Taleo Enterprise

Taleo Web Services User Guide

Feature Pack 12B
September, 2012

Confidential Information and Notices

Confidential Information

The recipient of this document (hereafter referred to as "the recipient") agrees that the Confidential Information disclosed herein by Taleo shall be retained in confidence by the recipient, and its respective employees, affiliates and/or subsidiaries.

For the purpose herein, the term "Confidential Information" shall mean the following:

1. Any information, know-how, data, process, technique, design, drawing, program, formula or test data, work in process, business plan, sales, suppliers, customer, employee, investor or business information contained in a document, whether in written, graphic, or electronic form; or
2. Any document, diagram, or drawing which is either conspicuously marked as "Confidential", known or reasonably known by the other party to be confidential, or is of a proprietary nature, and is learned or disclosed in the course of discussions, demonstrations, or other collaboration undertaken between the parties.

Limited Rights Notice (Dec 2007)

1. These data are submitted with limited rights under Subcontract No. 6896589. These data may be reproduced and used by the Government with the express limitation that they will not, without written permission of the Contractor, be used for purposes of manufacture nor disclosed outside the Government; except that the Government may disclose these data outside the Government for the following purposes, if any; provided that the Government makes such disclosure subject to prohibition against further use and disclosure: None.
2. This notice shall be marked on any reproduction of these data, in whole or in part.

Table of Contents

Getting Started

The Taleo API

Taleo provides programming access to your organization's information using a simple, powerful, and secure application programming interface, the Taleo Web Service API (the API).

To use this document, you should have a basic familiarity with software development, Web services, and the Taleo user interface. Knowledge of the Taleo Connect Client is not required, but would greatly help to understand and visualize the Data Model made available through the Taleo API.

The API consists of a set of callable methods, and some API endpoints. Its documentation is divided in two parts:

- The first part (this document) describes the purposes of the API, its Standard Compliance, and how to use it using common Development Platforms. It further describes basics about Web services Calls and Standard Taleo Data Types, and covers Error Handling, Security and Limits applying to any Web service of the API.
- The second part consists of several documents, referred to as Taleo data dictionaries. Each data dictionary applies to one specific Taleo product and lists a set of callable methods specifically made available for that Product. A data dictionary further describes the Data Model applying to the associated Product, making reference to its Entities, Fields and Relations.

Integrate And Extend Taleo Solutions

Speed and agility are the keys to success in the highly competitive market for top talent. Integration between your talent management solution and your extended network of service providers is critical for streamlined processes and high quality hires.

The Taleo API allows you to integrate and extend Taleo Solutions using the language and platform of your choice:

- **Integrate** Taleo with your organization: The Taleo API enables seamless transfer between Taleo Enterprise Edition, data warehouses, backend human resources information system (HRIS), and financial systems such as Oracle, PeopleSoft, JD Edwards, SAP, Lawson, and others.
- **Extend** Taleo Solutions: The Taleo API helps you to extend your talent management processes to external partners, eliminating manual steps in your process, costly process delays and errors, and the headaches of typical integration projects.

For more information about Taleo Solutions, visit http://www.taleo.com/solutions or contact your customer representative.

Standard Compliance

The API is implemented to comply with the following specifications:

Standard Name	Website
Simple Object Access Protocol (SOAP) 1.1	http://www.w3.org/TR/2000/NOTE-SOAP-20000508
Web Service Description Language (WSDL) 1.1	http://www.w3.org/TR/2001/NOTE-wsdl-20010315
Hypertext Transfer Protocol (HTTP) 1.1	http://www.w3.org/Protocols/rfc2616/rfc2616.html
WS-I Basic Profile 1.1	http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

Standard Name	Website
Web Services Security 1.1	http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
WS-Security SAML Token Profile 1.1 #	http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf
WS-Security Username Token Profile 1.1 #	http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf

There are many different styles of SOAP messages; the two most common today are **rpc/encoded** and **document/literal**. The Taleo API only supports the **document/literal** style, as rpc/encoded style is not endorsed by WS-I Basic Profile 1.1 and was removed in the SOAP 1.2 specifications. Technically, using the **document/literal** style means that a SOAP Body of a Web service request will be a complex message document that must conform to a specific XML schema (included in the WSDL of the Web service).

Development Platforms

The API has already been successfully tested against the following Development Platforms:

Development Platform	Website
AXIS2 v1.3 using XmlBeans and ADB Data Binding	http://ws.apache.org/axis2/
C# using .NET Framework 3.0 SDK	http://msdn2.microsoft.com/en-us/netframework/default.aspx
XFire 1.2.6 using XmlBeans Data Binding	http://xfire.codehaus.org/

If your Platform is not listed above, this means it has not yet been tested. Assuming this one is compliant with our supported Standard Compliance, you should be able to access and use the API successfully.

API Support Policy

Taleo recommends that your new client applications use the most recent version of the WSDL file to fully exploit the benefits of richer features and greater efficiency. When a new version is released, use the following steps in the Quick Start to update your WSDL:

- Regenerate the WSDL file (see Step 3: Generate or Obtain the Web Services WSDL Files
- Import it into your environment (see Step 4: Import the WSDL File Into Your Development Platform

Backward Compatibility

Taleo strives to make backward compatibility easy when using its API. Each Taleo product is associated to a data dictionary, defining its product specific API. Each data dictionary is bound to a specific mapping version. Backward compatibility support differs between minor versions (i.e. 7.5 SP1, 7.5 SP2) and major versions (7.0, 7.5, 8.0).

We maintain support for each minor version of a product. We also maintain support for the last major version preceding the current version. The API is backward compatible in that an application created

