

# CIM IDENTITIES WEB SERVICE

*Using ICE 61968-100 standard*

## Table of Contents

Why use a Web Service? .....	4
Common Information Model IEC 61968 Messages .....	5
Header.....	5
Payload.....	5
Reply .....	5
Web Service Application Creation .....	6
Program Flow .....	6
Creating the Web Service in NetBeans .....	6
Organization.....	8
How this all works with Java .....	8
Creating the Header.....	10
Creating the Payload.....	12
Creating the Reply.....	12
Bringing It All Together .....	13
Web Service Client Creation .....	14
Testing the Web Service via SOAPUI.....	16
Designing the Web Service .....	18
QueryCIMIdentities.....	19
SendCIMIdentities.....	22
Designing the Client .....	24
Modifying the WSDLs.....	24
Adding Method Prototypes .....	25
Creating the Data Table .....	26
Network Setup .....	28
Configuring PostgreSQL for Remote Access .....	28
Configuring Firewall for Access.....	30
Required Software .....	30
PostgreSQL Installation .....	31
Troubleshooting.....	31
NetBeans and JDK .....	32
Oracle GlassFish Installation .....	32
Java.....	33



## Why use a Web Service?

When researching different aspects about the project, the question came up: Why do this as a web service instead of directly connecting to a database? A quick search of the question online pulled up this response on Stack Overflow

1. Security. You're not granting DB access to anyone but web server/app user. This is extra important when you have tons of users. You avoid the pain of user/role maintenance on DB side.
2. DB load reduction. Web service can cache the data it retrieved from DB.
3. Ability for fault tolerance - the service can switch between primary/DR data sources without having details of fail-over be implemented by service consumers.
4. Scalability - the service can spread requests between several parallel data sources without having details of the resource picking be implemented by service consumers.
5. Encapsulation. You can change underlying DB implementation without impacting service users.
6. Data enrichment (this includes anything from client customization to localization to internalization). Basically any of these might be useful but any of them is a major load on database and often very hard to implement inside a DB.
7. May or may not apply to you - certain architecture decisions are not DB aces friendly. E.g. Java Servers running on Unix have an easy access to a database, whereas a java client running on a Windows PC is not database aware nor do you possibly want it to be.
8. Portability. Your clients may not all be on the same platform/architecture/language. Re-creating a good data access layer in each one of those is harder (since it must take into account such issues as above-mentioned failovers/etc...) than building a consumer layer for a web service.
9. Performance tuning. Assuming the alternative is clients running their own queries (and not pre-canned stored procedures), you can be 100% sure that they will start using less than optimal queries. Also, if the web service bounds the set of allowable queries, it can help with your database tuning significantly. I must add that this logic is equally applicable to stored procedures, not unique to web services.

### What about this Web Service vs Client stuff?

There are two major pieces of a Web Service. One is the Web Service Application itself, and another is the Web Service client. The Web Service Application is in charge of defining the data (what type is it, what does it look like, etc.) and handles the direct connection to the database, while the client application is what the customer sees, i.e. the GUI (Graphical User Interface) that they use to send data to the server. The client is responsible for obtaining the data, and then the Web Service takes that data and handles the rest.

It is because of this that a client can get away with using an older version of software despite changes to the underlying Web Service.

## Common Information Model IEC 61968 Messages

### Header

#### *Required Fields*

- Verb
- Noun
- (if User) UserID
- (if Property) Name

#### *Optional Fields*

- Revision
- ReplayDetection
- Context
- Timestamp
- Source
- AsyncReplyFlag
- ReplyAddress
- AckRequired
- User
- MessageID
- CorrelationID
- Comment
- Property
- other

### Payload

#### *Required Fields*

None

#### *Optional Fields*

- Format (Hint as to format of payload e.g. XML, RDF, SVF, PDF, etc.)
- Other (For XML payload, usually CIM profiles defined using an XSD in a profile specific namespace)
- OperationSet
- Compressed
- ID
  - idType
  - idAuthority
  - kind

### Reply

The only required part of the Reply is the Result, which is either “OK”, “PARTIAL”, or “FAILED”. In this particular application, “PARTIAL” shouldn’t be possible, so check for `reply.value.getResult().equals(“OK”)`

or “FAILED”. The “FAILED” message should only appear upon some type of user-error or data connection error, therefore even exceptions should call the web service methods.

## Web Service Application Creation

### Program Flow

Most Java web applications consist of two separate applications: the web service and web service client. The web service contains the WSDL and XSD(s), as well as the Java files that describe those XSDs and their implementation. Put simply, the Web Service contains the generic information for using the Web Service, while the client would have more flexibility depending on the developer.

#### Web Service:

- Defines the structure of the data and what data is expected
- Implements the methods described in the WSDL
- Implements getter/setter methods for classes that represent each element in the XSDs

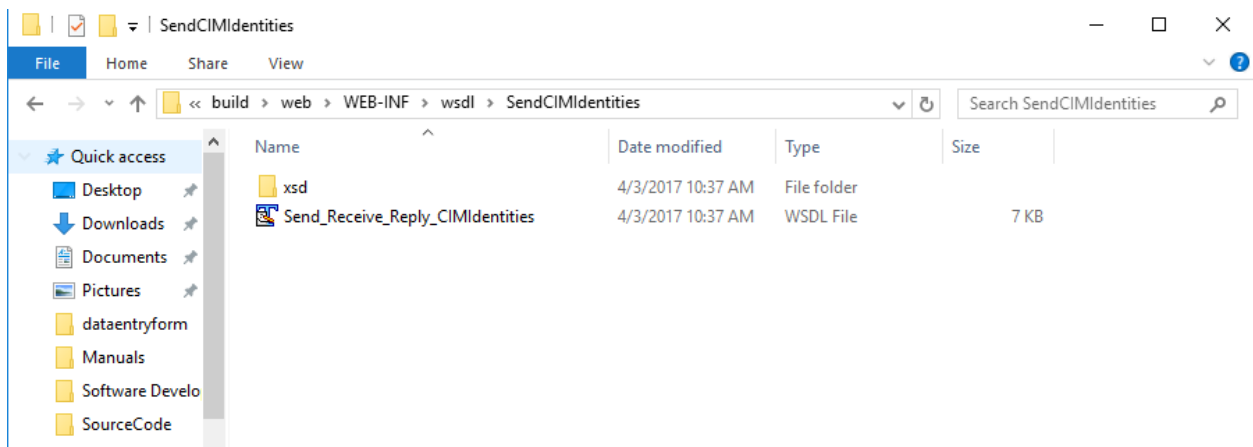
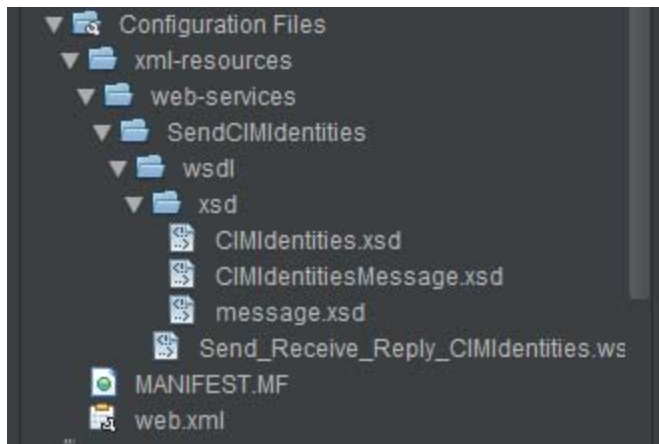
#### Web Service Client:

- Custom tailored application that makes use of the existing WSDL/XSDs
- Invokes methods in the Web Service that are defined in the WSDL, such as “createdCIMIdentities”.

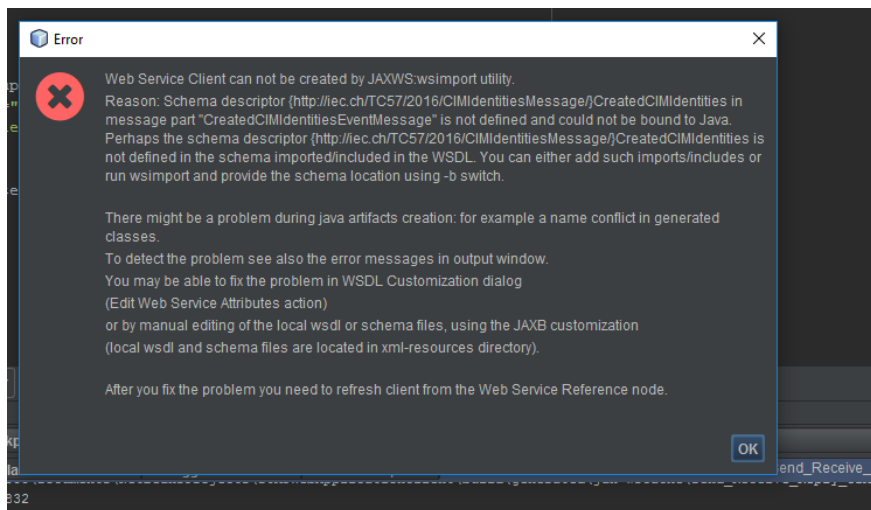
## Creating the Web Service in NetBeans

To create the SOAP web service in NetBeans, start a New Project and select Java Web under categories, and then Web Application under Projects. This will create an index.html file that isn’t needed, so feel free to delete that. In the Navigator, right click the project and select “New” -> “Web Service from WSDL” and select the WSDL file being used as the base. Since the WSDL in this case relies on three already-built XSDs, an error message will appear because it was unable to find the XSDs.

