reconnect
COMMUNITY BUSINESS INTELLIGENCE

User Implementation Guide

Version 1.03.08

Community Business Intelligence
c/o Reconnect Mental Health Services
56 Aberfoyle Crescent, Suite 400 Toronto, ON M8X 2W4
Tel. +1-416-248-2050

Published: June, 2013

Version	Date Released
1.03.08	July 25, 2014
1.03	June 2013

Table of Contents

1 Overview

This document is a web services application programming interface and implementation guide for the Community Business Intelligence (CBI) Project.

The Toronto Central Local Heath Integration Network (TC LHIN) has determined the need to improve upon current decision support capabilities within the community sector and is establishing an integrated Community Sector Database that will include client-level information about service utilization within TC LHIN- funded Health Service Providers (HSP) across three sub-sectors: Community Mental Health (CMH), Community Addiction (CA), and Community Support Services (CSS). Reconnect Mental Health Services (Reconnect) was chosen by the TC LHIN as the Project Sponsor to lead the development of a CBI tool that would address this need for the Toronto Community Information Infrastructure (TCII) community.

The CBI project facilitates the collection of information from an HSP's Client Management System (CMS) for the purposes of reporting at the individual and aggregate levels. A CBI tool is being developed to give HSPs and the TC LHIN that ability to query data as well as receive standard reports. HSPs will receive data quality reporting that will alert them to any data quality issues within their current CMS and will allow them to compare data quality statistics to sector averages.

Because data collection at HSPs is implemented using diverse technologies and methods, a single, standardized method is needed to facilitate the interoperability and gathering of data into a single database for analysis, reporting and strategic planning purposes. The CBI Web Service is the central gathering point for data that will be collected from HSPs.

The Drug and Alcohol Treatment Information System (DATIS) has been selected as the electronic service provider to house the community sector data that will be collected from HSPs and act as the centralized reporting repository. DATIS is an initiative of the Centre for Addiction and Mental Health and is funded by the Ministry of Health and Long-Term Care.

This guide is intended for developers working at software vendors who provide CMSs for HSPs funded by the TC LHIN to develop and implement the CBI Tool.

1.1 Assumptions and Prerequisites

This section describes the technical and knowledge assumptions and prerequisites required to create a web services implementation using this document as a guide.

1.1.1 Programming Knowledge and Experience

Programmers who have been hired as a Vendor by an HSP should have knowledge of and experience in the following technology and skill areas in order to use this guide and build the implementation described in it:

- Relational Database Management Systems (RDBMS).
- Database-driven web development.
- SOAP services, including working with WSDL files.
- Network and web application security, including the implementation of secure methods of transmitting data across the internet, such as SSL/TLS, certificates, etc.
- A computer programming language capable of building a web services interface. Examples in this guide are provided in Java.
- The specifics of how SOAP works with technologies in use in your environment, such as SOAP in Java, Ruby, PHP, C#, .NET, etc.

1.1.2 Database Assumptions

The client information databases at HSPs are assumed to have the following features in order for the data to be compatible with the database on the web services server hosted by DATIS:

- The database is a relational database (RDBMS).
- The following tables and keys are present and have a relationship:
  - A Client table with a Primary Key that uniquely identifies each client (e.g., ClientID).
  - A table for tracking admission to functional centres with a Primary Key that uniquely identifies each admission.
  - The relationship between Clients and Admissions is one-to-many (because a client can be enrolled in more than one functional centre or program).

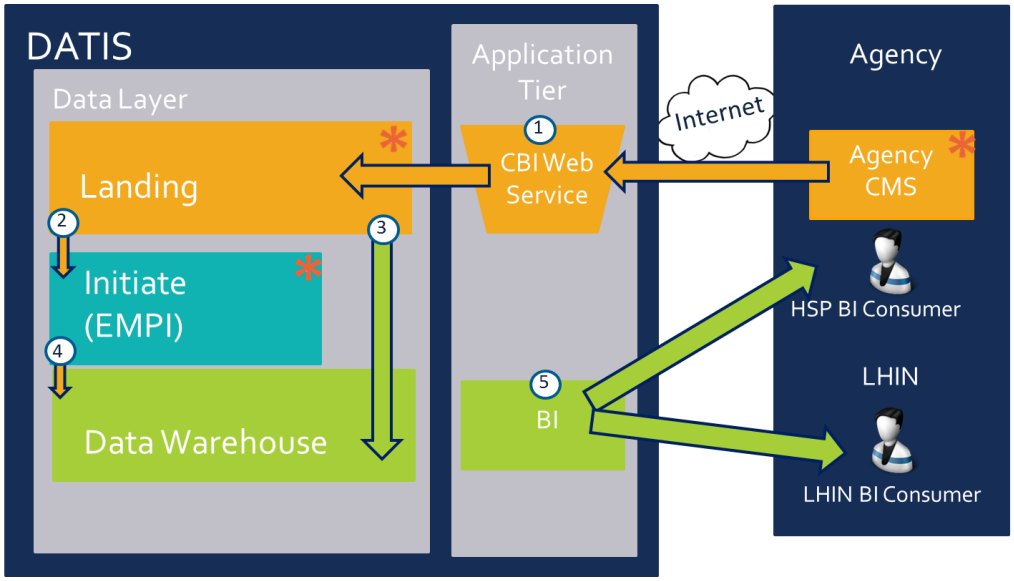
Note: If your database does not meet these requirements, please contact cbisupport@reconnect.on.ca .