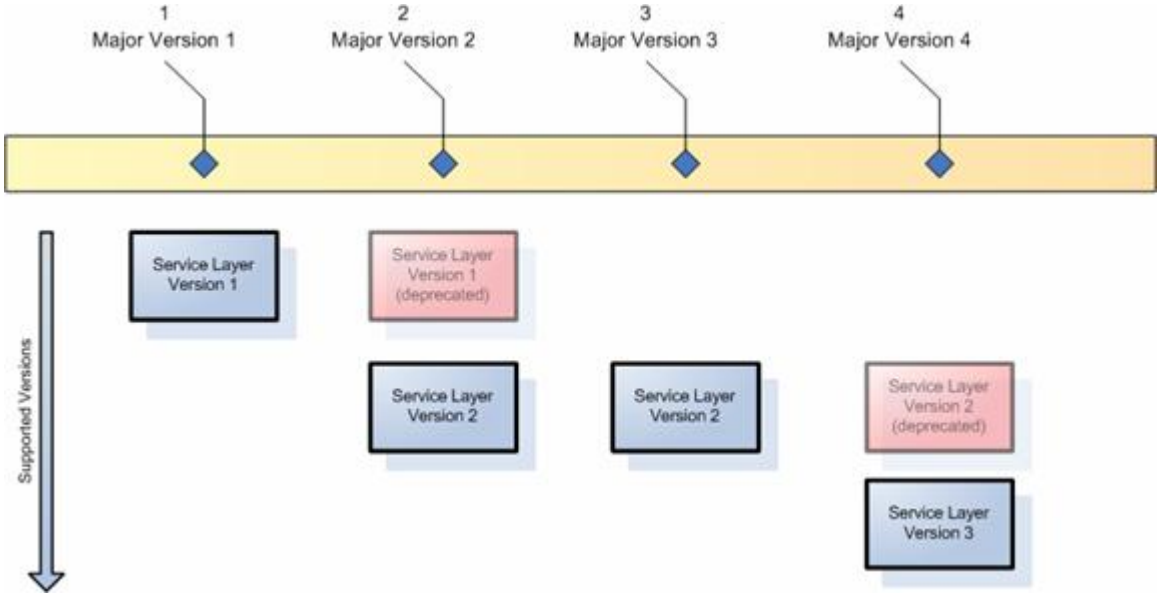
to work with a given data dictionary mapping version will work with that same mapping version in future minor versions and the next future major version of the product.

Taleo does not guarantee that an application written against one API version will work with future API versions: Changes in method signatures and data representations are often required as we continue to enhance the API. However, we strive to keep the API consistent from version to version with minimal, if any changes, required to port applications to newer API versions.

For example, an application written using the Taleo Professional 7.5 API shipped with the Taleo Enterprise Edition 7.5 release will continue to work with all future minor versions (i.e. Taleo Enterprise Edition 7.5 SP1), and with the next major version of the product (i.e. Taleo Enterprise Edition 8.0), using that same API. However, the same application may not work with later versions of the product (i.e. Taleo Enterprise Edition 8.5) without modifications to the application, using the latest available product API (i.e. Taleo Professional 8.5 API).

API End of Life

Taleo is committed to supporting each API version for a minimum of two (2) major versions from the version of first release. In order to mature and improve the quality and performance of the API, versions that were introduced more than one major version before the current version may cease to be supported.



For example, in the figure above, a major version is released, with a given version 1 of a service layer (Web Service API). A new major version is released, and the service layer also releases a new version. The former version is now supported, but deprecated. Then, a new major version is released, but no new service layer version is released. Support for version 1 of the service layer ends, and now only one version is supported. Then, a new major version is released, with a new service layer version. The version 2 is still supported, but deprecated.

Taleo Web Services Namespaces

Various namespaces are used inside a Taleo Web Service WSDL and SOAP document. These namespaces define specific parts of the document and are also used for versioning purposes. Taleo Web Services can be defined into two categories of services: import and export services.

Taleo 10 Namespaces

Namespace	Description
http://www.taleo.com/ws/integration/toolkit/[version]	The integration toolkit namespace. This namespace is used to define elements related to the integration toolkit framework. The integration toolkit framework is where the Taleo component web service infrastructure resides. It is used in the definition of elements such as the WebServiceFault.
http://www.taleo.com/ws/[productNickname][nsVersion]/[service]	The service namespace. This namespace is used to define elements related to the web service itself. It is used in the definition of elements such the operation parameters data types. It is the link between the service and the Taleo data model. Its name is composed of the product nickname (i.e.: tee for Taleo Enterprise, so for Smartorg, etc.) followed by the namespace version (i.e. 2009/01) and the service name (i.e. for the DepartmentService, it will be department). Example: WebService: DepartmentService Namespace: http://www.taleo.com/ws/tee800/2009/01/department
http://www.taleo.com/ws/[productNickname][nsVersion]/import	The product import data model. This namespace is used to define elements related to the integration data model used to define all the entities that can be used in import services. It is used in the definition of elements such as the User, the Candidate, the Requisition and all the other Taleo entities. Its name is composed of the product nickname (i.e.: tee for Taleo Enterprise, so for Smartorg, etc.) followed by the namespace version (i.e. 2009/01) and the import string. Example: Namespace: http://www.taleo.com/ws/tee800/2009/01/import
http://www.taleo.com/ws/[productNickname][nsVersion]	The product export data model. This namespace is used to define elements related to the integration data model used to define all the entities that can be used in export services.

Namespace	Description
	It is used in the definition of elements such as the User, the Candidate, the Requisition and all the other Taleo entities. Its name is composed of the product nickname (i.e.: tee for Taleo Enterprise, so for Smartorg, etc.) followed by the namespace version (i.e. 2009/01). For technical reasons, this namespace is not ended by the export string (as opposed to the import one). Example: Namespace: http://www.taleo.com/ws/tee800/2009/01

Version 7.5 Namespaces

Namespace	Description
http://www.taleo.com/ws/integration/toolkit/[version]	The integration toolkit namespace. This namespace is used to define elements related to the integration toolkit framework. This is the Taleo component where the web service infrastructure resides. It is used in the definition of elements such as the WebServiceFault.
http://www.taleo.com/ws/[productNickname][nsVersion]	The product import and export data model. This namespace is used to define elements related to the integration data model used to define all the entities that can be used in import and export services. It is used in the definition of elements such as the User, the Candidate, the Requisition and all the other Taleo entities. Its name is composed of the product nickname (i.e.: art for Active Recruiting Technology, so for Smartorg, etc.) followed by the namespace version (i.e. 2006/12). Example: Namespace: http://www.taleo.com/ws/art750/2006/12

Namespace Limitations

The namespace strategy has evolved between Taleo 7.5 and Taleo 10 versions. The 7.5 version is missing some concepts to really make each piece of information independent of each other. This will be described in greater detailed in the Taleo 7.5 Namespaces section.

Using Web Services

For each specific Taleo product, a list of WSDL files is available. Any number of commercial or open source tools can then be used to create clients that access these services. The soapUI project (www.soapui.org) offers a free open source no-frills yet complete user interface to create and test web service calls. Other commercial solutions offer more advanced features: XML Spy (www.altova.com), Stylus Studio (www.stylusstudio.com) and oXygen (www.oxygenxml.com). In order to embed web service calls within an application, the Apache Web Service Axis2 project (ws.apache.org/axis2) offers a WSDL2Java tool that generates the proper source code for a Java based project. Microsoft .NET also offers a web service toolkit for its development framework. The Quick Start section provides a step by step procedure using these toolkits.

