oBay Installation Guide

This document describes how to install and run the sample oBay application as described in the Oracle SOA Suite Developer's Guide. It assumes that you have already installed the Oracle SOA Suite as documented in the installation guide that accompanies this book.

Installing oBay

The sample oBay application consists of the following components that need to be installed and configured in the following order:

- oBay Database Schema
- EJB Web Services
- OSB Services
- XML Schema Library on SOA Suite
- BPEL Processes, Fault Policies, Rules on SOA Suite

In preparation for installing the oBay application, unzip the `obay.zip` file into its own directory.

Preparing the oBay database schemas

The first step in installing oBay is to install the database schemas. This is done by running the running the `installSchemas.bat` command found in the `obay\sql` directory where you unpacked the ZIP file.

Before running the `installSchemas` script, it is necessary to set the following environment variable:

- ORACLE_HOME: The location of the database files. This can be set to be the `app\oracle\product\10.2.0\server` directory underneath the directory where you installed Oracle XE.

To run the script, open up a **Command Prompt** and then navigate to the `obay\sql` directory. Then enter the following command:

```
installSchema system/<password>
```

Where you need to substitute `<password>` with the system password you set during the installation of XE.

When completed the script, you will have created the user `obay` (password `obay`) and the various database objects required to run the application.

Installing the EJB Web Services

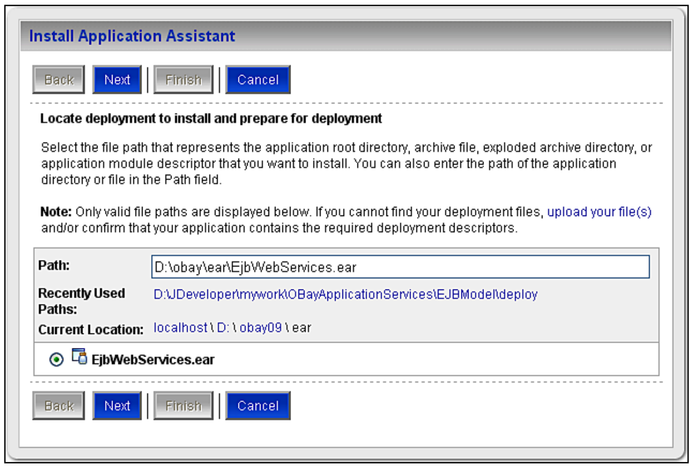
oBay uses a number of web service enabled EJB's for managing the data held within the `obay` database. These EJB's are packaged up in the file `obay\ear\EjbWebServices.ear`.

To install the file, first ensure that the Oracle Service Bus is started. Then within a browser, go to the following URL:

```
http://localhost:7001/console
```

Sign in as a user with administrator privileges (For example: **weblogic**). Then carry out the following steps:

- In the Change Center of the **Administration Console**, click **Lock and Edit** (see Use the Change Center).
- In the left pane of the Console, select **Deployments**.
- In the right pane, click **Install**. This will display the **Install Application Assistant** shown as follows:



Locate deployment to install and prepare for deployment

Select the file path that represents the application root directory, archive file, exploded archive directory, or application module descriptor that you want to install. You can also enter the path of the application directory or file in the Path field.

Note: Only valid file paths are displayed below. If you cannot find your deployment files, upload your file(s) and/or confirm that your application contains the required deployment descriptors.

Path: D:\obay\ear\EjbWebServices.ear

Recently Used Paths: D:\JDeveloper\mywork\OBayApplicationServices\EJBModel\deploy

Current Location: localhost \ D: \ obay09 \ ear

● **EjbWebServices.ear**

- In the **Path** field, enter the full path to the directory containing `EjbWebServices.ear` file and hit *Enter*. The screen will be updated to list all the EAR files contained in this directory.
- Locate the `EjbWebServices.ear` file and click on the radio button to select it. Then click **Next**.
- You will then be prompted to choose a target style, keep the default (**Install this deployment as an application**) and click **Next**.
- On the remaining screens keep the default configurations and just click **Next**. When you reach the final screen just click **Finish** and your EAR file will be deployed.
- Remember to click **Activate Changes** in the **Change Center**.

Installing the OSB services

The OSB services are packaged up in the file `obay\osb\sbconfig.jar`. To install the file, first ensure that the Oracle Service Bus is started. Then within a browser, go to the following URL:

```
http://localhost:7001/sbconsole
```

Sign in as a user with administrator privileges (such as **weblogic**). Then carry out the following steps:

- Click on **Create** with the Change Center to start an edit session.
- Click on the **System Administration** section in the navigation bar and then select **Import Resources**.
- Click on **Browse** and then locate and select the `sbconfig.jar` file. Then click **Next**.
- Ensure that both **oBay** and **xmllib** are selected and then click **Import**.
- The OSB will then precede with the import. Once completed it will confirm that the items have been successfully imported.
- Finally, click on **Activate** within the **Change Center**.