1.2 Where to go for Help

If, throughout any stage of the CBI Project, you have questions, please contact cbisupport@reconnect.on.ca for assistance.

1.3 Architecture

The diagram below illustrates the data collection architecture that will be used to transfer data from HSPs to DATIS. A web services/WSDL/SOAP/XML architecture has been selected for this project.



In the left section of the diagram, labelled “HSP,” the orange items at the application and data layers (i.e., Client Management System, User Interface, and CMS DB) are elements that already exist in the HSP environment. Within this existing HSP environment, vendors will build a CBI-specific User Interface and a Web Service Client, and will extend the database schema to be able to export to the CBI XML schema. The Web Service Client is used to transfer data from the CMS DB, in XML, to the CBI WS Server which is hosted within DATIS.

In the centre of the diagram, labelled “DATIS,” the orange items (WS Server and Landing DB) have been developed for the CBI project and are in use. The CBI Project Team oversees this development as well as the CBI XML Schema and all blue, green, and purple elements (that are used for reporting and client record matching). Infrastructure needed for phases 2 and 3 are not currently in place, but will be implemented once complete.

Note: Functional Java Client code is available to Vendors on the TCII web site to facilitate their implementation and integration efforts.

Note: The CBI Web Service endpoint URL for testing is https://ws.cbiproject.ca:8443/CBIService/clientupload?wsdl. The Go Live endpoint URL is different and will be provided to vendors once HSPs are ready to go live.

1.4 Extending the Database Schema

In any system where new database records are copied elsewhere on a regular basis i t is important to have a method to differentiate between records that have already been uploaded and new ones. By adding the following fields to your database, your SQL statements will be able to determine what should be uploaded at each execution.
Add the following fields to both the table holding client data and to the table holding program or functional centre admissions in your database:

Name	Data Type	Description
MODIFIED_DATETIME	DATETIME	Date and time of the last modification to this record.
UPLOAD_DATETIME	DATETIME	Date and time of the last successful upload of this record.

1.5 Selecting Client Records for Uploading

This section provides examples, using SQL statements, of how Vendors might select clients for upload.

The first time you upload data to the web services server, any client who is currently admitted to at least one functional centre or program will be included in the upload. Depending on how many qualifying records there are in your database, this operation could take a long time, so be sure to do the initial upload at a time when other database or network operations will not be affected.

After the initial upload, clients who are currently admitted to at least one functional centre or program AND whose information has been updated since the last upload will be uploaded during subsequent sessions. The number of records in subsequent sessions should be much smaller than the initial upload, however, it is still advised that you schedule the subsequent uploads at a time when other database or network operations will not be affected.

When a record has been successfully uploaded, the UPLOAD_DATETIME field for that record will be updated with the current system date and time.

The following are SQL strings for selecting records for uploading:

Main

```
SELECT
      a.admission_date
      ,a.referral_date
      ,a.service_init_date
      ,a.discharge_date
      ,c.*
FROM
     admissions a
     JOIN client c
     ON a.client_id = c.client_id
```

WHERE Clause

```
(
     a.upload_datetime IS NULL
     AND a.admission_date IS NOT NULL
     AND a.discharge_date IS NULL
) OR (
     a.modified_datetime > a.upload_datetime
     OR a.upload_datetime IS NULL
) OR (
     c.modified_datetime > c.upload_datetime
     OR c.upload_datetime IS NULL
)
```

1.6 Deleting Client Records

There will be occasions when an enrollment needs to be deleted, such as when a client has been enrolled in a program by mistake. In this case, it is possible to delete the enrollment that was made in error.

