Basic Tutorial on using the CentraSite Importing APIs Programmatically

This tutorial demonstrates you how to use CentraSite's importing APIs to import *WSDL* and *XSD* files. To implement the necessary methods with CentraSite, we provide an interface. CentraSiteJAXRImporter.java

What is WSDL?

WSDL (pronounced 'wiz-del') is the acronym for **W**eb **S**ervices **D**escription **L**anguage, which is an XML-based language for describing web services. All the necessary information that describes the web service's ports and what the web service does should be listed in the *WSDL* file.

What is XSD?

XSD is the acronym for **X**ML **S**chema **D**ocumentation. With this XML-based language it is possible to define the structures of XML documents.

CentraSite Importing API

There are some special classes for registering a web service to CentraSite. You can find these classes in the *CentraSiteUtils.jar* file.

WebServiceRegistrator

The `WebServiceRegistrator` takes care of importing *WSDL* files.

SchemaImporter

The `SchemaImporter` takes care of importing *XSD* files.

1. Establishing and Closing a CentraSite API for JAXR Connection

Opening a connection for version management works slightly differently than opening a normal connection, as we use the `JAXRAccessor` class for it. *Listing 1.1* shows you how to use methods to open and close such a connection. You should always use the JAXRAccessor as an attribute of your class; this has the advantage that you are always aware of the status of the connection.

Listing 1.1: openCon/ closeCon

```
private JAXRAccessor jaxr = null;

public void openCon() throws JAXRException {
    jaxr = new
JAXRAccessor("http://localhost:53305/CentraSite/CentraSite","Administrator",
"manage");
}

public void closeCon() {
    jaxr.close();
}
```

2. Import Functionalities

In this section, we show how to import WSDL and XSD files into the registry. First we demonstrate the basic functionality to do that; then, in section 2.3, we show an extended method of importing WSDL files which offers options for creating a new version and reusing or overwriting the referenced *WSDL* or XSD files.

2.1 Importing WSDL files

When importing a *WSDL* file that describes one or more services, the parameters that you have to supply are the path to the file, the name of the organization and the concept to which the service belongs. *Listing 2.1* demonstrates this using the straightforward method `importWSDL(...)`.

Listing 2.1: importWSDL

```
public void importWSDL(String wsdlFile, String orgName, String productConceptName)
throws Exception {
    WebServiceRegistrator wsr = new WebServiceRegistrator(wsdlFile, jaxr); //
create wWbServiceRegistrator with the given WSDL file
    wsr.setOrganization(orgName); // set the organization
    wsr.setProductConcept(productConceptName); // set the product concept
    wsr.register(); // register the service with the given information
}// end importWSDL
```

JAXRImporter.java

2.2 Importing XSD Files

Listing 2.2 shows you how to import an *XSD* file with the method `importXSD(...)`. Again, we need the *XSD* file as input for the creation of the `SchemaImporter` and again this is very easy to accomplish.

Listing 2.2: importXSD

```
public void importXSD(String xsdFile) throws Exception {
    SchemaImporter si = new SchemaImporter(xsdFile, jaxr); // create SchemaImporter
with given XSD file
    si.add(); // add the Schema to the registry
}// end importXSD
```

JAXRImporter.java

2.3 Customized Importing of WSDL Files

Last but not least, it is sometimes necessary to modify a WSDL or XSD file and then create a new version of the service. In these cases you want to create a new version of the service by re-importing the WSDL. This being the case, you have to decide whether to reuse or overwrite the referenced *XSD* files or to create a new version of them. This usually depends on whether they have been changed since the last import. In any case, you must make this decision for each *XSD* file that is referenced by the *WSDL*.
For this reason we create another method, called `customImportWSDL(...)`, which does this for us. As this way of importing *WSDL* files is somewhat more complicated and needs some user interaction, we describe it in detail. However, this alternative method gives us many ways of making changes to the *WSDL* files.

The most important thing to know is that no suitable interface is available to do that; rather, the processing is handled by an exception, which is thrown when an *XSD* file is referenced by a *WSDL* that already exists. This may seem to be a strange way of handling this task, unless you know that usually you are working in a GUI which can only handle such cases by passing events back to a higher level functionality. In our case these events are the exceptions which we then have to handle.

We now focus on the method shown in *Listing 2.3.1* and explain how it works, step by step.

1. Ask the user whether he wants to create a new version and create an import parameter object with the desired initialization. It is important to set the interactiveResolution to *TRUE* so that callbackexceptions are thrown. See *m1 - m2* in *Listing 2.3.1*.
2. Try to import the WSDL; this is similar to the `importWSDL(...)` method above. See *m2 - m3* in *Listing 2.3.1*.
3. If an XSD file is referenced by the WSDL, an exception is thrown and we have to decide what to do with the file from the exception. See *m3 - m4* in *Listing 2.3.1*. Then do this for each referenced XSD file. When all XSD files have been treated, the loop ends and the WSDL is stored in the registry.