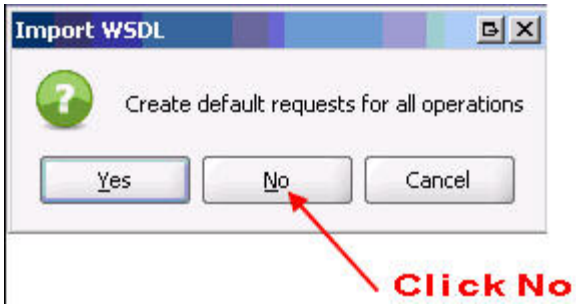
Multiple books and articles are available that describe in detail how to interact with the web services, SOAP, and WSDL standards. Some interesting starting points are:

- http://www.w3.org/2002/ws (standards and links)
- http://www.webservices.org (Vendor-neutral Web Services industry portal)
- http://java.sun.com/webservices (Sun's Java web services portal)
- http://msdn.microsoft.com/webservices (Microsoft's .NET web services portal)

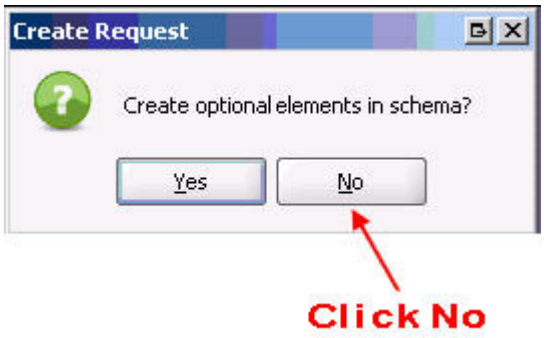
Using SOAP UI with Taleo WSDL

There are constraints that SOAP UI users must be aware of when using Taleo WSDL to generate a test suite and/or a test request.

When creating a new WSDL project and adding a Taleo WSDL, DO NOT "Create default requests for all operations".



When creating a new test request, DO NOT "Create optional elements in schema."



Many Taleo operations use entity type parameters that are composed of base type entities. These base type entities can be specialized, and sometimes must be, to pass the correct data. To correctly support that characteristic, each base type in the WSDL is composed of a list of elements from all its specialized types. For detailed information about how to work with base types, refer to section Operations On Parameters With Base Type Elements.

Quick Start

You will need a User Account with "Web Service" permissions to be able to access and use the Taleo API. If you do not have such an account, contact your System Administrator to request one.

The following steps will create a sample application in your development environment:

Step 1: Have the Taleo Production Team Enable the Web Service Framework

By default, the web service framework is disabled. The Taleo Production Team will activate it for you upon a proof of purchase. Contact your Customer Representative for further details about pricing and activation.

Step 2: Obtain and Activate a Taleo User Account with Web Services Permissions

To access the Taleo API you need to have a Taleo User Account activated with Web Service Permissions. While developing, staging, and testing your application, we strongly recommend to use a dedicated testing or staging application to test your application against sample data instead of your organization's live data. This is especially true for applications that will be inserting, updating, or deleting data (as opposed to simply reading data). Your System Administrator will provide you with a login username and password for your product environment.

Step 3: Generate or Obtain the Web Services WSDL Files

To access the Taleo API, you need the Web Service Description Language (WSDL) files corresponding to the Web Services. A WSDL file defines a Web service that is available to you. Your development platform uses this WSDL to generate an API to access the Web service it defines. Each Web service available through the Taleo product is defined by a dedicated WSDL file. You can either obtain the WSDL files from your Customer Representative or you can generate them yourself if you have access to the WSDL download page in the Taleo product user interface. For more information about WSDL, see http://www.w3.org/TR/wsdl

Generating the WSDL File for Your Organization

Any user with "Web Service" permission can download the Web Service Description Language (WSDL) file to integrate and extend Taleo using the API. Any user with the "System Integrator" role has this permission.

The WSDL file is dynamically generated based on which Taleo product (i.e. Taleo Enterprise Edition - Professional) you download. The generated WSDL defines all API calls, objects (including standard and common objects), and fields that are available for API access for your organization.

To generate the WSDL file for your organization:

- Log in to your account using the URL specified in the data dictionary corresponding to your Taleo product (i.e. for Taleo Professional, log in to *https://hostname.taleo.net/servlets/soap*). You must log in as an administrator or as a user who has the "Web Service" permission.
- You should see a list of Web services available for this product. If the Web service you are looking for is not in the list, you may not have enough privileges to access it, you may be using the wrong URL for the Product, or you are searching for a deprecated Web service that has been removed or replaced by another one since the last major version.
- Right-click the Web service name to display your browser's save options, and save the WSDL to a local directory.

Note: If a new version of the data dictionary (Product API) is released, you will need to regenerate the WSDL file in order to access the newest call and type definitions.

Step 4: Import the WSDL Files Into Your Development Platform

Once you have the WSDL file, you need to import it into your development platform so that your development environment can generate the necessary objects for use in building client Web service applications in that environment. This section provides sample instructions for Apache Axis and Microsoft Visual Studio. For instructions about other development platforms, see your platform's product documentation.

Instructions for Java Environments (Apache Axis)

Java environments access the API through Java objects that serve as proxies for their server-side counterparts. Before using the API, you must first generate these objects from your Web service's WSDL file. If you are using more than one Web service in your application, you must generate these objects from each WSDL file.

Each SOAP Java client has its own tool for this process. For Apache Axis2, use the WSDL2Java utility.

Note: Before you run WSDL2Java, you must have Axis2 installed on your system.

The basic syntax for WSDL2Java from the Axis2 InstallPath/bin is:

```
wsdl2java.bat -uri pathToWsdl/WsdlFilename -d xmlbeans -ns2p
 namespaceURL=javaPackageName
```

The `-d` specifies the Databinding framework; here xmlbeans (http://xmlbeans.apache.org) is used. The `-ns2p` specifies a comma separated list of namespaces and packages where the given package will be used in the place of the auto generated package for the relevant namespace. For more information, see the WSDL2Java documentation.

Taleo strongly recommends to always specify a different target package name for each WSDL file (or Web service) because different WSDL files may refer to the same data type name although using different data type definition. Specifying different Java package names will prevent Java class name collisions when more than one Taleo Web service is used within the same application.