At this point, there should already be a directory with the WSDL located within, so navigate to that directory in the project folder, create a new folder named “xsd,” and place the necessary XSD files within. It should end up looking like so:

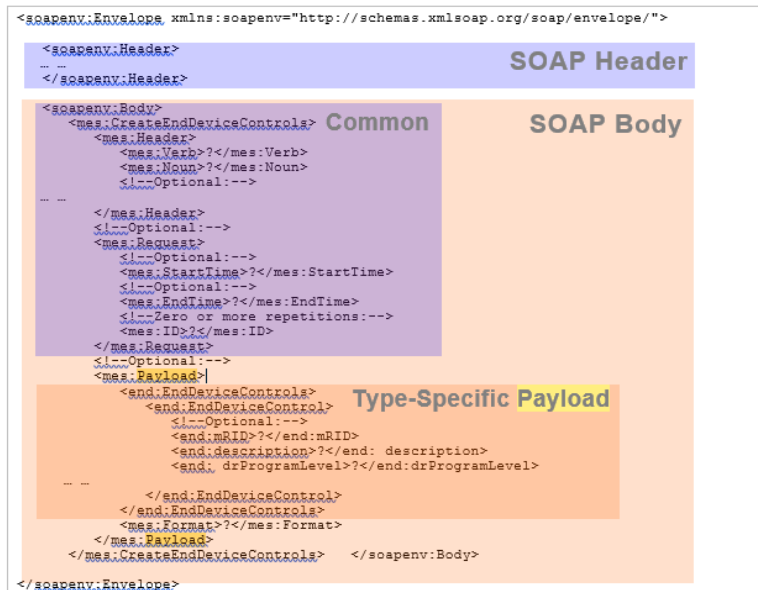


Be mindful of error messages received during this process. For instance, a common error could be the following:



## Organization

Payload will contain the meat the data transfer. While the header will hold the information for whether a modification, deletion, or insertion is occurring, the Payload will contain the information that is being changed, deleted, or added. The following is an example Payload:



## How this all works with Java

For each XML element (Name, NameType, etc.) there exists a corresponding Java file for it (Name.java, NameType.java, etc.) and each Java file has corresponding getter/setter methods (e.g. getName(), setName(String value), etc.).

Taking this from the top:

1. Get CIMIdentity() returns an ArrayList of type CIMIdentity
2. CIMIdentity is first used to obtain the mRID by calling IdentifiedObject getIdentifiedObject()
  - a. Class IdentifiedObject contains the methods String getMRID() and void setMRID(String value).
  - b. CIMIdentity.setIdentifiedObject(IdentifiedObject value) is the other related UUID method in CIMIdentity
  - c. Lastly, CIMIdentity has method public List<Name> getNames() which returns an array list of names.
3. Name has getName() and setName(String value) methods, as well as public NameType getNameType() and setNameType(NameType value).
4. NameType and NameType Authority class are identical, with getter/setter methods for Name and description.

```
public void createdCIMIdentities(Holder<ch.iec.tc57._2016.schema.message.HeaderType> header,
Holder<ch.iec.tc57._2016.cimidentitiesmessage.CIMIdentitiesPayloadType> payload,
```



Holder<ch.iec.tc57.\_2016.schema.message.ReplyType> reply) throws  
ch.iec.tc57.\_2016.sendcimidentities.FaultMessage { ... }

The above is one of the provided java functions for createdCIMIdentities. In order to use it, it is necessary to understand what the Holder type is and how it works.

**Holder** is from the javax.xml.ws.Holder library. With SOAP it is possible to return multiple values in a single request. This is impossible in Java as a method can only return one object.

JAX-WS solves this problem with the concept of Holders. A javax.xml.ws.Holder is a simple wrapper object that can be passed into the @WebService method as a parameter. The application sets the value of the holder during the request and the server will send the value back as an OUT parameter.

Each Holder is of the following types:

- HeaderType (defined in Message.xsd)
  - String verb = “create”, “delete”, “change” etc.
  - String noun = “CIMIdentities”
- PayloadType (defined in CIMIdentitiesMessage.xsd)
  - CIMIdentities type defined in CIMIdentities.xsd
    - All classes defined above
- ReplyType (defined in Message.xsd)
  - String result – required

In the web service java file, you can do header.value.setName(“example\_name”); or header.value.getName() if you pass in a defined data set.

For payload, you would do payload.value.getCIMIdentities() but it gets more complicated:

- getCIMIdentities is defined in CIMIdentitiesMessage.xsd. It returns a “cimidentities” object defined in CIMIdentities.xsd
- CIMIdentities contains the function getCIMIdentity(), which returns a List of CIMIdentity type
- Follow the chain outlined above.

In other words, payload.value.getCIMIdentities().getCIMIdentity(); should return a list of all data going into the database.

Once you obtain this list, you should probably assign it to a value as you’ll be using it a lot. In the case of my code example, I did

```
ArrayList<CIMIdentity> cim = (ArrayList<CIMIdentity>) payload.value.getCIMIdentities().getCIMIdentity();
```

This is where things can be a little confusing. What we have is an ArrayList of CIM Identities. However, the application this is being developed for only allows submission of one CIM Identity at a time, therefore we only need to access index 0 of this ArrayList. Thus, assigning values would appear as follows:

```

ArrayList<CIMIdentity> cim = (ArrayList<CIMIdentity>) payload.value.getCIMIdentities().getCIMIdentity();
|
mRID = cim.get(0).getIdentifiedObject().getMRID();
NName = cim.get(0).getNames().get(0).getName();
NTName = cim.get(0).getNames().get(0).getNameType().getName();
NTDes = cim.get(0).getNames().get(0).getNameType().getDescription();
NTAName = cim.get(0).getNames().get(0).getNameType().getNameTypeAuthority().getName();
NTADes = cim.get(0).getNames().get(0).getNameType().getNameTypeAuthority().getDescription();

```

In this case, we call get(0) because in the XSD/WSDL file, code can be written so that multiple CIM Identities are changed/modified/deleted at once. However, for this application, only one CIM Identity is being changed/modified/deleted at a time, therefore its information will always exist at index 0.

This doesn't deal with one major problem with the original application, however....the way it's designed, the user can enter an mRID or have one randomly generated. However, the web service REQUIRES a UUID already. In this case, we must go ahead and create the UUID before executing the web method, which while require a database connection within both the client and host application.

## Creating the Header

### Nouns

When creating the Header, two fields are needed: Noun and Verb. As per IEC 61968, Nouns are used to identify the type of the information being exchanged. These are also commonly called profiles. Each noun typically has a corresponding XML Schema definition defined using a namespace unique to each noun. Nouns are typically identified by use cases. Within a message, the noun is used to identify the type of the payload or the type of object to be acted or has been acted upon. Some common example nouns taken from IEC 61968-9 are:

- EndDeviceControls
- EndDeviceEvents
- MeterReadings

Nouns can be defined as needed to distinguish the contents of different information flows. They need not be defined as classes in a UML model, but instead the contents and structure of the noun are defined using classes, attributes and relationships from a UML model. In this case, our Noun will be "CIMIdentities".

### Verb

IEC 61968-1 identifies a set of verbs, where annex B of this standard defines a normative list. This sub-clause is to provide more specificity on the usage of each verb and identify the deprecation some verbs as well as synonyms. In the following table verbs used for requests are associated with the verb that should be used on a response message and as would be used for publication of an event, where often events are a consequence of the successful completion of a transaction initiated by a request.

Request Verb	Reply Verb	Event Verb	Usage
get	reply	(none)	query
create	reply	created	transaction
change	reply	changed	transaction
cancel	reply	canceled	transaction
close	reply	closed	transaction
delete	reply	deleted	transaction
execute	reply	executed	transaction

When creating a new entry in the table, the verb will be “create”, and when modifying an entry will use the verb “change”. Removing an entry will use the verb “delete”.

The usage of request verbs are as follows.

- ‘get’ is used to query for objects of the type specified by the message noun
- ‘create’ is used to create objects of the type specified by the noun
- ‘delete’ is used to delete objects, although sometimes an object is never truly deleted in the target system in order to maintain a history
- ‘close’ and ‘cancel’ imply actions related to business processes, such as the closure of a work order or the cancellation of a control request
- ‘change’ is used to modify objects, but it is important to note that there can be ambiguities that need to be addressed through business rules, especially in the case of complex data sets (e.g. complex data sets typically have N:1 relationships and it is important to be clear when relationships are additive or are to be replaced by an update).
- ‘execute’ is used when a complex transaction is being conveyed using an OperationSet, which potentially contains more than one verb.