Installing the XML Schemas

The BPEL processes make use of a number of shared XML Schemas that need to be installed in `xmllib`.

To do this, copy the content of `obay\xmllib` to the directory:

```
<SOA_HOME>\bpel\system\xmllib
```

Installing the BPEL Processes

The directory `obay\jdeveloper` contains the following applications:

- **Auction**
- **ExternalServices**
- **JobScheduler**
- **ListItem**
- **OrderFulfillment**
- **UserAccount**

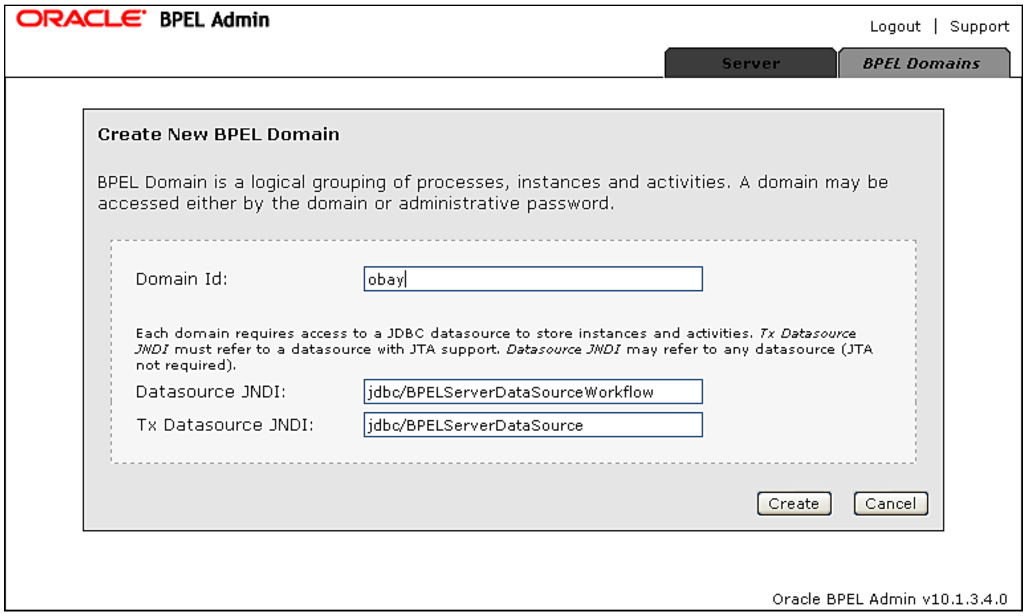
Each of these contains a number of BPEL Processes which must be deployed to BPEL PM in the `obay` domain.

Creating the oBay domain

To create the `obay` domain you need to log into the BPEL Admin console. In your browser enter the following url:

```
http://localhost/BPELAdmin
```

Log in as **bpeladmin**; click on the tab **BPEL Domains** in the top right hand corner, next click on **Create New BPEL Domain**. This will bring up the screen **Create New BPEL Domain** as shown:



Create New BPEL Domain

BPEL Domain is a logical grouping of processes, instances and activities. A domain may be accessed either by the domain or administrative password.

Domain Id: obay

Each domain requires access to a JDBC datasource to store instances and activities. *Tx Datasource JNDI* must refer to a datasource with JTA support. *Datasource JNDI* may refer to any datasource (JTA not required).

Datasource JNDI: jdbc/BPELServerDataSourceWorkflow

Tx Datasource JNDI: jdbc/BPELServerDataSource

Enter **obay** as the **Domain Id**, keep the default values specified for the data sources and click **Create** and then **OK**. Once you have received confirmation that the domain has been successfully created, you are ready to install the BPEL processes.

Installing the BPEL fault policies

The BPEL processes make use of a number of fault policies that need to be installed in `obay` domain.

To do this, copy the contents of `obay\config` to the directory:

```
<SOA_HOME>\bpel\domains\obay\config
```

You will need to restart BPEL PM in order for these policies to come into effect.

Modify the sample user community

The SOA Suite comes with a sample user community is defined in the file:

```
<SOA_HOME> \j2ee\oc4j_soa\config\system-jazn-data.xml
```

Replace this with the file `obay\users\system-jazn-data.xml`. This adds the additional group `oBayAdministrator` containing the users `cdoyle`, `mtwain`, and `jcooper`.

Configuring the schedule process

We use the file adapter to initiate the `schedule` process as covered in Chapter 14, which in turn will execute the `AccountBilling` process nightly (at 9pm).

Before deploying the `schedule` process, we need to create the directory structure and schedule file that the file adapter will poll.

Create a base directory, such as **D:\Soa1013** and copy the content of `obay\schedule` into this directory. This will create two sub-directories **config** and **execute**; and the file **schedule.xml** in **config**.