For example, if the Axis JAR files are installed in `C:\axis2-1_3`, and the WSDL is named `CandidateService.wsdl` and is stored in `C:\mywsdls`, and you want to map the Web service mapping version `http://www.taleo.com/ws/art750/2006/12` to a specific `com.taleo.art750.candidate` package, you would invoke:

```
C:\axis2-1.3\bin\wsdl2java.bat -uri C:\mywsdls
\CandidateService.wsdl -d xmlbeans -ns2p http://www.taleo.com/ws/
art750/2006/12=com.taleo.art750.candidate,http://www.taleo.com/ws/
integration/toolkit/2005/07=com.taleo.itk
```

This command will generate a set of folders and Java source code files in the same directory in which it was run. After these files are compiled, they can be included in your Java programs for use in creating client applications.

For most Java development environments, you can use wizard-based tools for this process instead of the command line. For more information about using WSDL2Java, see http://ws.apache.org/axis/java/reference.html

Instructions for Microsoft Visual Studio

Visual Studio languages access the API through objects that serve as proxies for their server-side counterparts. Before using the API, you must first generate these objects from your Web service's WSDL file. If you are using more than one Web service in your application, you must generate these objects from each WSDL file.

Visual Studio provides two approaches for importing a WSDL file and generating an XML Web service client: an IDE-based approach and a command line approach.

Note: Before you begin, you must create a new application or open an existing application in Visual Studio. In addition, you need to have generated the WSDL file(s), as described in Step 3: Generate or Obtain the Web Services WSDL Files.

A Visual Studio XML Web service client is any component or application that references and uses an XML Web service. This does not necessarily need to be a client-based application. In fact, in many cases, your XML Web service clients might be other Web applications, such as Web Forms or even other XML Web services. When accessing XML Web services in managed code, a proxy class and the .NET Framework handle all of the infrastructure coding.

To access an XML Web service from managed code:

1. Add a Web reference to your project for the XML Web service that you want to access. The Web reference creates a proxy class with methods that serve as proxies for each exposed method of the XML Web service.
2. Add the namespace for the Web reference.
3. Create an instance of the proxy class and then access the methods of that class as you would the methods of any other class.

To add a Web reference:

1. On the Project menu, choose *Add Web Reference*.
2. In the URL box of the Add Web Reference dialog box, type the URL to obtain the service description of the XML Web service you want to access, such as: `file:///c:\WSDLFiles\CandidateService.wsdl` or `https://hostname/servlets/soap?ServiceName=CandidateService&wsdl`.
3. Click *Go* to retrieve information about the XML Web service.
4. In the Web reference name box, rename the Web reference, such as *taleo.candidatesvc*, which is the namespace you will use for this Web reference.
5. Click *Add Reference* to add a Web reference for the target XML Web service. For more information, see the topic "Adding and Removing Web References" in the Visual Studio documentation.
6. Visual Studio retrieves the service description and generates a proxy class to interface between your application and the XML Web service.

To import other Web services in your application, follow the same procedure described above for each WSDL file.

For a walk through of sample code that uses the WSDL generated stub, refer to the Appendix WebServices Client Sample Code section.

Standard Type Basics

Generally speaking, a data dictionary is the complete reference for the data model and services of a given Taleo Product. The data model consists of entities with fields and relations between other entities. Entities represent the information stored in the application. The services expose callable methods that allow you to access the data model entities from a client application.

To allow you to query, add, update, or delete data, all entity fields and relations are mapped into Taleo specific datatypes, hereafter called Standard Types.

The API exposes two categories of services: export and import services. Taleo products expose one single export service, called `FindService`: this one uses **Export Standard Types** to allow you to query data. All other services use **Import Standard Types** to allow you to add, edit, or delete data. As opposed to the export service, which is available for all products, import services are specific to each product: Please refer to the data dictionary of each Taleo product for an exhaustive list of its available Web services.

Import Standard Types

The following sections describes the standard types used by Web services that can add, edit, or delete data, as opposed to the Export service that can only query data.

The first section describes the mapping between Entity fields, as described in the Taleo product data dictionary and the Standard Types used by the API.

The next sections describe each of these standard types, providing usage samples using XML (SOAP messages), Java, and C# samples. These samples refer to the Taleo Enterprise Edition, Professional 7.5 data dictionary. The code snippets presented here are partial only, and aim to demonstrate the usage of each standard type. For a complete code sample in Java or C# involving an import request, refer to the Quick Start section

Entity Fields Definition vs. Import Standard Types Mapping

Each Entity Field is defined in the data dictionary (Field Details section) with the following import-relevant attributes:

- Create: The field can be set when not yet already set.
- Update: The field can be updated when already set.
- Search: The field can be set or updated using a lookup query (specifying a search value).
- Multilingual: The field can be set to multiple value, one per language.

The API considers all field values as String, no matter the type specified in the data model, and uses different Import Standard Types to handle the field attributes:

Import Standard Type (used to build import requests)	Attributes
String	Create Update
SearchableStringField	Create Update Search

Import Standard Type (used to build import requests)	Attributes
SearchableSearchOnlyField	Search
SearchableMultilingualStringField	Create Update Search Multilingual
SearchableMultilingualSearchOnlyField	Search Multilingual

If neither create, update, and search are checked for a given entity field in the data dictionary, the field cannot be imported or updated. If only search is checked, the field is either mapped to a *SearchableMultilingualSearchOnlyField* if the field is multilingual, or to a *SearchableSearchOnlyField* otherwise. This means the field is available for lookup but cannot be set to a non-existing value (you have to lookup the value to set it to the entity field). If create or update are checked but not search, then you can set a new value (or update one if update is checked), but cannot use an existing value (no lookup available). In this case, no matter the field type, the field is mapped to a simple *String* (cannot be multilingual). Finally, if create or update are checked, and search is also checked, the field is either mapped to a *SearchableMultilingualStringField* if the field is multilingual, or a *SearchableStringField* otherwise.

String

This type is a standard `xs:string` field, as specified in the *Schema W3C reference* (http://www.w3.org/TR/xmlschema-2/#string).

It is used when no lookup is available for the field. This means you will not be able to use this field to search for a specific entity, and setting a value to this field will replace any existing value for the field, if any was previously set.

Sample #1: Set a Candidate prefix

The Entity `Candidate` contains a Field `Prefix` that corresponds to a String field.