The response to each of the above requests uses the ‘reply’ verb. Event verbs are often the consequence of a request, where a ‘create’ may result in the generation of a ‘created’ event. The verbs used for events use the ‘past tense’ form of the associated request verb. There is no requirement that event be initiated through a request, as it may be appropriate for events to be generated independently of any specific request.

Validation and business rules may need to be defined for application of verbs in specific cases. This is in part true in that many rules are beyond the descriptive capabilities of UML and XML Schema.

It is also important to note that the enumerations for verbs in the standard Message XML Schema use the **lower case** form. The uppercase form is otherwise convenient for documentation purposes.

IEC 61968-1 previously identified verbs ‘update’, ‘updated’, ‘show’, ‘subscribe’, ‘unsubscribe’ and ‘publish’, all of which have been deprecated. The reason is that ‘show’ is a synonym for ‘reply’, and the

verbs 'subscribe', 'unsubscribe' and 'publish' are functions that are performed within the transport layer (e.g. using JMS).

### Creating the Payload

Payload is used to convey message information as a consequence of the 'Verb' and 'Noun' combination in the message Header. Required for 'create', 'change' and 'execute' requests. It is also required for event messages. Optional in other cases as described later in this document and specifically within annex B. The payload structure provides options for payload compression. For this application, separate web methods are created for use depending on the verb, so a verb of "create" should only be possible within the createdCIMIdentities method, while "change" should only be in changedCIMIdentities().

For this application, the payload must contain an entry for each field, even if just an empty string. Thus the payload in this case will contain the

- mRID
- Name
- NameType Name
- NameType Description
- NameType Authority Name
- NameType Authority Description

### Creating the Reply

The reply according to the 61968 standard is required only for response messages to indicate success, failure and error details. Not used for request or event messages.

The Reply.result value is enumerated in Message.xsd, and would be populated in the following manner:

- "OK" if there are no errors and all results have been returned. There is no requirement that a Reply.Error element be present.
- "PARTIAL" if only a partial set of results has been returned, with or without errors. Existence of errors is indicated with one or more Reply.Error.code elements.
- "FAILED" if no result can be returned due to one or more errors, indicated with one or more Reply.Error elements, each with a mandatory application level 'code' '.

#### 4.3 Request/Reply Using an ESB

The simple request/reply use case can also be extended to leverage an ESB. Within the ESB many actions can be taken by intermediaries as needed to facilitate integration and the decoupling of components, such as transformations and routing.

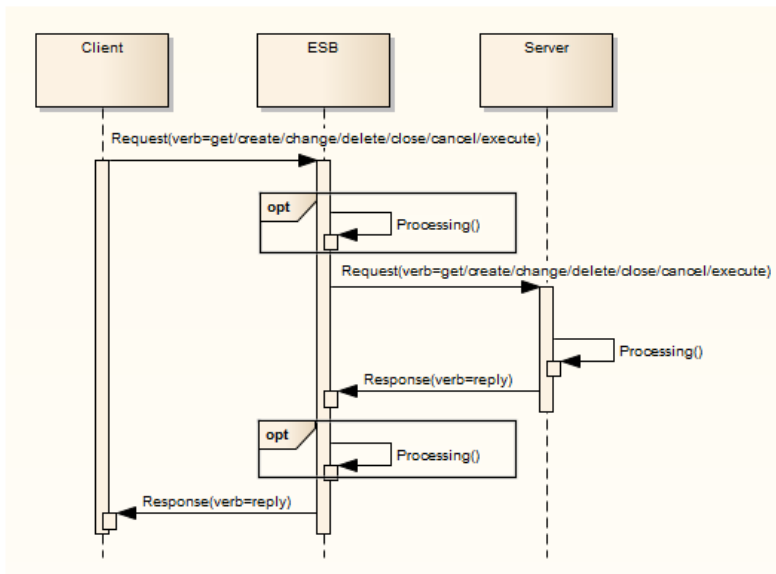


Figure 3 - Request/Reply Using Intermediaries

When NetBeans builds the code for you based off the existing WSDL/XSDs, it creates a method with the parameters header/payload/reply. However, the client shouldn't be pushing a reply to the Web Service, so that parameter should be removed from the method, but still set within the method by creating a global variable for the Web Service.

#### Bringing It All Together

According to the oracle documentation:

*"When you and your business partners agree on a "contract" in the form of a WSDL file, you can use the IDE to implement it. The WSDL file is an agreement on the data and messages that will be exchanged as well as how these messages will be sent and received. In the IDE, you can use the WSDL file to implement the web service.*

*Business requirements may demand that you create a platform-independent description of a web service as a set of XML schema files and WSDL files. Based on this platform-independent description, you can use the steps that follow to generate the implementation files. The WSDL file that you use in the steps below can either be available on disk or via a URL.*

1. Depending on the implementation form, create a web application project or an EJB module project.

2. In the Projects window or Files window, right-click the project node and choose **New > Other**. In the Web Services folder, choose **Web Service from WSDL**.

The New Web Service from WSDL wizard opens.

3. Type the web service name and specify a package to house the web service.
4. Browse to a WSDL file in your filesystem or type the URL to a WSDL file.
5. Select **Use Provider** if you want to bypass the XML <--> Java binding layer and have the service use raw XML when processing requests. Instead of XML <--> binding, the Provider interface is used. This is an advanced feature and is unselected by default.
6. Click **Finish**.

The IDE runs the `wsimport` tool, which reads the WSDL file and generates all the required artifacts for web service development, deployment, and invocation. Finally, the IDE generates the skeleton implementation class, which corresponds to the WSDL port selected in the wizard.”

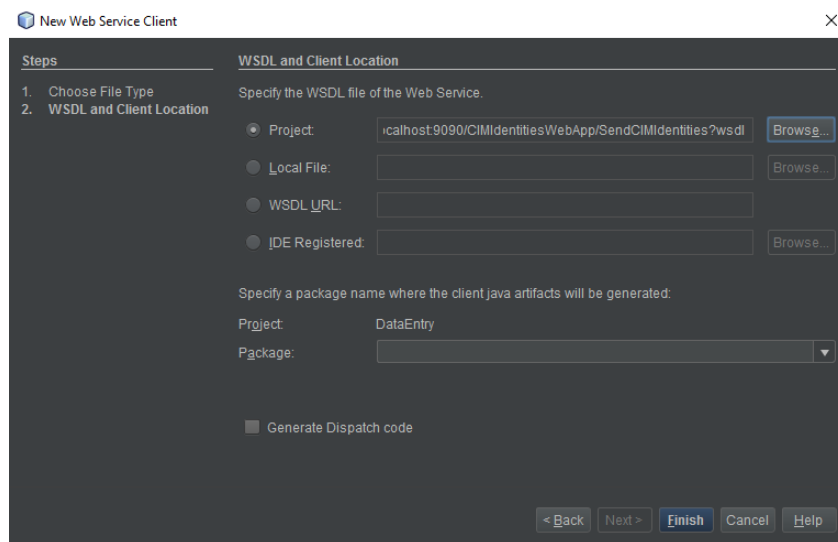
Therefore our “skeleton implementation class” in this case is `SendCIMIdentities.java`, which contains methods for created/changed/deleted CIM Identities. By default, extra methods are provided by the WSDL but are not needed for the purpose of this application, and those methods are changed/closed CIM Identities.

```
ArrayList<CIMIdentity> cim = (ArrayList<CIMIdentity>) payload.value.getCIMIdentities().getCIMIdentity();  
|  
mRID = cim.get(0).getIdentifiedObject().getMRID();  
NName = cim.get(0).getNames().get(0).getName();  
NTName = cim.get(0).getNames().get(0).getNameType().getName();  
NTDes = cim.get(0).getNames().get(0).getNameType().getDescription();  
NTAName = cim.get(0).getNames().get(0).getNameType().getNameTypeAuthority().getName();  
NTADes = cim.get(0).getNames().get(0).getNameType().getNameTypeAuthority().getDescription();
```

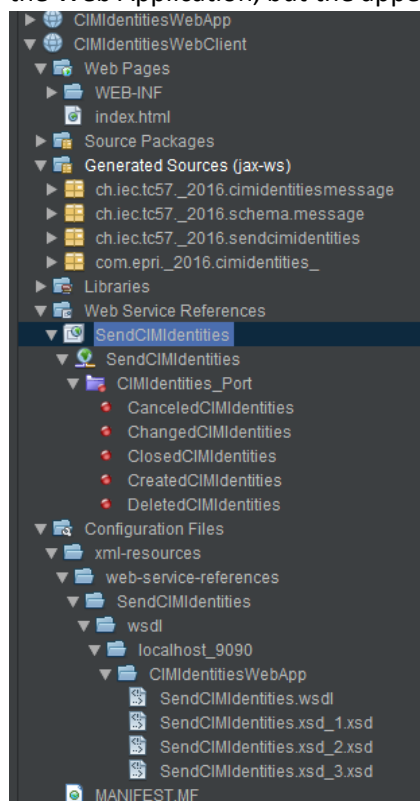
We will be calling the set methods instead of the get methods.