Configure bpel.xml for schedule process

The file `oBay\JDeveloper\JobScheduler\Schedule\bpel\bpel.xml` contains a section which configures the activation agent for the file adapter shown as follows:

```
<activationAgents>
<activationAgent
  className="oracle.tip.adapter.fw.agent.jca.JCAActivationAgent"
  partnerLink="ScheduleFileAdapter">
    <property name="schedulerExecuteDir"
```

```
        type="LogicalDirectory">D:\Soa1013\schedule\execute</property>
      <property name="portType">ReadScheduleFile_ptt</property>
    </activationAgent>
  </activationAgents>
```

The property `LogicalDirectory` (highlighted) needs to be set to hold the full path name of the `execute` directory, as this configures the directory that the file adapter will poll. Use a text editor to modify this file as appropriate.

Configure schedule.xml file

The file `schedule.xml` contained in the directory `config` also needs to be configured to point at the appropriate directories; this file contains the following two elements:

```
<schedulerExecuteDir>D:\Soa1013\schedule\execute</schedulerExecuteDir>
<schedulerConfigDir>D:\Soa1013\schedule\config</schedulerConfigDir>
```

These need to be modified to specify the full path name to the `execute` and `config` directories that you have created.

Once done, place a **copy** of the `schedule.xml` file in the `execute` directory, this will trigger the `schedule` process to run once it has been deployed.

Deploy BPEL Processes

Within JDeveloper, open up each of the applications in the directory `obay\jdeveloper` and deploy each BPEL Process in the following order:

- ExternalService
    - Any order
- UserAccount
    - BillUser
    - ValidateEmailAddress
    - UserRegistration
- OrderFulfilment
    - Any order
- Auction
    - Auction
    - AuctionProxy
    - AuctionItem

- ListItem
    - CheckSuspectItem
    - ListItem
    - SubmitBid
- JobScheduler
    - JobServices
    - AccountBilling
    - Schedule

This completes the installation of the oBay application.

Running oBay

The following External Virtual Services are provided as part of the oBay application.

- **UserAccount**: Used to register for an account with oBay.
- **SellItem**: Provides all the required operation to list and item for sale, and then manage the order fulfilment process from the perspective of a seller.
- **BuyItem**: Provides all the required operations to the find items up for auction and place a bid on them. For items won, provides the operations to manage the order fulfilment process from the perspective of a buyer.
- **Reference Date**: Provides all the required operations to look up reference date (i.e. listing categories).

The simplest way to execute these services, as well as check the input they expect and the response they return is via the Service Bus Test Console.

[While these services are designed to provide all the operations required to build a web based interface for oBay. This example application doesn't include a UI as we believe this to be a separate exercise beyond the scope of the book.

However, at some point we do plan on writing a UI for oBay which we will make available on our blogs.]

UserAccount

The WSDL for this service is located at the url:

```
http://localhost:7001/oBay/EVS/UserAccount?wsdl
```

UserRegistration

To register a user, you first need to invoke the `submitUserRegistration` operation, with a payload similar to the following:

```
<user:submitUserRegistration xmlns:usr="http://schemas.packtpub.
com/obay/usr" xmlns:cmn="http://schemas.packtpub.com/obay/cmn"
xmlns:user="http://xmlns.packtpub.com/obay/evs/UserAccount">
  <usr:user>
    <usr:userId>jcooper</usr:userId>
    <cmn:name>
      <cmn:title>Mr</cmn:title>
      <cmn:forename>James</cmn:forename>
      <cmn:surname>Cooper</cmn:surname>
    </cmn:name>
    <cmn:dob>1978-09-29</cmn:dob>
    <cmn:address>
      <cmn:addressLine1>21 Pine Way</cmn:addressLine1>
      <cmn:addressLine2></cmn:addressLine2>
      <cmn:city>Melbourne</cmn:city>
      <cmn:state>VIC</cmn:state>
      <cmn:zip>3000</cmn:zip>
      <cmn:country>Australia</cmn:country>
    </cmn:address>
    <cmn:emailAddress>jcooper@localhost</cmn:emailAddress>
    <cmn:creditCard>
      <cmn:cardType>MasterCard</cmn:cardType>
      <cmn:cardHolderName>James Cooper</cmn:cardHolderName>
      <cmn:cardNumber>2345234523452345</cmn:cardNumber>
      <cmn:expiryMonth>12</cmn:expiryMonth>
      <cmn:expiryYear>2010</cmn:expiryYear>
      <cmn:securityNo>123</cmn:securityNo>
    </cmn:creditCard>
  </usr:user>
  <usr:password>welcome1</usr:password>
</user:submitUserRegistration>
```

The proxy service will route this request to the `UserRegistration` BPEL process, which will perform validation against the content (as described in Chapter 9). Causing the service to return a fault with details of the error should it fail validation.

For credit card validation the card number **must** contain a '1' and a '9' in the number, otherwise it will fail validation.

ConfirmEmailAddress

Assuming `submitUserRegistration` is invoked successfully, it will send an email to the address specified. This will contain a token value that is required to complete the user registration process.