Java

```
Candidate candidate = Candidate.Factory.newInstance();
 candidate.setPrefix("Mr");
```

C#

```
taleo.candidateSvc.Candidate candidate = new
 taleo.candidateSvc.Candidate();
candidate.Prefix = "Mr";
```

XML

```
<Candidate>
    <Prefix>Mr</Prefix>
</Candidate>
```

SearchableStringField

This type is a Taleo object that allows to search for a specific value and optionally to replace it with another value. It holds three search criteria and the new value to set:

- **searchType:** The value can be "none" (do not use search feature), "search" (search for the searchValue attribute value but do not try to replace the value) or "searchAndValue", (search and replace the value with the field value).
- **searchValue:** The value of the field to search for.
- **searchTarget:** The value can be "." (entity to edit is the one containing the field), ".." (entity to edit is the parent of the one containing the field), "../.." (entity to edit is the grandparent of the one containing the field), etc.
- **stringValue:** The value of the field (field will be edited with this value).

Sample #1: Create a candidate and set LastName and FirstName

The `Candidate` entity contains the `LastName` and `FirstName` fields that both correspond to *SearchableStringFields*. The following example will create a candidate and set the candidate's email address, firstname, and lastname.

Java

```
Candidate candidate = Candidate.Factory.newInstance();
SearchableStringField emailAddress = candidate.addNewEmailAddress();
emailAddress.setStringValue("jsmith@acme.com");
SearchableStringField firstName = candidate.addNewFirstName();
firstName.setStringValue("John");
SearchableStringField lastName = candidate.addNewLastName();
lastName.setStringValue("Smith");
```

C#

```
taleo.candidateSvc.Candidate candidate = new
 taleo.candidateSvc.Candidate();
candidate.EmailAddress = new taleo.candidateSvc.searchableStringField();
candidate.EmailAddress.Value = "jsmith@acme.com";
candidate.FirstName = new taleo.candidateSvc.searchableStringField();
candidate.FirstName.Value = "John";
candidate.LastName = new taleo.candidateSvc.searchableStringField();
candidate.LastName.Value = "Smith";
```

XML

```
<Candidate>
    <EmailAddress>jsmith@acme.com</EmailAddress>
    <FirstName>John</FirstName>
    <LastName>Smith</LastName>
</Candidate>
```

Sample #2: Update LastName of existing candidate, searching by candidate's EmailAddress

The `Candidate` entity contains the `EmailAddress` and `LastName` fields that both correspond to *SearchableStringFields*. The following example will look for the candidate based on the e-mail address and update the candidate's lastname (leaving the candidate's email address and firstname as-is).

Java

```
Candidate candidate = Candidate.Factory.newInstance();
SearchableStringField emailAddress = candidate.addNewEmailAddress();
emailAddress.setSearchType(SearchableStringField.SearchType.SEARCH);
emailAddress.setSearchValue("jsmith@acme.com");
SearchableStringField lastName = candidate.addNewLastName();
lastName.setStringValue("Brown");
```

C#

```
taleo.candidateSvc.Candidate candidate = new
 taleo.candidateSvc.Candidate();
candidate.EmailAddress = new taleo.candidateSvc.searchableStringField();
candidate.EmailAddress.searchType =
 taleo.candidateSvc.searchableMultilingualStringFieldValueSearchType.search;
candidate.EmailAddress.searchTypeSpecified = true
candidate.EmailAddress.searchValue = "jsmith@acme.com";
candidate.LastName = new taleo.candidateSvc.searchableStringField();
candidate.LastName.Value = "Brown";
```

XML

```
<Candidate>
    <EmailAddress searchType="search" searchValue="jsmith@acme.com"/>
    <LastName>Brown</LastName>
</Candidate>
```

Sample #3: Update EmailAddress of existing candidate, searching by candidate's email address

In certain cases, the value used to determine the entity may also have to be updated; the syntax of the instruction would then be as follows (compare with previous samples):

Java

```
Candidate candidate = Candidate.Factory.newInstance();
SearchableStringField emailAddress = candidate.addNewEmailAddress();
emailAddress.setSearchType(SearchableStringField.SearchType.SEARCH_AND_VALUE);
emailAddress.setSearchValue("jsmith@acme.com");
emailAddress.setStringValue("jbrown@acme.com");
```

C#

```
taleo.candidateSvc.Candidate candidate = new
 taleo.candidateSvc.Candidate();
candidate.EmailAddress = new taleo.candidateSvc.searchableStringField();
candidate.EmailAddress.searchType =
 taleo.candidateSvc.searchableMultilingualStringFieldValueSearchType.searchAndValue;
candidate.EmailAddress.searchTypeSpecified = true
candidate.EmailAddress.searchValue = "jsmith@acme.com";
candidate.EmailAddress.Value = "Brown";
```

XML

```
<Candidate>
    <EmailAddress searchType="searchAndValue"
 searchValue="jsmith@acme.com">jbrown@acme.com</EmailAddress>
</Candidate>
```

Sample #4: Create a candidate and have that candidate apply on a specific requisition

When importing entities, most relations are lookups and the related entity is only linked to the main entity. This is the case for the `Applications` relation of the `Candidate` entity or the `Requisition` relation of the `PreselectionApplication`. To determine the related entity, we re-use the same search attributes, but in a different context. The following sample describes how to specify that John Smith applied on a specific Job Requisition (here Req001).

Java

```
Candidate candidate = Candidate.Factory.newInstance();
SearchableStringField emailAddress = candidate.addNewEmailAddress();
emailAddress.setStringValue("jsmith@acme.com");
SearchableStringField firstName = candidate.addNewFirstName();
```

```
firstName.setStringValue("John");
SearchableStringField lastName = candidate.addNewLastName();
lastName.setStringValue("Smith");
Applications applications = candidate.addNewApplications();
PreselectionApplication presel =
 applications.addNewPreselectionApplication();
Requisition reqcontainer = presel.addNewRequisition();
com.taleo.art750.candidate.Requisition requisition =
 reqcontainer.addNewRequisition();
SearchableStringField contestnumber = requisition.addNewContestNumber();
offerSequence.setSearchType(SearchableStringField.SearchType.SEARCH);
offerSequence.setSearchValue("Req001");
```