## Web Service Client Creation

Once the Web Service application has been constructed, it is time to create the Web Service Client, which will pass the methods used in the Web Service. To create a client version, create a New Project and select Java Application. Once created, right click and select “New” and “Web Service Client”. You will be importing the Project built for the Web Application, as seen in the figure below.



Once this has been done, NetBeans will create a new project that imports all of the Java files created for the Web Application, but the appearance will be slightly different, as seen in the figure below.

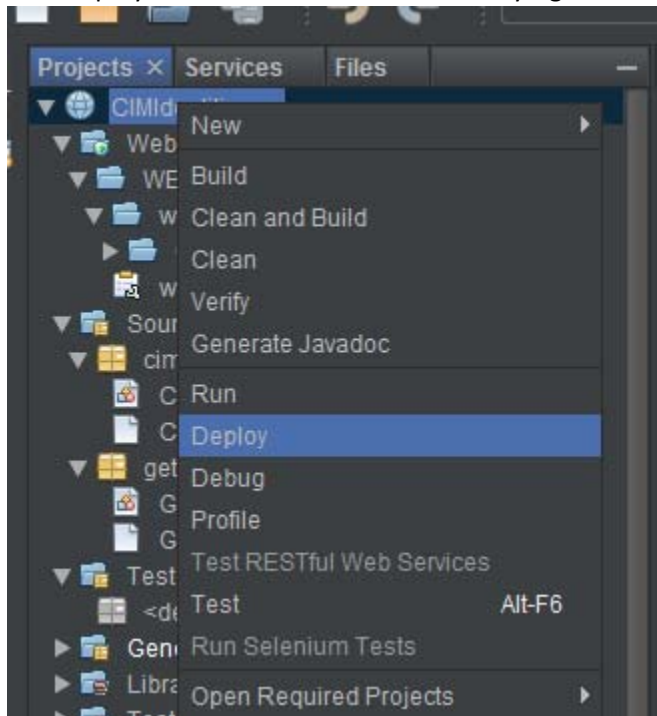


Alternatively, you can create a regular Java Application and perform a similar process of creating a New Web Service Client. Once created, open up the source code file and right click anywhere you can type code, select “Insert Code” → “Call Web Service Operation” → “SendCIMIdentities” → “CIMIdentities\_Port” → each method you plan on implementing. This will automatically insert the entire function for you. No changes need to be made.

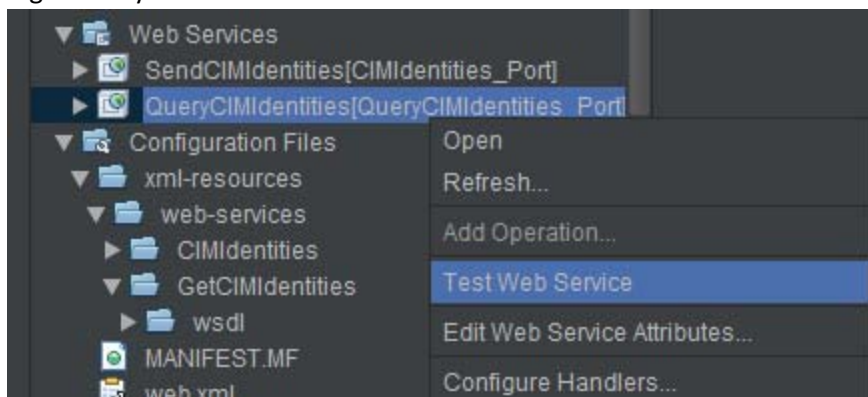
## Testing the Web Service via SOAPUI

Once the web service is ready to be tested, there are two ways to go about it: either write a client application to use the service, or use SOAPUI. Since SOAPUI already is a client used for testing Web Services, this should be the first choice in testing the service. The following guide will outline how to test the SOAP Web Service in this project.

- First deploy the web service in NetBeans by right clicking the project and selecting **Deploy**



- Right click your web service and select **Test Web Service**





- Once the web service launches in your browser, select the link for the WSDL file pictured here

## QueryCIMIdentities Web Service Tester

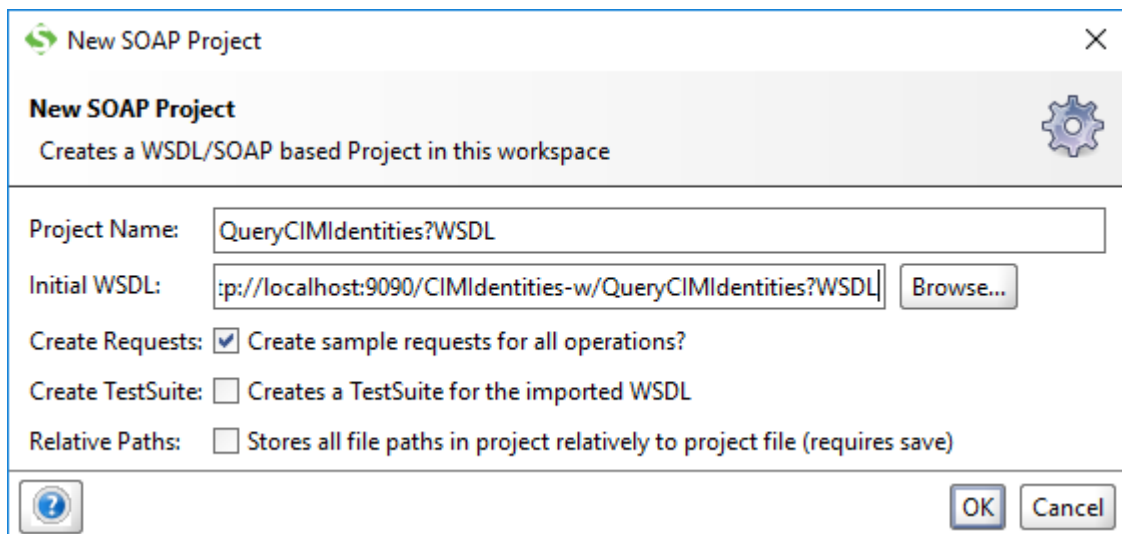
This form will allow you to test your web service implementation ([WSDL File](#))

To invoke an operation, fill the method parameter(s) input boxes and click on the button

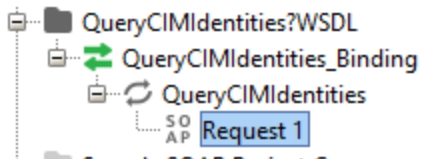
- Copy the link at the top. It should look something like this.

[localhost:9090/CIMIdentities-w/QueryCIMIdentities?WSDL](http://localhost:9090/CIMIdentities-w/QueryCIMIdentities?WSDL)

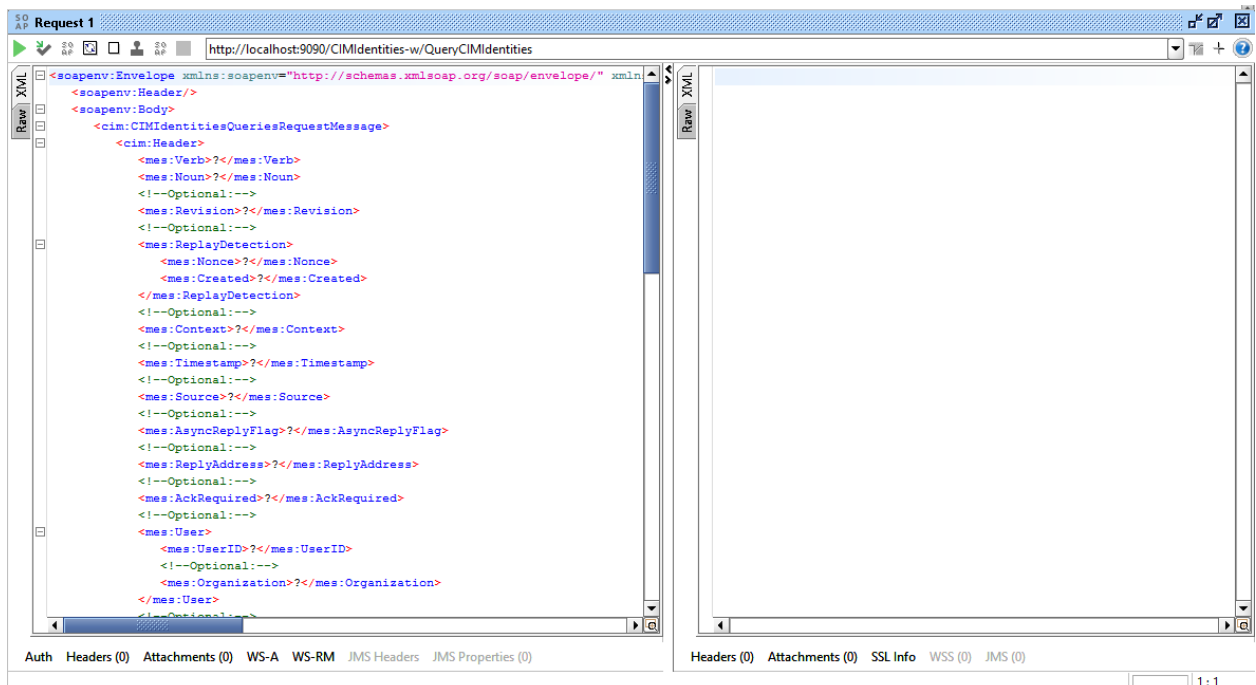
- In SOAPUI, select **File -> New SOAP Project** and paste that address in the textbox for **Initial WSDL** as shown below.