When an enrollment is deleted, the CMS should queue a record for upload to the CBI system. The record will look exactly like the enrollment record, however, it will have the "D" flag set for the "action" field. The CBI system will use the orgID, client_id and prog_enrollment_id to refer to the original record.

If a client is deleted, the same process should occur; the record should be created and queued for upload with the "D" flag set for the "action" field. The CBI system will use the combination of OrgID and ClientID to find the correct individual.

Note: The DELETE action will result in a virtual delete, not an actual one. This means that a flag will be set on the record indicating that it has been deleted from the source CMS. As a result, the record will not be included in any reports.

1.7 Frequency of Uploads

Ideally, HSP records should be uploaded every 24 hours during non-business hours. It is recommended that the computer on which the client management system is running be connected to the internet at all times and is left running at night.

The upload mechanism should be coded as a background interface that attempts to contact the web services server at DATIS automatically. It should not require any interaction to start the process, and it should attempt to upload any records that have changed since the last upload.

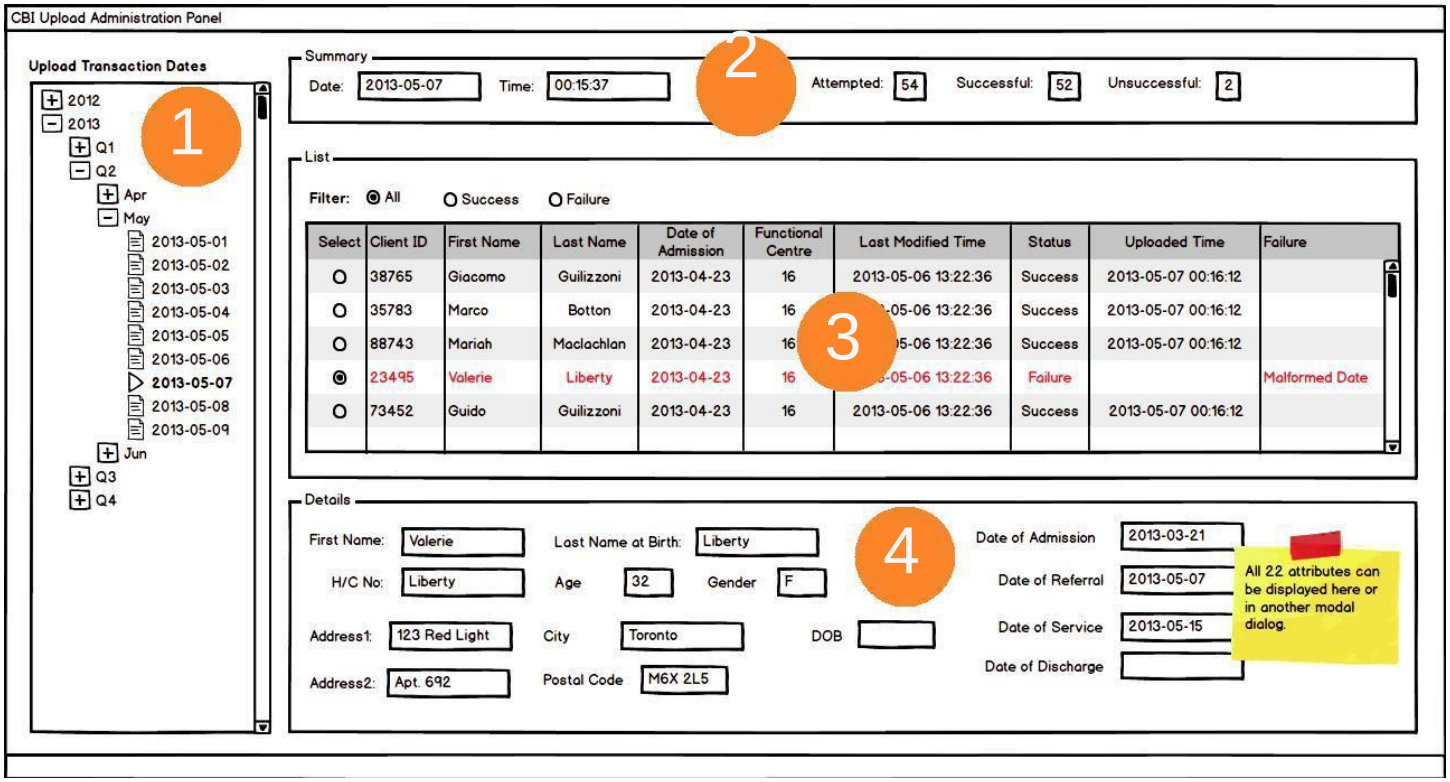
1.8 Building the Administrator Interface

The administrator interface enables you to track your uploads, monitor their success and perform the following tasks:

- View uploads by date
- View a summary of an upload
- View a list of all records, successful records or failed records included in a download
- View details of any record
- Trigger an upload manually (for example, by clicking a button)

To be built by developers hired by each Vendor, the administrator interface will enable the HSP to verify that all of their records are being successfully uploaded, to identify which records failed to upload and to see the reason for the failure and know where to foc us work on fixing faulty records.