Assuming you haven't configured the SOA Suite to use a particular email provider, the value of this token is the **instance id** of the `UserRegistration` BPEL process, which you can obtain from the BPEL Console.

To complete the User Registration process you need to invoke the `confirmEmailAddress` operation, with a payload similar to the following:

```
<user:confirmEmailAddress xmlns:cmn="http://schemas.packtpub.com/obay/
cmn" xmlns:user="http://xmlns.packtpub.com/obay/evs/UserAccount">
  <cmn:emailAddress>jcooper@localhost</cmn:emailAddress>
  <cmn:emailAddressToken>840048</cmn:emailAddressToken>
</user:confirmEmailAddress>
```

Where the value for `emailAddressToken` should contain the instance Id of the corresponding `UserRegistration` process.

Assuming this completes successfully, the specified user (i.e. `jcooper` in the above example) should be created in the database table `OBAY_USER`.

For the `SellItem` and `BuyItem` services to work the user must be registered with oBay, and be defined in the repository used by the Identity Service.

By default, during installation of the SOA Suite, a sample user community is installed for use with the identity service (see the BPEL Process Managers Administrators Guide for details of the sample user community). The sample community is defined in the file:

```
<SOA_HOME> \j2ee\oc4j_soa\config\system-jazn-data.xml
```

Changes to this file need to be made manually and are only picked up on server restarts. We recommend for ease of use, that you just register users already defined in this sample community (e.g. `jcooper`, `jstein`, `istone`, `wfaulk`, and so on).

In a production deployment, you would typically configure the identity service to use an LDAP Repository such as Oracle Internet Directory or Active Directory; we could then modify the `UserRegistration` process to insert the user details into this repository.

getUser

For the specified `userId`, will return details of that user, it is invoked with a payload similar to the following:

```
<user:getUser xmlns:user="http://xmlns.packtpub.com/obay/evs/
UserAccount">
  <usr:userId xmlns:usr="http://schemas.packtpub.com/obay/
usr">jcooper</usr:userId>
</user:getUser>
```

getUserAccountTransactions

For the specified `userId` will return details of all transactions against the users account; i.e. Listings Fees, Sale Fees and Payments to oBay as well as the current balance of the account. It is invoked with a payload similar to the following:

```
<user:getUserAccountTransactions startRow="0" endRow="0"
xmlns:usr="http://schemas.packtpub.com/obay/usr" xmlns:user=
"http://xmlns.packtpub.com/obay/evs/UserAccount">
  <usr:userId>jcooper</usr:userId>
</user:getUserAccountTransactions>
```

By default this will list all transactions against an account, with the most recent listed first. However, this can be restricted by specifying the appropriate values for the attributes `startRow` and `endRow` (as highlighted).

For example, setting `startRow` to 1 and `endRow` to 10 will return the 10 most recent transactions.

ReferenceData

The WSDL for this service is located at the URL:

```
http://localhost:7001/oBay/EVS/ReferenceData?wsdl
```

This service provides the two following operations:

- `getCategory`
- `getSubCategories`

These operations will be required by both sellers and buyers to look up valid categories for listing and searching for items.

getCategory

Given a category code, will return the description for that category and its parent category code, it is invoked with a payload similar to the following:

```
<ref:getCategory  xmlns:ref="http://xmlns.packtpub.com/obay/evs/
ReferenceData"
xmlns:lst="http://schemas.packtpub.com/obay/lst">
 <lst:categoryCode>AUDIO</lst:categoryCode>
</ref:getCategory>
```

This will return a result similar to the following:

```
<ref:getCategoryResponse  xmlns:ref="http://xmlns.packtpub.com/obay/
evs/ReferenceData">
   <lst:category  xmlns:lst="http://schemas.packtpub.com/obay/lst">
     <lst:code>AUDIO</lst:code>
     <lst:description>Audio</lst:description>
     <lst:parentCategory>ELECTRON</lst:parentCategory>
   </lst:category>
</ref:getCategoryResponse>
```

getSubCategories

Given a category code, will return a list of all its sub categories, it is invoked with a payload similar to the following:

```
<ref:getSubCategories xmlns:ref="http://xmlns.packtpub.com/obay/evs/
ReferenceData"
xmlns:lst="http://schemas.packtpub.com/obay/lst ">
    <lst:categoryCode>ELECTRON</lst:categoryCode>
</ref:getSubCategories>
```