- Select **OK** and then check the Projects Navigator tab on the left for the newly added project. Expand **QueryCIMIdentities\_Binding** and select **Request 1**.



- The window below should appear for testing the project. In this particular example, providing nothing at all will return every entry in the database.



- Note the link at the top: <http://localhost:9090/CIMIdentities-w/QueryCIMIdentities>. In this example, the web service was being hosted on the local machine. However, if the web service is setup for remote access, “localhost” would need to be replaced with the hosting machine’s correct IP Address. To change this in the WSDL file, navigate to this location:

```
<wsdl:service name="QueryCIMIdentities">
  <wsdl:port name="QueryCIMIdentities_Port" binding="tns:QueryCIMIdentities_Binding">
    <soap:address location="http://iec.ch/TC57/2016/QueryCIMIdentities"/>
  </wsdl:port>
</wsdl:service>
```

The current URL listed above is a “dummy” URL and should be replaced if hosting a remote service.

## Designing the Web Service

When creating the web service, two WSDLs were used to create two separate Web Services within the Web Application: Send\_Receive\_Reply\_CIMIdentities and CIMIdentities\_Query. As their names imply, one was used to define a service for creating, changing, canceling, modifying, or deleting data. The other was used purely for retrieval of data. Each required multiple XSDs for further defining the information present. The following list is of the XSDs used to create the Web Application (Bold font denotes shared XSDs):

- CIMIdentities.xsd**
- CIMIdentitiesMessage.xsd
- Message.xsd**

- CIMIdentitiesQueries.xsd
- CIMIdentitiesQueriesMessage.xsd

## QueryCIMIdentities

The QueryCIMIdentities web service is meant for retrieval of data. If the request message sent to the web service contains no UUID, then all data in the database is returned to the client. Otherwise information is sent back regarding the UUID only.

**Note:** This code is written as proof-of-concept and thus makes no attempt to protect against SQL Injection. Please be mindful when designing your own Web Service. The UUID isn't type-checked here at all and is assumed that the user will provide correct input.

```
public ch.iec.tc57._2016.cimidentitiesqueriesmessage.CIMIdentitiesQueriesResponseMessageType
queryCIMIdentities
    (ch.iec.tc57._2016.cimidentitiesqueriesmessage.CIMIdentitiesQueriesRequestMessageType message)
        throws QueryCIMIdentitiesFaultMessage {

    //create response message object
    ch.iec.tc57._2016.cimidentitiesqueriesmessage.CIMIdentitiesQueriesResponseMessageType response = new
ch.iec.tc57._2016.cimidentitiesqueriesmessage.CIMIdentitiesQueriesResponseMessageType();

    //create header object to assign to response. Set to "get" by default
    ch.iec.tc57._2011.schema.message.HeaderType header = new
ch.iec.tc57._2011.schema.message.HeaderType();
    header.setVerb("get");
    header.setNoun("CIM Identities");
    response.setHeader(header);

    //create reply object that if successfully is "OK", if not is "FAILED"
    ReplyType value = new ReplyType();
    value.setResult("OK");
    response.setReply(value);

    //create Payload object for response message
    ch.iec.tc57._2016.cimidentitiesqueriesmessage.CIMIdentitiesQueriesPayloadType payload = new
ch.iec.tc57._2016.cimidentitiesqueriesmessage.CIMIdentitiesQueriesPayloadType();
    response.setPayload(payload);

    //extract individual values for insertion into database

    try {
        String query;
        String uuid = message.getRequest().getCIMIdentitiesQueries().getEndDeviceGroup().get(0).getMRID();
        if (uuid == null || uuid.equals("") || uuid.equals("?")) {

            query = "SELECT *" +
                "FROM public.\"NameType\" as nt, public.\"Name\" as n, " +
                "public.\"NameTypeAuthority\" as nta " +
                "WHERE n.n_pkey = nt.nt_pkey AND " +
```

```

        "nt.nt_pkey = nta.nta_pkey " +
        "ORDER BY n.n_name ASC;";
    } else {
        query = "SELECT *" +
            "FROM public.\"NameType\" as nt, public.\"Name\" as n, " +
            "public.\"NameTypeAuthority\" as nta " +
            "WHERE n.n_pkey = " + uuid + " AND " +
            "n.n_pkey = nt.nt_pkey AND " +
            "nt.nt_pkey = nta.nta_pkey";
    }

    Connection conn = DriverManager.getConnection(host, uName, password);
    Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
    ResultSet rs = stmt.executeQuery(query);

    int size = 0;
    if (rs != null) {
        rs.beforeFirst();
        rs.last();
        size = rs.getRow();
    }
    rs.beforeFirst();

    CIMIdentities cimIDs = new CIMIdentities();
    payload.setCIMIdentities(cimIDs);
    ArrayList<CIMIdentity> cim = (ArrayList<CIMIdentity>) payload.getCIMIdentities().getCIMIdentity();
    cim.ensureCapacity(size); //sets size of the ArrayList to size of the Result Set

    int i = 0;

    while (rs.next()) {
        //create new CIM Identity objects for each object that will be assigned later
        //when querying the database

        CIMIdentity cimid = new CIMIdentity();
        IdentifiedObject idObj = new IdentifiedObject();
        Name name = new Name();
        cim.add(i, cimid);

        List<Name> names = cim.get(i).getNames();
        NameType nameType = new NameType();
        NameTypeAuthority nameTypeAuthority = new NameTypeAuthority();

        //set the mRID
        idObj.setMRID(rs.getString("n_pkey"));
        cim.get(i).setIdentifiedObject(idObj);

        names.add(0, name);
        name.setName(rs.getString("n_name"));
        names.get(0).setName(name.getName());
    }

```

```

        nameType.setName(rs.getString("nt_name"));
        nameType.setDescription(rs.getString("nt_description"));
        names.get(0).setNameType(nameType);

        nameTypeAuthority.setName(rs.getString("nta_name"));
        nameTypeAuthority.setDescription(rs.getString("nta_description"));
        names.get(0).getNameType().setNameTypeAuthority(nameTypeAuthority);

        i++;
    }
    rs.close();
    stmt.close();
    conn.close();
    response.setPayload(payload);

} catch (Exception err) {
    value.setResult("FAILED");
    err.printStackTrace();
}
response.setReply(value);

return response;

```

To use this in a client, code might look like so:

```

public void createResponse() {
    HeaderType header = new HeaderType();
    header.setNoun("CIMIdentities");
    header.setVerb("get");

    CIMIdentitiesQueriesRequestType request = new CIMIdentitiesQueriesRequestType();
    CIMIdentitiesQueries var = new CIMIdentitiesQueries();
    EndDeviceGroup edg = new EndDeviceGroup();
    edg.setMRID(null); //can be null, '?', or '' to receive all data, else set mRID

    message.setRequest(request);
    message.getRequest().setCIMIdentitiesQueries(var);
    message.getRequest().getCIMIdentitiesQueries().getEndDeviceGroup().add(0, edg);
    message.getRequest().getCIMIdentitiesQueries().getEndDeviceGroup();

    message.setHeader(header);

    try {
        response = queryCIMIdentities(message);
    } catch (QueryCIMIdentitiesFaultMessage ex) {
        Logger.getLogger(CIMIdentitiesClient.class.getName()).log(Level.SEVERE, null, ex);
    }
    if (response.getPayload().getCIMIdentities().getCIMIdentity() != null)
        responseSize = response.getPayload().getCIMIdentities().getCIMIdentity().size();
}

```

Note the creation of new objects for the Request message before it's sent. Even if a no UUID is provided by the client, it is still necessary to create empty objects to avoid a Null Pointer Exception being thrown. Note that the client in this case has no mechanism for retrieving data for a specific UUID. Since the client in this case has no use for it, code was never written for anything but an empty GetCIMIdentities request. However, modifying the client to handle both is as simple as passing a String as parameter in the createResponse() method, such that the String can either contain the UUID or be blank.