The following diagram is an example of an administrator interface. The one developed for your HSP does not have to work or look exactly like this one, but it is recommended that you emulate the functionality in this example as much as possible.



The following is a description of each section of the diagram:

1. The Upload Transaction Dates section enables you to find an upload by the date on which it occurred. Using a tree enables administrators to find the date th ey are looking for easily. Administrators will click on the date they want to view, and its details will populate the rest of the screen.
2. The Summary section displays a summary of the activity for the upload date selected by the administrator. It provides quick, at-a-glance information about when the upload occurred and whether there were any failures in the transaction.
3. The List section displays all the records that were included in an upload. You can filter the view to see all records, only successfully uploaded records, or only failed records. Administrators can select an individual record and see its details in the

Details section of the screen. Failed records are highlighted in red and provide a reason for the failure.

The Web Service will provide three distinct pieces of information to indicate upload failures and assist with resolving them. resultCode will contain a numeric value such as 1001 or 1002 (0 indicates success). The reason field will indicate the name of the field that failed validation. The description field will contain a free-text explanation that includes two pieces of information: first, the clientId of the client whose record failed to upload will be included; second, the reason that the field named in the reason field failed validation will be explained.

For example, if a record belonging to a client with a clientId of 2343 failed to upload because the program_EnrollmentId was an empty, the resultCode would be 1002, the reason field would contain the text “program_EnrollmentId” and the description field might include the text: “ClientId: 2343. program_EnrollmentId cannot be equal to 0 or an empty string”.

The Administrative user interface should display this information to the Administrator to enable her/him to quickly identify why a record failed to upload and which client the record belonged to.

For more information on the web service response codes and response elements, see sections 2.1 and 2.2.
This enables you to identify which record to work on and what should be corrected. Once you have changed a record, for example, to correct an issue identified in the List section, it will be included in the next upload.

4. The Details section displays the information contained in the record that you select in the List section. Although this example includes the Details section on the same screen as the other sections, you could choose to display them in a pop-up or separate dialog.

In addition to these features, you may want to consider building a control panel for creating and managing the user accounts that can access this administrator interface, or integrating the administrator interface with LDAP, depending on how your environment is configured and on the requirements of IT policies in your HSP.

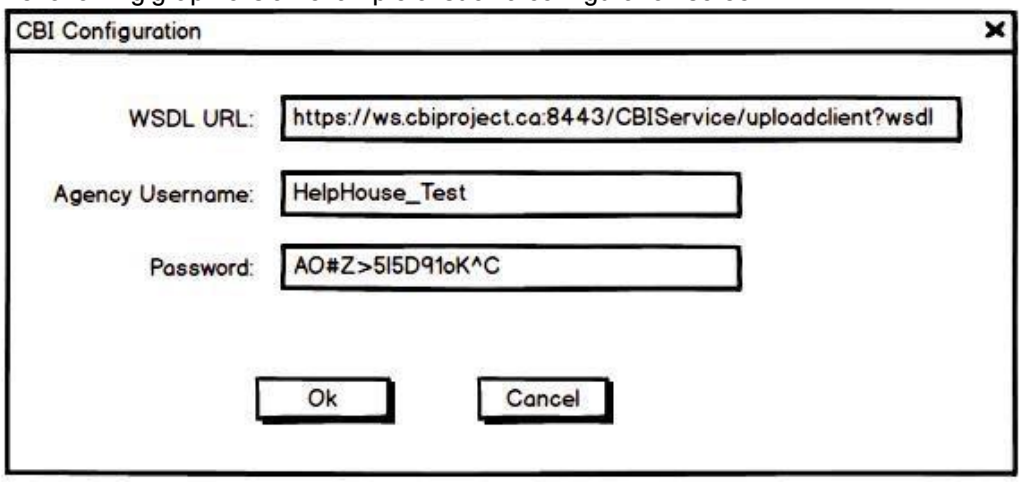
1.8.1 HSP Go-Live Configuration Screen

When you, the vendor, are finished testing the CBI implementation, you will roll the new software out to your customers (the agencies). At that time, the agencies will go through their own test phase in which they will perform a series of steps to create a fictitious client in your software and then upload the client to the CBI web service. The agency will be given credentials which will allow them to connect to the Test CBI web service. These credentials will differ from the ones given to you. Once the agency has success fully passed their test, they will be given credentials for the production environment. For this reason, the URL and credentials should not be hard-coded; they should be parameterized to read from the database or a configuration file. We recommend that the password be encrypted, however it is stored.

