note that there is one and only one WS-CDL+ container could have its initiator attribute valued true. And for that true-valued instance, the initial values for some of its variables can be specified within the instance element.

Since the simulation description file is easy to write, we are not going to explain it any further. We think it won't be hard for users to write their own simulation description files by referencing the simulation description file samples bundled within our software package.