As for the queryCIMIdentities() method, the IDE should have a mechanism for inserting the prototype method definition, which in this project appeared as thus:

```
private static CIMIdentitiesQueriesResponseMessageType
queryCIMIdentities(ch.iec.tc57._2016.cimidentitiesqueriesmessage.CIMIdentitiesQueriesRequestMessageType
queryCIMIdentitiesRequestMessage) throws QueryCIMIdentitiesFaultMessage {

    ch.iec.tc57._2016.querycimidentities.QueryCIMIdentities service = new
ch.iec.tc57._2016.querycimidentities.QueryCIMIdentities();

    ch.iec.tc57._2016.querycimidentities.QueryCIMIdentitiesPort port = service.getQueryCIMIdentitiesPort();

    return port.queryCIMIdentities(queryCIMIdentitiesRequestMessage);
}
```

## SendCIMIdentities

The SendCIMIdentities web service is responsible for the following types of requests (bold denotes its use in this project):

- Canceled
- **Changed**
- Closed
- **Created**
- **Deleted**

JAX-WS created the method prototype definitions, and also provided @WebService definitions. Each Web Method took a CIMIdentitiesEventMessageType type as parameter and returned a message of type CIMIdentitiesResponseMessageType. The following code shows how the createdCIMIdentities method was created.

```
public class CIMIdentities {
    String mRID;
    String NName;
    String NTName;
    String NTDes;
    String NTAName;
    String NTADes;
```

```

String host = "jdbc:postgresql://144.58.246.143:5432/CIMIdentity";
String uName = "postgres";
String password = "epri97!!";

public ch.iec.tc57_2016.cimidentitiesmessage.CIMIdentitiesResponseMessageType
createdCIMIdentitiesRequest
    (ch.iec.tc57_2016.cimidentitiesmessage.CIMIdentitiesEventMessageType message) {

    ch.iec.tc57_2016.cimidentitiesmessage.CIMIdentitiesResponseMessageType response = new
ch.iec.tc57_2016.cimidentitiesmessage.CIMIdentitiesResponseMessageType();
    //Event message type only contains the header/payload
    //Response message type contains header/payload/reply
    //therefore, response message type must be set for CIMIdentitiesResponseMessageType
    ReplyType value = new ReplyType();
    //String result = value.getResult();
    value.setResult("OK");

    //set response header/payload/reply
    response.setHeader(message.getHeader());
    response.setPayload(message.getPayload());
    response.setReply(value);

    //extract individual values for insertion into database
    ArrayList<CIMIdentity> cim = (ArrayList<CIMIdentity>)
response.getPayload().getCIMIdentities().getCIMIdentity();
    mRID = cim.get(0).getIdentifiedObject().getMRID();
    NName = cim.get(0).getNames().get(0).getName();
    NTName = cim.get(0).getNames().get(0).getNameType().getName();
    NTDes = cim.get(0).getNames().get(0).getNameType().getDescription();
    NTAName = cim.get(0).getNames().get(0).getNameType().getNameTypeAuthority().getName();
    NTADes = cim.get(0).getNames().get(0).getNameType().getNameTypeAuthority().getDescription();

    try {

        Connection con = DriverManager.getConnection(host, uName, password);
        Statement stmt = con.createStatement();

        //first check the mRID. If it's an empty string, treat it the same
        //as uuidEntered == false
        if (mRID.equals("") || mRID.equals("?")) {

            String genUUID = "INSERT INTO public.\"Identity\" (id_pkey, entry)"
                + "VALUES (DEFAULT, DEFAULT)";
            stmt.executeUpdate(genUUID);
            String getUUID = "SELECT id.id_pkey FROM \"Identity\" id "
                + "ORDER BY id.entry desc LIMIT 1";
            ResultSet rs = stmt.executeQuery(getUUID);
            rs.next();
            mRID = rs.getString("id_pkey");

            response.getPayload().getCIMIdentities().getCIMIdentity().get(0).getIdentifiedObject().setMRID(mRID);
        } else {
            String entUUID = "INSERT INTO public.\"Identity\"(id_pkey, entry)"
                + "VALUES ('" + mRID + "', DEFAULT)";

```

```

        stmt.executeUpdate(entUUID);
    }

    String insertIDObj = "INSERT INTO public.\"IdentifiedObject\"(io_pkey)"
        + "VALUES('" + mRID + "')";
    String insertName = "INSERT INTO public.\"Name\"(n_pkey, n_name)"
        + "VALUES('" + mRID + "', '" + NName + "')";
    String insertNT = "INSERT INTO public.\"NameType\"(nt_pkey, nt_description, nt_name)"
        + "VALUES ('" + mRID + "', '" + NTDes + "', '" + NTName + "')";
    String insertNTA = "INSERT INTO public.\"NameTypeAuthority\"(nta_pkey, nta_name,
nta_description)"
        + "VALUES ('" + mRID + "', '" + NTAName + "', '" + NTADes + "')";

    stmt.executeUpdate(insertIDObj);
    stmt.executeUpdate(insertName);
    stmt.executeUpdate(insertNT);
    stmt.executeUpdate(insertNTA);

    stmt.close();
    con.close();

    } catch(SQLException err){
        value.setResult("FAILED");
        System.out.println(err.getMessage());
    }

    return response;

    //TODO implement this method
    //throw new UnsupportedOperationException("Not implemented yet.");
}

```

As with the GetCIMIdentities web service, the code in this case doesn't protect against SQL Injection. Also note all of the required fields. If there is no entry for any of these, a Null Pointer Exception will be thrown, so be sure that the client passes in data or an empty string.

## Designing the Client

The basics for creating the client were described earlier, however, the purpose of this section is to outline the overall design of the client. The beauty of a SOAP web service is that a client can be written in a programming language of the programmer's choice. The client can be a website, a phone App, or a desktop application. In this project, the client was created as a desktop application.

## Modifying the WSDLs

When building the client, if both the web service and client are being developed by the same person, then it is important to be sure that the soap:address location is set properly. For example, to host your webservice at a machine with the IP address 144.58.246.100, the soap address for the



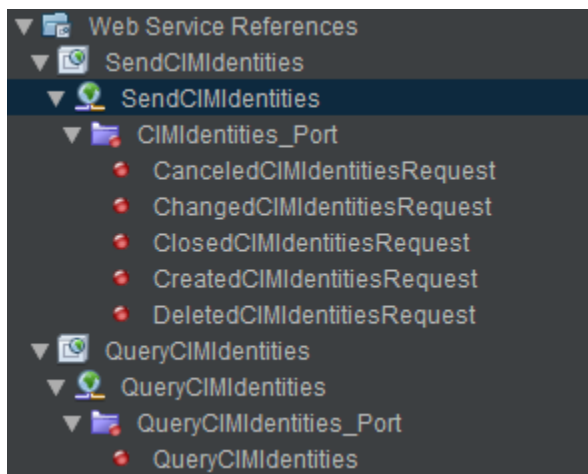
SendCIMIdentities web service might be:

<http://144.58.246.143:9090/CIMIdentities-w/SendCIMIdentities>

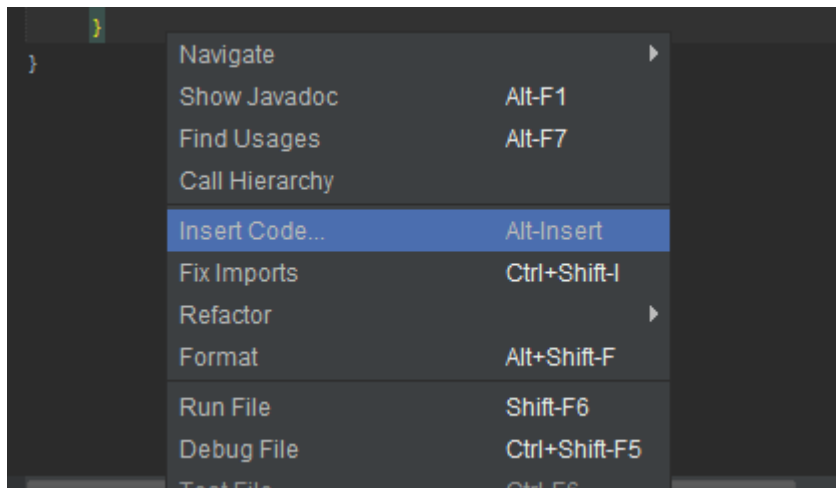
This **must** match up with the web service WSDL. It is possible to setup a more typical web address, for example, google.com is also <http://216.58.195.78/>. In this way, if a domain name service is setup for the host's IP address, that can be used instead.

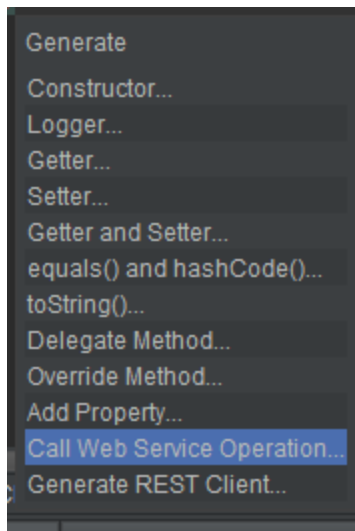
## Adding Method Prototypes

Adding the function prototypes can be done in 2 ways. You can either drag and drop from the Projects Navigator tab, or right click within the code editor and insert the code.



*Drag and drop the needed web methods*





This will give provide function prototypes implemented in the web service such as queryCIMIdentities, createdCIMIdentitiesRequest, etc.

## Creating the Data Table

Creating a table to represent the data within the database required some mathematical logic in order to create a table with pagination ability. In order to create the table, it helps to be familiar with a few different Java data structures: **DefaultTableModel** and **ResultSet**.