C#

```
taleo.candidateSvc.Candidate candidate = new
 taleo.candidateSvc.Candidate();
candidate.EmailAddress = new taleo.candidateSvc.searchableStringField();
candidate.EmailAddress.Value = "jsmith@acme.com";
candidate.FirstName = new taleo.candidateSvc.searchableStringField();
candidate.FirstName.Value = "John";
candidate.LastName = new taleo.candidateSvc.searchableStringField();
candidate.LastName.Value = "Smith";
candidate.Applications = new
 taleo.candidateSvc.PreselectionApplication[1]candidate.Applications[0] =
 new taleo.candidateSvc.PreselectionApplication();
candidate.Applications[0].Requisition = new
 taleo.candidateSvc.PreselectionApplicationRequisition();
candidate.Applications[0].Requisition.Requisition = new
 taleo.candidateSvc.Requisition();
candidate.Applications[0].Requisition.Requisition.ContestNumber = new
 taleo.candidateSvc.searchableStringField();
candidate.Applications[0].Requisition.Requisition.ContestNumber.searchType
 =
 taleo.candidateSvc.searchableMultilingualStringFieldValueSearchType.search;
candidate.Applications[0].Requisition.Requisition.ContestNumber.searchTypeSpecified
 = true;
candidate.Applications[0].Requisition.Requisition.ContestNumber.searchValue
 = "Req001";
```

XML

```
<Candidate>
    <EmailAddress>jsmith@acme.com</EmailAddress>
    <FirstName>John</FirstName>
    <LastName>Smith</LastName>
    <Applications>
        <PreselectionApplication>
            <Requisition>
                <Requisition>
                    <ContestNumber searchType="search"
 searchValue="Req001"/>
                </Requisition>
            </Requisition>
        </PreselectionApplication>
    </Applications>
</Candidate>
```

Sample #5: Update the application of a candidate having applied on a specific requisition

If we now want to set the date of entry for John Smith for the "Req001" job, we need to update the proper `Application` entity. To do so, we must find it among all his other possible applications. Since

there is no identifier in the `Application` entity itself, we must use the Requisition `ContestNumber` field to find the application. This is possible by specifying a searchTarget among the search attributes.

Java

```
Candidate candidate = Candidate.Factory.newInstance();
SearchableStringField emailAddress = candidate.addNewEmailAddress();
emailAddress.setStringValue("jsmith@acme.com");
SearchableStringField firstName = candidate.addNewFirstName();
firstName.setStringValue("John");
SearchableStringField lastName = candidate.addNewLastName();
lastName.setStringValue("Smith");
Applications applications = candidate.addNewApplications();
PreselectionApplication presel =
 applications.addNewPreselectionApplication();
SearchableStringField dateOfEntry = presl.addNewDateOfEntry();
dateOfEntry.setStringValue("2006-06-01T14:15:00-04:00"); // DateTime fields
 must always use that format
Requisition reqcontainer = presel.addNewRequisition();
com.taleo.art750.candidate.Requisition requisition =
 reqcontainer.addNewRequisition();
SearchableStringField contestnumber = requisition.addNewContestNumber();
contestnumber.setSearchType(SearchableStringField.SearchType.SEARCH);
contestnumber.setSearchTarget("..");
contestnumber.setSearchValue("Req001");
```

C#

```
taleo.candidateSvc.Candidate candidate = new
 taleo.candidateSvc.Candidate();
candidate.EmailAddress = new taleo.candidateSvc.searchableStringField();
candidate.EmailAddress.Value = "jsmith@acme.com";
candidate.FirstName = new taleo.candidateSvc.searchableStringField();
candidate.FirstName.Value = "John";
candidate.LastName = new taleo.candidateSvc.searchableStringField();
candidate.LastName.Value = "Smith";
candidate.Applications = new taleo.candidateSvc.PreselectionApplication[1];
candidate.Applications[0] = new
 taleo.candidateSvc.PreselectionApplication();
candidate.Applications[0].DateOfEntry = new
 taleo.candidateSvc.searchableStringField();
candidate.Applications[0].DateOfEntry.Value =
 "2006-06-01T14:15:00-04:00"; // DateTime fields must always use that
 format
candidate.Applications[0].Requisition = new
 taleo.candidateSvc.PreselectionApplicationRequisition();
candidate.Applications[0].Requisition.Requisition = new
 taleo.candidateSvc.Requisition();
candidate.Applications[0].Requisition.Requisition.ContestNumber = new
 taleo.candidateSvc.searchableStringField();
candidate.Applications[0].Requisition.Requisition.ContestNumber.searchType
 =
 taleo.candidateSvc.searchableMultilingualStringFieldValueSearchType.search;
candidate.Applications[0].Requisition.Requisition.ContestNumber.searchTypeSpecified
 = true;
candidate.Applications[0].Requisition.Requisition.ContestNumber.searchTarget
 = "..";
candidate.Applications[0].Requisition.Requisition.ContestNumber.searchValue
 = "Req001";
```

XML

```
<Candidate>
    <EmailAddress>jsmith@acme.com</EmailAddress>
    <FirstName>John</FirstName>
    <LastName>Smith</LastName>
    <Applications>
        <PreselectionApplication>
            <DateOfEntry>2006-06-01T14:15:00-04:00</DateOfEntry>
            <Requisition>
                <Requisition>
                    <ContestNumber searchType="search" searchValue="Req001"
 searchTarget=".."/>
                </Requisition>
            </Requisition>
        </PreselectionApplication>
    </Applications>
</Candidate>
```

SearchableSearchOnlyField

This type is almost identical to the *SearchableStringField*, with the exception that you cannot update the value of the field being searched. The only allowed *searchType* value is therefore "search".

Sample #1: Update a candidate and set the US dollar as default currency for this candidate

The `Candidate` entity contains the `Currency` relation, whose referenced object contains a `ISO4217Code` field that corresponds to a *SearchableSearchOnlyField*. The following example will define the US dollar (whose ISO-4217 code is 840) as the default currency used by John Smith.