When preparing for the test phase and for Go-Live, the agency will be given credentials via a secure mechanism. The credentials are issued for the agency as a whole, not individual users. Once received, the agency will need to perform a one-time configuration

in which they enter their credentials and ensure that the software is pointing to the correct web service URL. This configuration step will likely be performed by an IT administrator, but it should be easy enough for a front-line worker to execute, should the need arise. We recommend that the software have a screen that allows an agency user to enter the username, password, and the correct URL. We feel this will be easier to configure than changing a configuration file somewhere, and will lessen the administrative burden on your company as the CBI-enabled software is rolled-out to agencies.

The following graphic is an example of such a configuration screen:



2 Operation

This section describes how to operate and test the application programming interface hosted on the CBI Web Services server.

2.1 Operating the API

The CBI Web Service defines a single processCBIItem operation with the following parameters:

- input: cbiDataItem
- output: cbiServiceResult

These parameters are defined by the XML schema (cbiservices.xsd).

cbiDataItem is defined as a complex data type and has the following elements (bold text indicates that the element is mandatory):

Element	Data Type	Constraints	Description	Maintains History
orgId	Number(8)	Mandatory; > 0	The master number issued by the MOHLTC.	No. Used to identify the record
clientId	varchar(256)	Mandatory; if numeric, > 0; if string, not empty	The value that uniquely identifies a client within a software and agency.	No. Used to identify the record
healthcardNo	varchar(32)		The OHIP health card number.	No. Treated as a correction
Firstname	varchar(256)		First name provided by the client.	Yes
lastnameAtBirth	varchar(256)		Last name on the client's birth certificate.	Yes
lastnameCurrent	varchar(256)		Last name provided by the client.	Yes
middleNames	varchar(256)		Middle name(s) provided by the client.	Yes
dateOfBirth	number(19)		The date of birth that appears on the client's birth certificate.	No. Treated as a correction
Age	number(8)		The actual or estimated age provided by a client if their date of birth is not available. Note: If you collect the date of birth, you do not need to collect the age as well.	No. Treated as a correction
Gender	varchar(256)		The gender selected by the client.	Yes
address1	varchar(256)		The address provided by the client.	Yes
address2	varchar(256)		The address provided by the client.	Yes
City	varchar(256)		The city provided by the client.	Yes
postalCode	varchar(7)		The Canadian postal code f or the address provided by the	Yes
lhin_OfResidence	varchar(256)		The LHIN in which the client resides.	Yes
phone	varchar(64)		The primary phone number provided by the client	Yes

Element	Data Type	Constraints	Description	Maintains History
program_EnrollmentId	varchar(256)	Mandatory; if numeric, > 0; if a string, not empty	Program_EnrollmentId uniquely identifies a client's enrollment in a program at the agency level. For example, HSP Inc. has a program called Community Support in the functional centre 72 5 09 30 (Com Case Management). When fictional client John Doe is enrolled in the Community Support program, the CMS creates a record in the table that records program enrollments. The purpose of this table is to be able to know what programs a client is enrolled in and how many clients are enrolled in each program at the agency. The primary key for this table should be equivalent to and should populate the program_EnrollmentId.	No. Used to identify the record
fc_Id	Varchar(32)	Mandatory; must match an item from the list of functional centres published by the MOHLTC	The functional centre in which the program is registered and reported on, e.g., 72 5 09 30.	No. Used to identify the record
fc_referralDate	number(19)		The date on which the HSP becomes aware a client's referral to a functional centre for service.	No. Treated as a correction
fc_admissionDate	number(19)	Valid date; ≤ date of upload of record	The date on which the HSP registers a client into a functional centre. Cannot be future dated and must be a valid date.	No. Treated as a correction
fc_serviceInitDate	number(19)		The date on which service delivery in a functional centre starts or started.	No. Treated as a correction
fc_dischargeDate	number(19)		The date on which the HSP deregisters a client from a functional centre.	No. Treated as a correction
action	character(1)	Mandatory: Must equal to 'I' or 'D'	Options: I – insert D - delete	n/a

Note: Functional Java Client code is available to Vendors on the TCII web site to facilitate their implementation and integration efforts.