**DefaultTableModel:** *This is an implementation of **TableModel** that uses a vector of vectors to store the cell value objects.* Check its official Oracle documentation at <https://docs.oracle.com/javase/7/docs/api/javax/swing/table/DefaultTableModel.html> for a list of supported methods.

**ResultSet:** *A table of data representing a database result set, which is usually generated by executing a statement that queries the database.* Each row of a data is store in the ResultSet. An integer called “index” was used to keep track of where the ResultSet’s cursor was pointing. See the official Oracle documentation at <https://docs.oracle.com/javase/7/docs/api/java/sql/ResultSet.html> for a list of supported methods and examples on how to use them.

```
public void createTable() {  
    curPage = Integer.parseInt(curPageBox.getText());  
    numRows = Integer.parseInt(numRowsBox.getText());  
    int index = 0;  
    DefaultTableModel Model = new DefaultTableModel();  
    Object[] colName = new Object[6];  
    colName[0] = "mRID";  
    colName[1] = "Name";
```

```

colName[2] = "NT Name";
colName[3] = "NT Des";
colName[4] = "NTA Name";
colName[5] = "NTA Des";
Model.setColumnIdentifiers(colName);

createResponse();

Object[] rowData = new Object[6];
for (int i = 0; i < responseSize; i++) {
    rowData[0] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(i).getIdentifiedObject().getMRID();
    rowData[1] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(i).getNames().get(0).getName();
    rowData[2] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(i).getNames().get(0).getNameType().getName();
    rowData[3] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(i).getNames().get(0).getNameType().getDescription();
    rowData[4] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(i).getNames().get(0).getNameType().getNameTypeAuthority().getName();
    rowData[5] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(i).getNames().get(0).getNameType().getNameTypeAuthority().getDescription();
    Model.addRow(rowData);
}
//now we add obtain the total number of rows:
int totalRows = Model.getRowCount();
//using that total number of rows, we computer the number of pages
totalPage = (totalRows/numRows) + 1;
totalPageBox.setText(Integer.toString(totalPage));

//move index to starting value in current page
index = ((curPage-1) * numRows);
//now loop through the result set adding to new model
DefaultTableModel newModel = new DefaultTableModel();
newModel.setColumnIdentifiers(colName);
if ((curPage != totalPage) && (curPage < totalPage) && (curPage >= 1)) {
    for (int i = numRows; i > 0; i--) {
        rowData[0] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(index).getIdentifiedObject().getMRID();
        rowData[1] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(index).getNames().get(0).getName();
        rowData[2] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(index).getNames().get(0).getNameType().getName();
        rowData[3] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(index).getNames().get(0).getNameType().getDescription();
        rowData[4] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(index).getNames().get(0).getNameType().getNameTypeAuthority().getName();
        rowData[5] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(index).getNames().get(0).getNameType().getNameTypeAuthority().getDescription();
    }
}

```

```

        newModel.addRow(rowData);
        index++;
    }
    } else if (curPage == totalPage) {
        for (int i = (totalRows % numRows); i > 0; i--) {
            rowData[0] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(index).getIdentifiedObject().getMRID();
            rowData[1] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(index).getNames().get(0).getName();
            rowData[2] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(index).getNames().get(0).getNameType().getNam
e();
            rowData[3] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(index).getNames().get(0).getNameType().getDesc
ription();
            rowData[4] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(index).getNames().get(0).getNameType().getNam
eTypeAuthority().getName();
            rowData[5] =
response.getPayload().getCIMIdentities().getCIMIdentity().get(index).getNames().get(0).getNameType().getNam
eTypeAuthority().getDescription();
            newModel.addRow(rowData);
            index++;
        }
    }
    } else {
        JOptionPane.showMessageDialog( null, "Invalid page number");
        return;
    }
    dataTable = new javax.swing.JTable();
    dataTable.setModel(newModel);
    jScrollPane3.setViewportViewView(dataTable);
}

```

## Network Setup

In order to access the Web Application remotely instead of through localhost, a few changes are going to have to be made to the config file for PostgreSQL to allow remote access. Additionally, changes will likely need to be made to the host machine's firewall settings to allow the port connections.

### Configuring PostgreSQL for Remote Access

On the PostgreSQL database server, by default, you'll notice the following records towards the end of the C:\Program Files\PostgreSQL\9.6\data\pg\_hba.conf. As indicated below, it accepts connections only from the localhost.

# IPv4 local connections:				
host	all	all	127.0.0.1/32	md5
# IPv6 local connections:				
host	all	all	::1/128	md5

Add the following line to the **pg\_hba.conf** file. This will allow connection from “144.58.246.143” ip-address (This is the client in our example). If you want to allow connection from multiple client machines on a specific network, specify the network address here in the CIDR-address format. See the PostgreSQL Tutorial at <https://www.postgresql.org/docs/9.1/static/auth-pg-hba-conf.html> for more information on connecting multiple hosts to the client machine.

host	all	all	144.58.246.143/24	md5
------	-----	-----	-------------------	-----

Next, change the listen address in **postgresql.conf**. By default this is typically localhost. To get this to listen to all addresses, change the value for listen\_addresses to ‘\*’. Otherwise, list only desired addresses.

**Note:** It is highly recommended *not* to set listen\_addresses to ‘\*’ if connections are known in advance (for example, if only work connections are allowed, it can be set up to only listen to IP addresses that follow a specific subset of IP addresses).

```

postgresql - Notepad
File Edit Format View Help

#data_directory = 'ConfigDir'           # use data in another directory
#                                     # (change requires restart)
#hba_file = 'ConfigDir/pg_hba.conf'      # host-based authentication file
#                                     # (change requires restart)
#ident_file = 'ConfigDir/pg_ident.conf'  # ident configuration file
#                                     # (change requires restart)

# If external_pid_file is not explicitly set, no extra PID file is written.
#external_pid_file = ''                  # write an extra PID file
#                                     # (change requires restart)

#-----
# CONNECTIONS AND AUTHENTICATION
#-----

# - Connection Settings -

listen_addresses = '*'                  # what IP address(es) to listen on;
#                                     # comma-separated list of addresses;
#                                     # defaults to 'localhost'; use '*' for all
#                                     # (change requires restart)
port = 5432                             # (change requires restart)
max_connections = 100                   # (change requires restart)
#superuser_reserved_connections = 3     # (change requires restart)
#unix_socket_directories = ''           # comma-separated list of directories
#                                     # (change requires restart)
#unix_socket_group = ''                 # (change requires restart)
#unix_socket_permissions = 0777        # begin with 0 to use octal notation

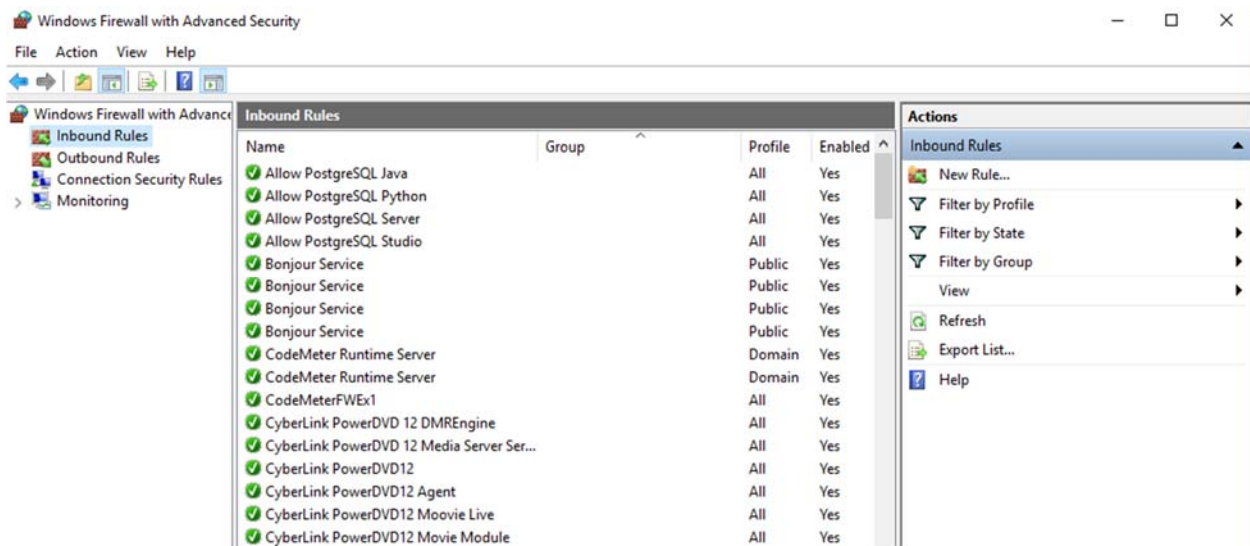
```

## Configuring Firewall for Access

The following ports are used in this web application:

- 5432 (for PostgreSQL)
- 8080 (for the Web Service)
- 9090 (for the Web Service)

By default, most firewalls are fairly strict and likely won't allow access to these ports, so Inbound Rules must be made to allow access. First, press the windows key to open the start menu and type "Firewall" to bring up **Windows Firewall with Advanced Security**.