This will return a result similar to the following:

```
<ref:getSubCategoriesResponse  xmlns:ref="http://xmlns.packtpub.com/
obay/evs/ReferenceData">
  <lst:categoryList
  xmlns:lst="http://schemas.packtpub.com/obay/lst">
    <lst:category>
      <lst:code>AUDIO</lst:code>
      <lst:description>Audio</lst:description>
      <lst:parentCategory>ELECTRON</lst:parentCategory>
    </lst:category>
```

```
      <lst:category>
        <lst:code>GADGETS</lst:code>
        <lst:description>Gadgets</lst:description>
        <lst:parentCategory>ELECTRON</lst:parentCategory>
      </lst:category>
      <lst:category>
        <lst:code>MOVIEPLY</lst:code>
        <lst:description>Movie Players</lst:description>
        <lst:parentCategory>ELECTRON</lst:parentCategory>
      </lst:category>
      <lst:category>
        <lst:code>MP3</lst:code>
        <lst:description>MP3 Players</lst:description>
        <lst:parentCategory>ELECTRON</lst:parentCategory>
      </lst:category>
      <lst:category>
        <lst:code>TV</lst:code>
        <lst:description>Televisions</lst:description>
        <lst:parentCategory>ELECTRON</lst:parentCategory>
      </lst:category>
    </lst:categoryList>
  </ref:getSubCategoriesResponse>
```

Specifying a category code of `TOP` will return all the top level categories.

SellItem

The WSDL for this service is located at the URL:

```
http://localhost:7001/oBay/EVS/SellItem?wsdl
```

Listing an item for auction

To list an item for auction, you first need to invoke the `submitItemListing` operation, with a payload similar to the following:

```
<sel:submitItemListing xmlns:usr="http://schemas.packtpub.com/
obay/usr" xmlns:lst="http://schemas.packtpub.com/obay/lst"
xmlns:auc="http://schemas.packtpub.com/obay/auc" xmlns:sel="http://
xmlns.packtpub.com/obay/evs/SellItem">
  <usr:userId>jcooper</usr:userId>
  <lst:item>
    <lst:categoryCode>APPLE</lst:categoryCode>
    <lst:title>Black iPod</lst:title>
    <lst:description>16GB Black iPod in excellent condition</
lst:description>
    <lst:condition>USED</lst:condition>
  </lst:item>
  <lst:startTime></lst:startTime>
  <lst:duration>P7D</lst:duration>
  <lst:paymentMethod>Cash, Cheque</lst:paymentMethod>
  <lst:listingFormat>
    <lst:formatType>AUCTION</lst:formatType>
    <lst:stdAuctionFormat>
      <auc:auctionType>STD</auc:auctionType>
      <auc:startingPrice>5.00</auc:startingPrice>
      <auc:reservePrice></auc:reservePrice>
    </lst:stdAuctionFormat>
  </lst:listingFormat>
</sel:submitItemListing>
```

Note the following restrictions apply:

- **categoryCode**: This needs to be a category code defined in the database table CATEGORY (note this table is populated with sample categories as part of the database setup).
- **condition**: This should be either `NEW` or `USED`.
- **startTime**: This can either be empty (i.e. start immediately) or a future date.
- **duration**: This should either be `P1D`, `P3D`, `P7D` or `P10D`.
- **formatType**: This should be `AUCTION`.
- **auctionType**: This should be `STD`.
- **reservePrice**: This should either be empty or greater than the `startingPrice`.

The proxy service will route this request to the `ListItem` BPEL process, which will perform validation against the content (as described previously), causing the service to return a fault with details of the error should it fail validation.

Listing approved

Assuming the listing is approved a response similar to the following will be returned.

```
<sel:submitItemListingResponse
xmlns:lst="http://schemas.packtpub.com/obay/lst"
xmlns:sel="http://xmlns.packtpub.com/obay/evs/SellItem">
  <lst:listingId>212</lst:listingId>
```

```
    <lst:listingStatus>APPROVED</lst:listingStatus>
    <listingFee>0.40</listingFee>
  </sel:submitItemListingResponse>
```

This contains the `listingId` as well as the `listingFee`.

Suspect Item

The schematron `chkSuspectItem.sch` is used to check if the item may not be suitable for listing on oBay (at the moment it checks for the word drug, gun, or sex in either the item title or description). If the item is suspect then a response similar to the following will be returned.

```
<sel:submitItemListingResponse
xmlns:lst="http://schemas.packtpub.com/obay/lst"
xmlns:sel="http://xmlns.packtpub.com/obay/evs/SellItem">
  <lst:listingId>214</lst:listingId>
  <lst:listingStatus>SUSPECT</lst:listingStatus>
  <lst:listingFee>0.40</lst:listingFee>
</sel:submitItemListingResponse>
```

The `ListItem` process will then create a workflow task that will be assigned to the group `oBayAdministrator`. It contains the user's `cdoyle`, `mtwain` and `jcooper`. You will need to log into the **Worklist Application** and approve the listing before the seller can confirm that they want to list it.

confirmItemListing

Before a listing goes live, the seller must first approve it. To complete the `ListItem` process, you need to invoke the `confirmItemListing` operation, with a payload similar to the following:

```
<sel:confirmItemListing xmlns:sel="http://xmlns.packtpub.com/obay/evs/
SellItem"
xmlns:lst="http://schemas.packtpub.com/obay/lst">
  <lst:listingId>212</lst:listingId>
</sel:confirmItemListing>
```

Where the value in `listingId` contains the `listingId` returned by `submitItemListing`.