Note: The CBI Web Service endpoint URL for testing is https://ws.cbiproject.ca:8443/CBIService/clientupload?wsdl. The Go Live endpoint URL is different and will be provided to vendors once HSPs are ready to go live.

2.2 Results and Status Codes

CBI handles HTTP specific status codes in the way prescribed in WS-I Basic Profile 1.1 Second Edition (http://www.ws-i.org/Profiles/BasicProfile-1.1.html#SOAPHTTP), section 3.4.

The cbiServiceResult operation is defined as a complex data type and has the following elements (bold text indicates that the element is mandatory):

Element	Type	Function	Returns
resultCode	Integer	Indicates whether the operation executed successfully or not.	◆ 0 (zero) - executed successfully ◆ <>0 (zero) - errors occurred
reason	String	Contains the name of the data element that caused the error.	<>0 (zero)
reason	Integer	Contains the internal ID of a new record created in the CBI database.	==0 (zero)
Description		Contains a description of the cause of the error.	text

The following HTTP response status codes can be returned by the web services server to a web service client:

Code	Description
200	OK
202	Accepted
400	Bad Request
405	Method Not Allowed
415	Unsupported Media Type
500	Internal Server Error

The following custom response codes are used by the system to indicate success or failure and provide diagnostic information:

Code (resultCode)	Description
0	No errors encountered
1001	Invalid credentials were used
1002	The credentials were valid, but a mandatory element was missing or invalid (see section 2.1 Operating the API for more information on mandatory fields and valid formats)

In the event of a 1002 resultCode error, the web service will return information that may be helpful in diagnosing problems: reason will indicate the name of the field that failed validation, and description will contain a free-text explanation that indicates both the clientId of the client whose record failed to upload and information on why the element named in the reason field failed validation.

This information is helpful both to vendors when testing their products and to HSPs when investigating why certain client records failed to upload. Therefore, the software should display the resultCode, reason and description fields in the Administrative user interface. See section 1.8 for more information.

2.3 Security Requirements

The CBI Web Service is WS-Security compliant. The authentication credentials (user name and password) are sent in the HTTP header. Communication between each web service client and the web service provider will be encrypted using transport-level encryption (SSL) over HTTPS and certificates to prevent authentication credentials from being compromised during transmission.

2.3.1 Downloading and Installing a Certificate

The following instructions use Java as an example. If you are using something other than Java, please consult your product's documentation for instructions on installing certificates.

1. On the computer that will run the CBI client, create a working folder for this procedure.
2. If you are using Java, you can download the Java certificate from the TCII web site:
   - http://tcii.reconnect.on.ca/assets/InstallCerts.jar
3. Verify the presence of the jssecacerts file in the `$JAVA_HOME/jre/lib/security/` folder (where "`$JAVA_HOME`" is the root folder of the Java installation). Do one of the following:
   - If it is present, it may mean that a production certificate was added after a test certificate. In this case, copy this jssecacerts file into the working folder that you created in Step 1. This causes the command that you will run in the next step to add a new certificate into the existing key store file instead of overwriting it.
   - If it is not present, continue with the next step of the procedure.
4. In the working folder that you created in Step 1, run the following command:

   ```
   java -classpath InstallCerts.jar com.aw.ad.util.InstallCert
   ws.cbiproject.ca:8443
   ```

   Note: You may need to run this command twice; the first attempt may produce exceptions, the second attempt should not.

   The jssecacerts file is created.

   Note: A different command will be provided to Vendors once HSPs are ready to go live.

5. Open the `$JAVA_HOME/jre/lib/security/` folder (where "`$JAVA_HOME`" is the root folder of the Java installation) and search for an earlier copy of the jssecacerts file:
   - If you find a copy, archive it elsewhere and continue with the next step.
   - If there is no copy found, continue with the next step.
6. Copy the version of the jssecacerts file that was created in Step 4 to the `$JAVA_HOME/jre/lib/security/` folder.