Right click **Inbound Rules** on the left and select "New Rule". On the next menu select "Port". On the next page, "TCP" should already be checked so leave that as it is, and under "Specific local ports:" enter in the specific port to be allowed (e.g. 5432, 8080, 9090). On the next page, select "Allow the connection" and on the page after that, all three boxes should be checked for Domain, Private, and Public. Lastly give it a name such as "PostgreSQL connection" or "Web Service Connection" and click "Finish". If connect problems consist, it may be necessary to consult with an IT specialist.

## Required Software

Software needed to build this application include the following:

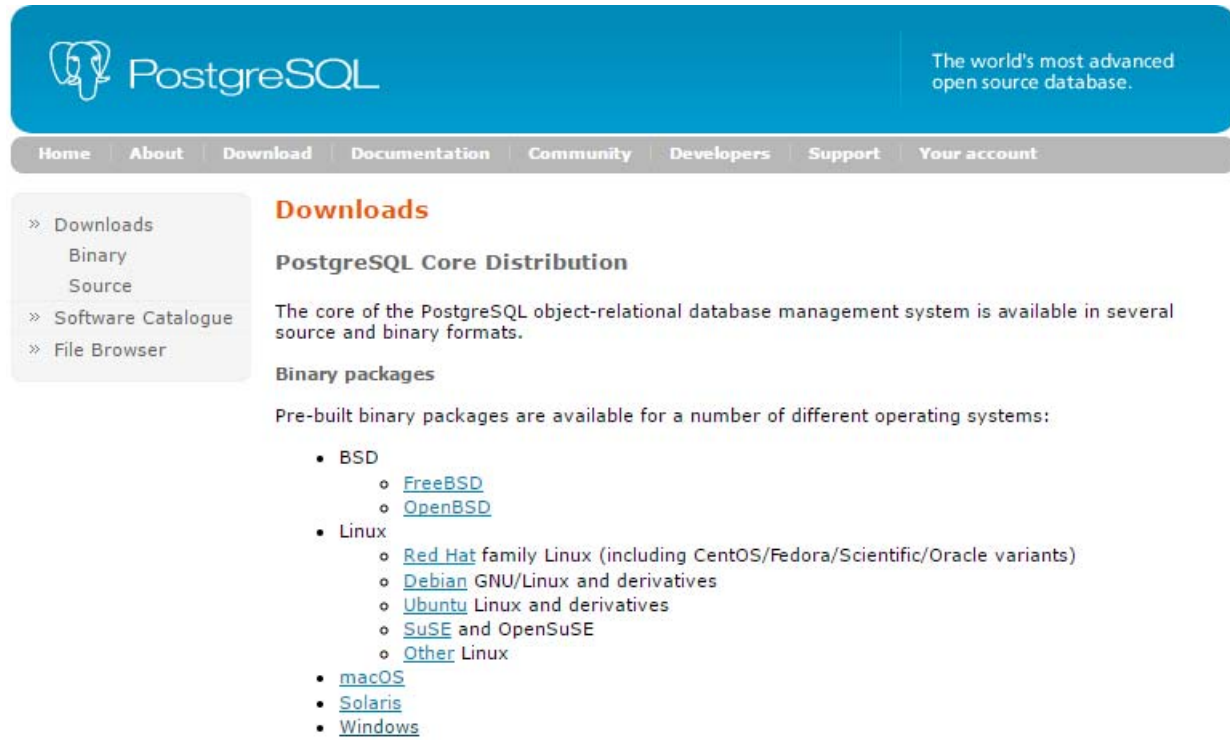
- PostgreSQL - <https://www.postgresql.org/>
- Java - <https://www.java.com/en/>
- Netbeans and JDK - <http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html>
- Glassfish - <http://www.oracle.com/technetwork/middleware/glassfish/downloads/index.html>

For the client machines, the only necessary piece of software is Java in order to run the application.

As for downloading and installing software for the host machine, brief instructions will be provided here, but are subject to change at any time at the will of the 3<sup>rd</sup> Party Software's development team.

## PostgreSQL Installation

PostgreSQL is an open-source, free object-relational database system. It is supported on various Operating Systems including Linux, UNIX, and Windows. To install it, navigate to the download page and select the appropriate Operating System on which to install it.



The screenshot shows the PostgreSQL website's 'Downloads' page. The header is blue with the PostgreSQL logo and the tagline 'The world's most advanced open source database.' Below the header is a navigation bar with links: Home, About, Download, Documentation, Community, Developers, Support, and Your account. On the left is a sidebar with links: » Downloads (with sub-links Binary and Source), » Software Catalogue, and » File Browser. The main content area is titled 'Downloads' and 'PostgreSQL Core Distribution'. It states that the core of the PostgreSQL object-relational database management system is available in several source and binary formats. Under 'Binary packages', it lists pre-built binary packages for various operating systems:

- BSD
  - [FreeBSD](#)
  - [OpenBSD](#)
- Linux
  - [Red Hat](#) family Linux (including CentOS/Fedora/Scientific/Oracle variants)
  - [Debian](#) GNU/Linux and derivatives
  - [Ubuntu](#) Linux and derivatives
  - [SuSE](#) and OpenSuSE
  - [Other](#) Linux
- [macOS](#)
- [Solaris](#)
- [Windows](#)

The Windows version is used for this tutorial, but instructions should be similar for other Operating Systems as well. Click **Download the installer** under **Interactive installer by EnterpriseDB** next. This should install the PostgreSQL server and pgAdmin, among other tools. When prompted to set up a user name and password, keep the defaults (such as 'postgres' for user) and set the password to whatever is preferred. This password will be used to login to the PostgreSQL database and will also be used in the Web Service code for connecting to the database.

## Troubleshooting

Troubleshooting PostgreSQL installation is beyond the scope of this tutorial. Common problems can be easily searched and resolved online, however. A common solution to issues with the installation is to temporarily disable either the Firewall or Antivirus program (or both).

See [https://wiki.postgresql.org/wiki/Troubleshooting\\_Installation](https://wiki.postgresql.org/wiki/Troubleshooting_Installation) for troubleshooting help.

## NetBeans and JDK

This software was built using NetBeans 8.1 (current version is 8.2 as of time-of-writing). To develop a Java application, it is necessary to obtain the Java Development Kit (JDK) before programming. There are multiple IDEs that can be used alongside the JDK such as Eclipse, IntelliJ, or NetBeans, but for the purpose of this project, NetBeans will be used as the example. To download both, navigate to <http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html> to download the latest version.






[Overview](#) | [Downloads](#) | [Documentation](#) | [Community](#) | [Technologies](#) | [Training](#)

### JDK 8u131 with NetBeans 8.2

This distribution of the JDK includes the Java SE bundle of [NetBeans IDE](#), which is a powerful integrated development environment for developing applications on the Java platform. [Learn more](#)

You must accept the [JDK 8u131](#) and [NetBeans 8.2](#) Cobundle License Agreement to download this software.

☐ Accept License Agreement ☒ Decline License Agreement

Java SE and NetBeans Cobundle (JDK 8u131 and NB 8.2)		
Product / File Description	File Size	Download
Linux x86	291.05 MB	 <a href="#">jdk-8u131-nb-8_2-linux-i586.sh</a>
Linux x64	286.33 MB	 <a href="#">jdk-8u131-nb-8_2-linux-x64.sh</a>
Mac OS X x64	342.15 MB	 <a href="#">jdk-8u131-nb-8_2-macosx-x64.dmg</a>
Windows x86	319.21 MB	 <a href="#">jdk-8u131-nb-8_2-windows-i586.exe</a>
Windows x64	329.42 MB	 <a href="#">jdk-8u131-nb-8_2-windows-x64.exe</a>

## Oracle GlassFish Installation

When installing Glassfish, the download page at <http://www.oracle.com/technetwork/middleware/glassfish/downloads/index.html> lists several different options. Since the JDK already comes packaged with NetBeans, the only link viable here is the one labeled “Oracle GlassFish Server x.x.x.x”, pictured below.



## Oracle GlassFish Downloads

### Previous Releases



### Java

JRE, or Java Runtime Environment, will be needed by clients using the Web Service application. When navigating to <https://www.java.com/en/>, the “Free Java Download” link should automatically redirect to a page with the OS-specific download screen. For instance, on a Windows 10 machine, the redirect will be to <https://www.java.com/en/download/win10.jsp>. From there, click on “Agree and Start Free Download” and follow the instructions provided by the installation wizard. Note that it may ask to change your web browser’s home page, so be mindful of messages that pop up before clicking “Next”.

## Download Java for Windows

**Recommended Version 8 Update 131 (filesize: 721 KB)**

Release date April 18, 2017

**Agree and Start Free  
Download**

By downloading Java you acknowledge that you have read and  
accepted the terms of the [end user license agreement](#)



When your Java installation completes, you may need to restart your browser (close all browser windows and re-open) to enable the Java installation.