Once confirmed, the `listItem` process will invoke the `auctionItem` process (which will initiate the `auction` process). The item will now have a status of `LIVE` and potential buyers can submit bids against it.

If a `startTime` was specified then `listItem` will set the status of the item to `SCHEDULED` and wait until this time before invoking the `auctionItem` process.

getListings

A user can use the operation `getListings` to return a list of all items that they have listed; it is invoked with a payload similar to the following:

```
<sel:getListings startRow="0" endRow="0"
xmlns:lst="http://schemas.packtpub.com/obay/lst"
xmlns:sel="http://xmlns.packtpub.com/obay/evs/SellItem">
  <lst:sellerId>jcooper</lst:sellerId>
  <!--Optional:-->
  <sel:status>ALL</sel:status>
  <!--Optional:-->
  <sel:period>P30D</sel:period>
</sel:getListings>
```

This will return all items that match the specified search criteria, where:

- **sellerId**: `UserId` of the seller.
- **status**: This is optional (defaults to `ALL`), and can either be `PRE-LIVE` , `LIVE`, `SOLD`, `NOTSOLD`, or `ALL`. If `PRE-LIVE` will return those listings with a status of `AWAITING APPROVAL`, `APPROVED` or `SCHEDULED`. If `ALL` it will return all listings.
- **period**: This is optional, and needs to take the format PnD (where n can have a value between 1 to 999). When a status other than `PRE-LIVE` or `LIVE` is specified, it will only return listings that have started in the last n days. Defaults to 30 if not specified.

getListing

A user can use the operation `getListing` to return details of an item that they have listed; it is invoked with a payload similar to the following:

```
<sel:getListing
xmlns:lst="http://schemas.packtpub.com/obay/lst"
xmlns:sel="http://xmlns.packtpub.com/obay/evs/SellItem">
  <lst:listingId>212</lst:listingId>
</sel:getListing>
```

getBids

A user can use the operation `getBids` to return details of all bids placed against an item; it is invoked with a payload similar to the following:

```
<sel:getBids
xmlns:lst="http://schemas.packtpub.com/obay/lst"
xmlns:sel="http://xmlns.packtpub.com/obay/evs/SellItem">
  <lst:listingId>212</lst:listingId>
</sel:getBids>
```

This will return a response similar to:

```
<sel:getBidsResponse
xmlns:lst="http://schemas.packtpub.com/obay/lst"
xmlns:sel="http://xmlns.packtpub.com/obay/evs/SellItem"
xmlns:auc="http://schemas.packtpub.com/obay/auc">
  <lst:listingId>212</lst:listingId>
  <auc:winningBid>
    <auc:bidNo>3</auc:bidNo>
     <auc:bidderId>jstein</auc:bidderId>
     <auc:bidtime>2009-03-06T11:22:14+11:00</auc:bidtime>
     <auc:bidAmount>52.50</auc:bidAmount>
     <auc:status>WINNING</auc:status>
  </auc:winningBid>
  <auc:bidHistory>
    <auc:bid>
      <auc:bidNo>1</auc:bidNo>
      <auc:bidderId>jstein</auc:bidderId>
      <auc:bidtime>2009-03-06T11:21:10+11:00</auc:bidtime>
      <auc:bidAmount>50.00</auc:bidAmount>
      <auc:status>OUTBID</auc:status>
    </auc:bid>
    <auc:bid>
      <auc:bidNo>2</auc:bidNo>
      <auc:bidderId>jstein</auc:bidderId>
      <auc:bidtime>2009-03-06T11:22:03+11:00</auc:bidtime>
      <auc:bidAmount>50.00</auc:bidAmount>
      <auc:status>OUTBID</auc:status>
    </auc:bid>
  </auc:bidHistory>
</sel:getBidsResponse>
```

BuyItem

The WSDL for this service is located at the URL:

```
http://localhost:7001/oBay/EVS/BuyItem?wsdl
```

findListings

Before bidding on an item, a potential buyer must find an item that they are interested in. To do this we can use the `findListings` operation, with a payload similar to the following:

```
<buy:findListings  startRow="0" endRow="0"
xmlns:buy="http://xmlns.packtpub.com/obay/evs/BuyItem">
  <buy:categoryCode>MP3</buy:categoryCode>
    <buy:title>Black</buy:title>
```

```
      <!--Optional:-->
      <buy:condition>ALL</buy:condition>
    </buy:findListings>
```

This will return all **live** items that match the specified search criteria, where:

- **categoryCode**: This will restrict the search to any item in this category (or sub category).
- **title**: This will restrict the search to any item with the specified value in its title.
- **condition**: This will restrict the search to item that match the specified condition. It can be NEW, USED, or ALL (if not specified then it will default to ALL).