Listing 2.3.1: customImportWSDL

```
// variables just for easier code understanding
private static final int REUSE = 1;
private static final int OVERWRITE = 2;
private static final int NEW_VERSION = 3;
```

```
public void customImportWSDL(String wsdlFile, String orgName, String
productConceptName) throws Exception {
    // m1
    boolean create = dialogueNew(); // console inquiry if user wants to create a
new version of the WSDL
    WebServiceRegistrator wsr = null;
    WsdlImportParameters importParameters = new WsdlImportParameters(jaxr); //
create import parameter object
    importParameters.setMode(WsdlImportParameters.MODE_REUSE_IMPORT_FILES); //
reuse all importet/included files in the registry
    importParameters.setInteractiveResolution(true); // callback exceptions can now
be thrown, by default this is false
    // m2
    boolean retry;
    do { // do try to register new version of service
        retry = false;
        try {
            wsr = new WebServiceRegistrator(wsdlFile, jaxr, importParameters, null,
null);
            wsr.setOrganization(orgName);
            wsr.setProductConcept(productConceptName);
            wsr.setCreateVersion(create); // if we want to create a new version,
'create' must be TRUE
            wsr.register(); // try to register the WSDL, i.e. web service
        // m3
        } catch(WSDLCallbackException we) { // if there are XSD files referenced by
the WSDL, we will get an exception
            WSDLCallbackParameters cb = we.getCallbackParameters(); // get the
information from the exception
            String action;
            switch(dialogueAction(cb.getLocation())) { // do what the user wants to
do with the XSD file from the exception
         case REUSE:
             importParameters.addReuseFileObject(cb, cb.getDefaultUsedObject()); //
use the file again
             break;
         case OVERWRITE:
                    action = "overwrite";
             importParameters.addReuseFileOverwrite(cb, dialoguePath(action)); //
overwrite the file with a new one
             break;
         case NEW_VERSION:
                    action = "create a new version of";
             importParameters.addReuseFileCreateVersion(cb, dialoguePath(action));
// create a new version
             break;
                default:
             importParameters.addReuseFileObject(cb, cb.getDefaultUsedObject()); //
the default is to use the file again
             break;
            } // end switch
            retry = true; // set retry to TRUE for another loop
        }// end catch
    // m4
    } while(retry); // until all XSD files were treated
```

```
    System.out.println("Registering completed!");
} // end customImportWSDL
```

JAXRImporter.java

As you can see in *Listing 2.3.1*, some additional methods are needed for displaying user dialogs on the console. These methods and their implementation are shown in *Listing 2.3.2*.

Listing 2.3.2: required methods

```
/**
* Console inquiry requesting if a new version of the WSDL should be created.
*
* @return
*/
public boolean dialogueNew() {
    boolean create;
    System.out.println("Do you want to create a new Version of the WSDL?");
    System.out.println("0: No");
    System.out.println("1: Yes");
    System.out.print("Decision: ");
    Scanner sc = new Scanner(System.in);
    int tmp = sc.nextInt();
    // check user input
    if(tmp == 0){ // if user input = 0
 create = false;
    } else if(tmp == 1){ // if user input = 1
 create = true;
    } else {
        create = false;
    }
    return create;
}// end createNewVersion

/**
* Console inquiry requesting what to do with the XSD file.
*
* @param name - the location and name of the XSD file
* @return
*/
public int dialogueAction(String name) {
    int action;
    System.out.println("What do you want to do with " + name + "?");
    System.out.println("1: REUSE");
    System.out.println("2: OVERWRITE");
    System.out.println("3: NEW_VERSION");
    System.out.print("Decision: ");
    Scanner sc = new Scanner(System.in);
    action = sc.nextInt();
    return action;
}// end xsdAction

/**
* Console inquiry requesting the path to the new XSD file.
*
* @param action - the chosen action
* @return
*/
public String dialoguePath(String action) {
    String path;
    System.out.println("You chose to " + action + " the file. Please specify the
path to the new file!");
    System.out.print("New File: ");
    Scanner sc = new Scanner(System.in);
    path = sc.nextLine();
    return path;
}// end getXSDPath
```

JAXRImporter.java

When using the method, the output on the console should look like this:

```
Do you want to create a new Version of the WSDL?
0: No
1: Yes
Decision: 0

What do you want to do with xsd/AmazonS3.xsd?
1: REUSE
2: OVERWRITE
3: NEW_VERSION
Decision: 2

You chose to overwrite the file. Please specify the path to the new file!
New File: D:\xsd\AmazonS3.xsd
Completed!
```