2.3.2 Testing the Certificate

To test that your certificate was installed correctly, run a sample client request with random credentials. This request would generate the following response:

```
1   <S:Envelope
    xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
2      <S:Body>
3         <ns2:processCBIItemResponse
          xmlns:ns2="http://ws.cbi.com/">
4            <return>
5               <resultCode>1002</resultCode>
6               <reason>clientId</reason>
7               <description>Invalid  data</description>
8            </return>
9         </ns2:processCBIItemResponse>
10     </S:Body>
11  </S:Envelope>
```

The <resultCode> tag stores the result code of the test. The following result codes may appear:

- 1001 - invalid credentials were used
- successful - the test was successful; the credentials were valid
- 1002 - the credentials were valid, but a mandatory element is missing, or invalid. See section 2.1 for more information on mandatory fields and valid formats.

2.4 Sample Code

This section is an example of how web service clients can be generated. It is intended for Vendors' developers and is for information purposes only. Although this example was created using the Java programming language, Vendors may program their implementation in any programming language they wish, however, it must meet the specifications described in this guide.

The SubmissionService client web service stub was built and tested with the following technologies:

- JDK 1.6.x (http://java.sun.com/javase/downloads/widget/jdk6.jsp)
- JAX-WS (Metro)
- WSDL file (cbiservices.wsdl)
- XSD file (cbiservices.xsd)
- Apache Tomcat 7.0.37
- Python 2.7.1 and Suds library version 0.4 GA

Note: Functional Java Client code is available to Vendors on the TCII web site to facilitate their implementation and integration efforts.

Note: The CBI Web Service endpoint URL for testing is https://ws.cbiproject.ca:8443/CBIService/clientupload?wsdl. The Go Live endpoint URL is different and will be provided to vendors once HSPs are ready to go live.

2.4.1 Creating the Client Stub Code

1. From a DOS or Unix command line, execute the following command:
   - wsimport <WSDL_URL>

   In this example, <WSDL_URL> is the URL where the CBI WSDL will be deployed. It will be provided to vendors when they are ready to begin their implementation.
2. Verify that the wsimport utility is present in the same folder as the Java JDK binaries (java, javac, etc.). If it is not present, move or install it to the Java JDK binaries folder.
3. Create the following folder structure and class file:
   - com/cbi/ws/*.class

2.4.2 Customizing the Client Stub Code for Security Purposes

Because the wsimport utility does not incorporate WS-Security out of the box, some code modifications are necessary to incorporate WS-Security into the utility. For an example of securing your code, see the CBIServiceClient.java file, lines 23 to 94, in http://tcii.reconnect.on.ca/assets/CBIClientF.zip.

2.5 Handling Date Input Fields in the Java Client

Some of the methods accept dates as input parameters, such as setDateOfBirth, setFcAdmissionDate, etc. The WSDL and the wsimport utility handle date elements , such as xs:dateTime, using the XMLGregorianCalendar input type. If your database uses a different date format, then you must convert those dates into the XMLGregorianCalendar input type in order to submit your data to the CBI Web Service API.

The following code snippets demonstrate how to convert text in the "dd/MM/yyyy HH:mm:ss" format into the XMLGregorianCalendar object:

```
1   private static DatatypeFactory df = null;
2   static {
3      try {
4         df = DatatypeFactory.newInstance();
5      } catch (DatatypeConfigurationException dce) {
6         throw new IllegalStateException(
7            "Exception     while     obtaining     DatatypeFactory
            instance", dce);
8      }
9   }
```

```
1   private static XMLGregorianCalendar asXMLGregorianCalendar(String
    fromDate)
2   {
3         SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy
          HH:mm:ss");
4
5      Date date = null;
6
7      try { date = sdf.parse(fromDate); } catch (Exception e) {
       e.printStackTrace(); }
8
9      if (date == null) {
10        return null;
11     } else {
12        GregorianCalendar gc = new GregorianCalendar();
13        gc.setTimeInMillis(date.getTime());
14        return df.newXMLGregorianCalendar(gc);
15     }
16  }
```

The following code is an example of how to use the conversion:

```
1   CbiDataItem cbiRec = new CbiDataItem();
2
3   ...
4
5   cbiRec.setDateOfBirth(asXMLGregorianCalendar("17/09/1980
    00:00:00"));
```

2.6 Testing

2.6.1 Testing Objectives

The following is a list of objectives for testing your solution:

- Uploads should include all qualifying records and no other records.
- Data that appear in the destination database should match the data found in the source database, except for converted dates and times.
- Dates and times should be converted into the Gregorian format in the destination database if the date format in the source database is different.
- XML should be valid and well-formed, and should comply with the XML schema.
- Transmission of upload data should be encrypted at all times.