This will return a response similar to the following:

```
<buy:findListingsResponse
xmlns:buy="http://xmlns.packtpub.com/obay/evs/BuyItem"
xmlns:lis="http://xmlns.packtpub.com/obay/ivs/Listing"
xmlns:lst="http://schemas.packtpub.com/obay/lst">
  <lis:itemsList>
    <lst:listing
      <lst:listingId>212</lst:listingId>
      <lst:listingDetail>
        <lst:sellerId>jcooper</lst:sellerId>
        <lst:listingStatus>LIVE</lst:listingStatus>
        <lst:item>
          <lst:categoryCode>APPLE</lst:categoryCode>
          <lst:title>Black iPod</lst:title>
          <lst:description>16GB Black iPod in excellent
                           condition</lst:description>
          <lst:condition>USED</lst:condition>
        </lst:item>
        <lst:paymentMethod>Cash, Cheque</lst:paymentMethod>
        <lst:startTime>2009-03-03T19:40:19+11:00</lst:startTime>
        <lst:duration>P7D</lst:duration>
        <lst:listingFormat>
          <lst:formatType>AUCTION</lst:formatType>
        </lst:listingFormat>
        <lst:currentPrice>5.00</lst:currentPrice>
      </lst:listingDetail>
    </lst:listing>
  </lis:itemsList>
</buy:findListingsResponse>
```

submitBid

To place a bid on an item, a buyer uses the operation `submitBid` with a payload similar to the following:

```
<buy:submitBid xmlns:lst="http://schemas.packtpub.com/obay/lst"
xmlns:buy="http://xmlns.packtpub.com/obay/evs/BuyItem">
    <lst:listingId>212</lst:listingId>
    <lst:buyerId>jstein</lst:buyerId>
    <lst:amount>50.00</lst:amount>
</buy:submitBid>
```

Where it will return a response similar to the following:

```
<buy:submitBidResponse
xmlns:buy="http://xmlns.packtpub.com/obay/evs/BuyItem"
xmlns:lst="http://schemas.packtpub.com/obay/lst"
xmlns:auc="http://schemas.packtpub.com/obay/auc">
  <lst:listingId>212</lst:listingId>
  <auc:bid >
    <auc:bidNo>1</auc:bidNo>
    <auc:bidderId>jstein</auc:bidderId>
    <auc:bidtime>2009-03-03T20:34:49+11:00</auc:bidtime>
    <auc:maxAmount>50.00</auc:maxAmount>
    <auc:bidAmount>5.00</auc:bidAmount>
    <auc:status>WINNING</auc:status>
  </auc:bid>
</buy:submitBidResponse>
```

getBidOnListings

Once a user has placed a bid on an item, they can use the operation `getBidOnListings` to return a list of items that they have already bid on. It has a payload similar to the following:

```
<buy:getBidOnListings  startRow="0" endRow="0"
xmlns:lst="http://schemas.packtpub.com/obay/lst"
xmlns:buy="http://xmlns.packtpub.com/obay/evs/BuyItem">
  <lst:buyerId>jstein</lst:buyerId>
  <!--Optional:-->
  <buy:status>LIVE</buy:status>
  <!--Optional:-->
  <buy:period>P8D</buy:period>
</buy:getBidOnListings>
```

This will return all items that the specified user has bid on, filtered according to the search criteria, where:

- **buyerId**: `UserId` of the buyer.
- **status**: This is optional, and can either be `LIVE` or `ALL`. If `ALL` it will also return listings where the auction has finished. Defaults to `ALL` if not specified.
- **period**: This is optional, and needs to take the format `PnD` (where `n` can have a value between 1 to 999). When a status of ALL is specified, then this will only return listings that have started in the last `n` days. Defaults to 30 if not specified.

getListing

A user can use the operation `getListing` to return details of a listed; it is invoked with a payload similar to the following:

```
<buy:getListing
xmlns:lst="http://schemas.packtpub.com/obay/lst"
xmlns:buy="http://xmlns.packtpub.com/obay/evs/BuyItem">
    <lst:listingId>212</lst:listingId>
</buy:getListing>
```

getBids

A user can use the operation `getBids` to return details of all bids placed against an item; it is invoked with a payload similar to the following:

```
<buy:getBids
   xmlns:lst="http://schemas.packtpub.com/obay/lst"
   xmlns:buy="http://xmlns.packtpub.com/obay/evs/BuyItem">
    <lst:listingId>212</lst:listingId>
</buy:getBids>
```

Order fulfillment

The auction will end automatically after the specified duration, assuming there is a winning bid. The `OrderFulfillment` process will be invoked to complete the sale.

getPurchasedItems

The `BuyItem` service includes the `getPurchasedItems` operation, which will return to the buyer a summary list of all items that they have won, which are currently going through the OrderFulfillment process. The payload of the operation is similar to the following:

```
<buy:getPurchasedItems xmlns:com="http://xmlns.oracle.com/bpel/
workflow/common" xmlns:ord="http://schemas.packtpub.com/obay/ord"
xmlns:buy="http://xmlns.packtpub.com/obay/evs/BuyItem">
  <com:workflowContext>
    <com:credential>
      <com:login>jstein</com:login>
      <com:password>welcome1</com:password>
      <com:identityContext>jazn.com</com:identityContext>
    </com:credential>
  </com:workflowContext>
  <ord:buyerId>jstein</ord:buyerId>
</buy:getPurchasedItems>
```