Java

```
Candidate candidate = Candidate.Factory.newInstance();
SearchableStringField emailAddress = candidate.addNewEmailAddress();
emailAddress.setSearchType(SearchableStringField.SearchType.SEARCH);
emailAddress.setSearchValue("jsmith@acme.com");
Currency currencyRelation = candidate.addNewCurrency();
com.taleo.art750.candidate.Currency currency =
 currencyRelation.addNewCurrency();
SearchableSearchOnlyField code = currency.addNewISO4217Code();
code.setSearchType(SearchableSearchOnlyField.SearchType.SEARCH);
code.setSearchValue("840"); // USD
```

C#

```
taleo.candidateSvc.Candidate candidate = new
 taleo.candidateSvc.Candidate();
candidate.EmailAddress = new taleo.candidateSvc.searchableStringField();
candidate.EmailAddress.searchType =
 taleo.candidateSvc.searchableMultilingualStringFieldValueSearchType.search;
candidate.EmailAddress.searchTypeSpecified = true
candidate.EmailAddress.searchValue = "jsmith@acme.com";
candidate.Currency = new taleo.candidateSvc.CandidateCurrency();
candidate.Currency.Currency = new taleo.candidateSvc.Currency();
candidate.Currency.Currency.ISO4217Code = new
 taleo.candidateSvc.searchableSearchOnlyField();
candidate.Currency.Currency.ISO4217Code.searchType =
 taleo.candidateSvc.searchableSearchOnlyFieldSearchType.search;
candidate.Currency.Currency.ISO4217Code.searchValue = "840"; // USD
```

XML

```
<Candidate>
```

```
    <EmailAddress searchType="search" searchValue="jsmith@acme.com"/>
    <Currency>
        <ISO4217Code searchType="search" searchValue="840"/>
    </Currency>
</Candidate>
```

SearchableMultilingualStringField

This type is a Taleo object very similar to the *SearchableStringField* type, but that further allows multilingual values. Multilingual values are provided individually by locale, each in a dedicated value object.

Sample #1: Set the current job title for a candidate

The `Candidate` entity contains a `CurrentJob` relation, whose referenced object contains a `CurrentJobJobTitle` field that corresponds to a *SearchableMultilingualStringField*. The following example will define software developer John Smith's current job title in different languages (English, French and German).

Java

```
Candidate candidate = Candidate.Factory.newInstance();
SearchableStringField emailAddress = candidate.addNewEmailAddress();
emailAddress.setStringValue("jsmith@acme.com");
CurrentJob curJobRelation = candidate.addNewCurrentJob();
com.taleo.art750.candidate.CurrentJob curJob =
 curJobRelation.addNewCurrentJob();
SearchableMultilingualStringField jobTitle =
 curJob.addNewCurrentJobJobTitle();
com.taleo.art750.candidate.MultilingualStringField.Value en =
 jobTitle.addNewValue();
en.setLocale("en");
en.setStringValue("Software Developer");
com.taleo.art750.candidate.MultilingualStringField.Value fr =
 jobTitle.addNewValue();
fr.setLocale("fr");
fr.setStringValue("Développeur logiciel");
com.taleo.art750.candidate.MultilingualStringField.Value de =
 jobTitle.addNewValue();
de.setLocale("de");
de.setStringValue("Software-Entwickler");
```

C#

```
taleo.candidateSvc.Candidate candidate = new
 taleo.candidateSvc.Candidate();
candidate.EmailAddress = new taleo.candidateSvc.searchableStringField();
candidate.EmailAddress.searchType =
 taleo.candidateSvc.searchableMultilingualStringFieldValueSearchType.search;
candidate.EmailAddress.searchTypeSpecified = true
candidate.EmailAddress.searchValue = "jsmith@acme.com";
candidate.CurrentJob = new taleo.candidateSvc.CandidateCurrentJob();
candidate.CurrentJob.CurrentJob = new taleo.candidateSvc.CurrentJob();
candidate.CurrentJob.CurrentJob.CurrentJobJobTitle = new
 taleo.candidateSvc.searchableMultilingualStringFieldValue[3]; // 3
 languages
candidate.CurrentJob.CurrentJob.CurrentJobJobTitle[0] = new
 taleo.candidateSvc.searchableMultilingualStringFieldValue();
candidate.CurrentJob.CurrentJob.CurrentJobJobTitle[0].locale = "en";
candidate.CurrentJob.CurrentJob.CurrentJobJobTitle[0].Value = "Software
 Developer";
```

```
candidate.CurrentJob.CurrentJob.CurrentJobJobTitle[1] = new
 taleo.candidateSvc.searchableMultilingualStringFieldValue();
candidate.CurrentJob.CurrentJob.CurrentJobJobTitle[1].locale = "fr";
candidate.CurrentJob.CurrentJob.CurrentJobJobTitle[1].Value = "Développeur
 logiciel";
candidate.CurrentJob.CurrentJob.CurrentJobJobTitle[2] = new
 taleo.candidateSvc.searchableMultilingualStringFieldValue();
candidate.CurrentJob.CurrentJob.CurrentJobJobTitle[2].locale = "de";
candidate.CurrentJob.CurrentJob.CurrentJobJobTitle[2].Value = "Software-
Entwickler";
```

XML

```
<Candidate>
    <EmailAddress searchType="search" searchValue="jsmith@acme.com"/>
    <CurrentJob>
        <CurrentJob>
            <CurrentJobJobTitle>
                <value locale="en">Software Developer</value>
                <value locale="fr">Développeur logiciel</value>
                <value locale="de">Software-Entwickler</value>
            </CurrentJobJobTitle>
        </CurrentJob>
    </CurrentJob>
</Candidate>
```

Sample #2: Mark an application state for a candidate as new

The `Candidate` entity contains a Applications relation, whose referenced object itself contains a `ApplicationState` relation, whose referenced object contains a `Description` field that corresponds to a *SearchableMultilingualStringField*. The following example will link the "new" application state to the preselection application of the candidate instead of creating any new application state object, by using a lookup on the English description value.

Java

```
Candidate candidate = Candidate.Factory.newInstance();
SearchableStringField emailAddress = candidate.addNewEmailAddress();
emailAddress.setStringValue("jsmith@acme.com");
Applications applications = candidate.addNewApplications();
PreselectionApplication presel =
 applications.addNewPreselectionApplication();
ApplicationState appstatecontainer = presel.addNewApplicationState();
com.taleo.art750.candidate.ApplicationState appstate =
 appstatecontainer .addNewApplicationState();
SearchableMultilingualStringField appstatedesc =
 appstate.addNewDescription();
Value descvalue = appstatedesc.addNewValue();
descvalue.setLocale("en");
descvalue.setSearchType(SearchableMultilingualStringField.Value.SearchType.SEARCH);
descvalue.setSearchValue("New");
```