2.6.2 Test Plan Phases

Testing of the CBI solutions will be done in phases. Documents, data and recommendations for specific tests will be released as they become available. Individual elements that may be tested may change as circumstances require.

2.6.2.1 Phase 1: Vendor Testing

2.6.2.1.1 Design-Time

Using the CBI Test File (http://tcii.reconnect.on.ca/community-business-intelligence-/implementation-/vendor-implementation/), vendors will test that:

- Their certificate is installed correctly
- They can authenticate successfully (i.e., their credentials work)
- They can successfully upload a test record, which implies the following:
  - The web service is being called properly
  - XML is well-formed
  - All other design-time requirements have been met
  - Can connect to the web service

Meets all requirements listed in the User Implementation Guide To perform the test, do the following:

1. Download the test data file from the following location:
   - http://tcii.reconnect.on.ca/assets/CBI-Test-Client-File.csv
2. Import the test data file into your testing database.
3. Upload the records to the web service.
4. Verify that the records were successfully uploaded.

2.6.2.2 Phase 2: HSP Testing

Using the HSP Testing and Validation Guide (http://tcii.reconnect.on.ca/assets/Full-Roll-Out-HSP-Testing-and-Validation-Guide-1.04.pdf), each HSP will perform tests on a non-production version of their CMS. HSPs will request test credentials, which will be different from the credentials requested by the vendor in the phase 1 tests. The tests consist of a series of alternating edit and send steps; the user is to edit test clients in the CMS and trigger uploads of the changed records. These changes are intended to mimic changes that would normally occur to a record over time.

2.6.3 Testing Connectivity to the Web Service

In order to verify that the client -side stub code has been configured correctly, the web service must be deployed on a server that is accessible to the client stub. For local testing, an application server such as Tomcat is suitable. The server-side web service must be deployed as a standard Java web application archive file (*.WAR). After the
*.WAR file has been deployed, the web service will be active and will listen for incoming client requests.
For an example request and response in raw XML obtained using the SoapUI client tool, see the files cbi_request_1.xml and cbi_response_1.xml in http://tcii.reconnect.on.ca/community-business-intelligence-/implementation-/vendor-imple_mentation/.

3 Code Samples

This section provides code samples for your reference.

3.1 Python Sample Code

To facilitate the Vendors' implementations, and to test the interoperability with something other than Java clients, the code has also been written in the Python Suds library. This library was selected because it offers better handling of WSDL generated by newer SOAP tools, such as JAX-WS. The library can be downloaded from https://fedorahosted.org/suds/. Functional Python Client code is available to Vendors at http://tcii.reconnect.on.ca/community-business-intelligence-/implementation-/vendor-imple_mentation/.
The code, when executed from the command line, produces the following output:

```
1  cbi-py$ python cbi-client.py
2
3  Suds  (  https://fedorahosted.org/suds/  )  version:  0.4  GA
   build: R699-20100913
4
5  Service ( CBIServiceImplService ) tns="http://ws.cbi.com/"
6     Prefixes (1)
7        ns0 = "http://ws.cbi.com/"
8     Ports (1):
9        (CBIServiceImplPort)
10          Methods (1):
      11       processCBIItem(cbiDataItem arg0, )
      12    Types (2):
      13       cbiDataItem
      14       cbiServiceResult
      15
16 Suds ( https://fedorahosted.org/suds/ ) version: 0.4 GA build:
   R699-20100913
17
18 Service ( CBIServiceImplService ) tns="http://ws.cbi.com/"
19    Prefixes (1)
20       ns0 = "http://ws.cbi.com/"
21    Ports (1):
22       (CBIServiceImplPort)
23          Methods (1):
      24       processCBIItem(cbiDataItem arg0, )
      25    Types (2):
      26       cbiDataItem
      27       cbiServiceResult
      28
      29
30 (cbiServiceResult){
31    resultCode = 0
32    reason = "12"
33    description = "Data processed successfully"
34 }
```

3.2 The WSDL File Sample Code

The WSDL file sample code is available at the following URL:

- https://ws.cbiproject.ca:8443/CBIService/clientupload?wsdl

3.3 The XSD File Sample Code

The XSD file sample code is available at the following URL:

- https://ws.cbiproject.ca:8443/CBIService/clientupload?xsd

3.4 Sample XML Request and Response Code

See the files cbi_request_2.xml and cbi_response_2.xml in http://tcii.reconnect.on.ca/community-business-intelligence-/implementation-/vendor-imple_mentation/ for sample XML request and response code.

Index


reconnect
COMMUNITY BUSINESS INTELLIGENCE