The `workflowContext` contains the users login details required to authenticate against the identity service. On successful execution the operation will return a result similar to:

```
<buy:getPurchasedItemsResponse  xmlns:buy="http://xmlns.packtpub.com/
obay/evs/BuyItem"
xmlns:ord="http://schemas.packtpub.com/obay/ord">
  <buy:purchasedItemsList>
    <ord:orderSummary>
      <ord:orderNo>10209</ord:orderNo>
      <ord:orderDesc>Black iPod</ord:orderDesc>
      <ord:itemId>4</ord:itemId>
      <ord:sellerId>jcooper</ord:sellerId>
      <ord:buyerId>jstein</ord:buyerId>
      <ord:itemPrice>5.0</ord:itemPrice>
      <ord:totalPrice>5.0</ord:totalPrice>
      <ord:orderDate>2009-03-03T11:00:00+11:00</ord:orderDate>
      <ord:orderStatus>Pre-Shipment</ord:orderStatus>
      <ord:lastUpdateDate>2009-03-03T20:35:55</ord:lastUpdateDate>
      <ord:nextAction>Enter Shipping Details</ord:nextAction>
    </ord:orderSummary>
  </buy:purchasedItemsList>
</buy:getPurchasedItemsResponse>
```

Note the value in the `orderNo` field. This number needs to be specified when performing any other operations against the order. Also note the vale in the `nextAction` field. This specifies the next action that needs to be performed against the order.

getPurchasedItemDetails

The `BuyItem` service includes the operation `getPurchasedItemDetails`, which for the specified order number, returns the complete details of the order. The payload of the operation is similar to the following:

```
<buy:getPurchasedItemDetails xmlns:com="http://xmlns.oracle.com/bpel/
workflow/common" xmlns:ord="http://schemas.packtpub.com/obay/ord"
xmlns:buy="http://xmlns.packtpub.com/obay/evs/BuyItem">
  <com:workflowContext>
    <com:credential>
      <com:login>jstein</com:login>
      <com:password>welcome1</com:password>
      <com:identityContext>jazn.com</com:identityContext>
    </com:credential>
  </com:workflowContext>
  <ord:orderNo>10209</ord:orderNo>
</buy:getPurchasedItemDetails>
```

The `SellItem` service contains the equivalent operations; `getSoldItems` and `getSoldItemDetails.`

set<NextAction>

To complete fulfilment of the order it needs to proceed through a number of steps. The next step for an order is defined in the `nextAction` field; for each action there is a corresponding operation to be invoked as set out in the table below.

Next Action	Operation
Enter Shipping Details	`buyItem.setShippingDetails`
Enter Shipping Costs	`sellItem.setShippingCosts`
Make Payment	`buyItem.setPaymentMade`
Confirm Receipt of Payment	`sellItem.setPaymentReceived`
Ship Item	`sellItem.setItemShipped`
Confirm Receipt of Item	`buyItem.setItemReceived`

The payload for the operation `setShippingDetails`, will look similar to the following:

```
<buy:setShippingDetails xmlns:com="http://xmlns.oracle.com/bpel/
workflow/common" xmlns:ord="http://schemas.packtpub.com/obay/ord"
xmlns:cmn="http://schemas.packtpub.com/obay/cmn" xmlns:buy="http://
xmlns.packtpub.com/obay/evs/BuyItem">
  <com:workflowContext>
    <com:credential>
      <com:login>jstein</com:login>
      <com:password>welcome1</com:password>
      <com:identityContext>jazn.com</com:identityContext>
    </com:credential>
  </com:workflowContext>
  <ord:orderNo>10209</ord:orderNo>
  <ord:shipTo>
    <ord:shippingName>John Stein</ord:shippingName>
    <ord:shippingAddress>
      <cmn:addressLine1>38 Henry Street</cmn:addressLine1>
      <cmn:addressLine2></cmn:addressLine2>
      <cmn:city>Melbourne</cmn:city>
      <cmn:state>VIC</cmn:state>
      <cmn:zip>3000</cmn:zip>
      <cmn:country>Australia</cmn:country>
    </ord:shippingAddress>
    <ord:additionalInstructions>None</ord:additionalInstructions>
  </ord:shipTo>
</buy:setShippingDetails>
```

This consists of the `workflowContext`, `orderNo` plus the information that needs to be specified as part of this step (i.e. the `shipTo` information). The remaining operations follow a similar format.

On successful execution of each of these operation, it will return a response containing the updated order details.

Once all these steps have completed, the status of the order will be set to `FULFILLED` and the `OrderFulfillment` process will complete, this will cause the corresponding `auctionItem` and `listItem` processes to complete.

This completes the end to end process of selling an item.