C#

```
taleo.candidateSvc.Candidate candidate = new
 taleo.candidateSvc.Candidate();
candidate.EmailAddress = new taleo.candidateSvc.searchableStringField();
candidate.EmailAddress.searchType =
 taleo.candidateSvc.searchableMultilingualStringFieldValueSearchType.search;
candidate.EmailAddress.searchTypeSpecified = true
candidate.EmailAddress.searchValue = "jsmith@acme.com";
candidate.Applications = new taleo.candidateSvc.PreselectionApplication[1];
```

```
candidate.Applications[0] = new
 taleo.candidateSvc.PreselectionApplication();
candidate.Applications[0].ApplicationState = new
 taleo.candidateSvc.PreselectionApplicationApplicationState();
candidate.Applications[0].ApplicationState.ApplicationState = new
 taleo.candidateSvc.ApplicationState();
candidate.Applications[0].ApplicationState.ApplicationState.Description =
 new taleo.candidateSvc.searchableMultilingualStringFieldValue[1];
candidate.Applications[0].ApplicationState.ApplicationState.Description[0]
 = new taleo.candidateSvc.searchableMultilingualStringFieldValue();
candidate.Applications[0].ApplicationState.ApplicationState.Description[0].locale
 = "en";
candidate.Applications[0].ApplicationState.ApplicationState.Description[0].searchType
 =
 taleo.candidateSvc.searchableMultilingualStringFieldValueSearchType.search;
candidate.Applications[0].ApplicationState.ApplicationState.Description[0].searchTypeSpe
 = true;
candidate.Applications[0].ApplicationState.ApplicationState.Description[0].searchValue
 = "New";
```

XML

```
<Candidate>
    <EmailAddress searchType="search" searchValue="jsmith@acme.com"/>
    <Applications>
        <PreselectionApplication>
            <ApplicationState>
                <ApplicationState>
                    <Description>
                        <value searchType="search" searchValue="New"
 locale="en" />
                    </Description>
                </ApplicationState>
            </ApplicationState>
        </PreselectionApplication>
    </Applications>
</Candidate>
```

SearchableMultilingualSearchOnlyField

This type is a Taleo object very similar to the *SearchableSearchOnlyField* type, but that further allows multilingual values. Multilingual values are provided individually by locale, each in a dedicated `value` object.

Sample #1: Set the current currency symbol for a candidate

The `Candidate` entity contains a `Currency` relation, whose referenced object contains a `Symbol` field that corresponds to a *SearchableMultilingualSearchOnlyField*. The following example will set the currency to be used for the candidate to the US dollar.

Java

```
Candidate candidate = Candidate.Factory.newInstance();
Currency currentRelation = candidate.addNewCurrency();
com.taleo.art750.candidate.Currency currency =
 currentRelation.addNewCurrency();
SearchableMultilingualSearchOnlyField symbol = currency.addNewSymbol();
com.taleo.art750.candidate.SearchableMultilingualSearchOnlyField.Value
 value = symbol.addNewValue();
value.setSearchType(SearchableMultilingualSearchOnlyField.Value.SearchType.SEARCH);
value.setSearchValue("$"); // US Dollar
value.setLocale("en");
```

C#

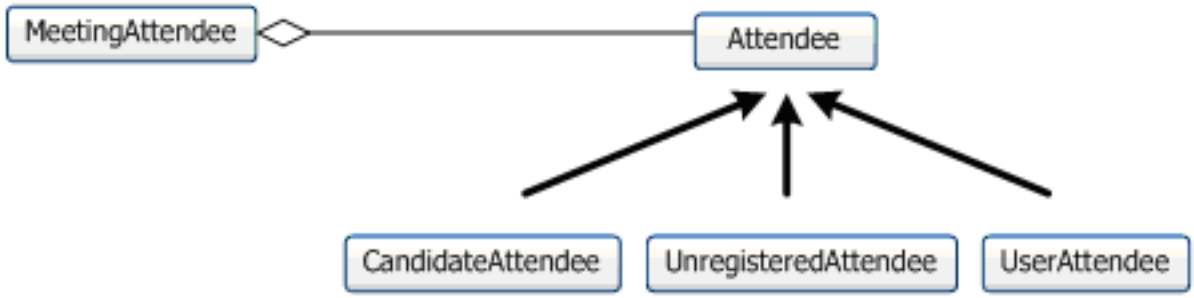
```
taleo.candidateSvc.Candidate candidate = new
 taleo.candidateSvc.Candidate();
candidate.Currency = new taleo.candidateSvc.CandidateCurrency();
candidate.Currency.Currency = new taleo.candidateSvc.Currency();
candidate.Currency.Currency.Symbol = new
 taleo.candidateSvc.searchableMultilingualSearchOnlyFieldValue[1];
candidate.Currency.Currency.Symbol[0] = new
 taleo.candidateSvc.searchableMultilingualSearchOnlyFieldValue();
candidate.Currency.Currency.Symbol.searchType =
 taleo.candidateSvc.searchableSearchOnlyFieldSearchType.search;
candidate.Currency.Currency.Symbol.searchValue = "$"; // US Dollar
candidate.Currency.Currency.Symbol.locale = "en";
```

XML

```
<Candidate>
    <Currency>
        <Currency>
            <Symbol>
                <Value locale="en" searchType="search" searchValue="$" />
            </Symbol>
        </Currency>
    </Currency>
</Candidate>
```

Operations On Parameters With Base Type Elements

The Taleo data model is composed of base and specialized elements. They define a logical hierarchical representation of the Taleo entities. A concrete example of this is:



You have an operation that requires a MeetingAttendee entity as a parameter. This entity is composed of an Attendee entity. The Attendee entity is an abstraction of the meeting expected participants. The three possible types of attendees are CandidateAttendee, UnregistredAttendee and UserAttendee. The Taleo WSDL is constructed to support the passage of one of these three entities as the MeetingAttendee parameter instead of the Attendee.

Sample #1: Using the requisition merge operation with an entity that is using UDF

Java

```
// Create a merge document
MergeDocument mergeDoc = MergeDocument.Factory.newInstance();
Merge merge = mergeDoc.addNewMerge();
Requisition req = merge.addNewRequisition();

//Set the search key
SearchableStringField  contestNumber = req.addNewContestNumber();
contestNumber.setSearchValue("INT-REQ-SRC-122354");
```

```
contestNumber.setStringValue("INT-REQ-SRC-122354");
contestNumber.setSearchTarget(".");
contestNumber.setSearchType(SearchableStringField.SearchType.SEARCH_AND_VALUE);
req.setContestNumber(contestNumber);

// Create the UDFs
UDFs udfs = UDFs.Factory.newInstance();

// Create a UDF
UDF udf = udfs.addNewUDF();
udf.setName("TST_5f22_5f00_5f30");

// Create and add a new UDSElement to the UDFCreate a UDSElement
// UDF normally points to a Entity object but since UDSElement is now a
 choice
// of UDF, it is possible to create a UDSElement and associate it to the
 UDF.
UDSElement udsElement = udf.addNewUDSElement();

SearchableMultilingualStringField desc =
 SearchableMultilingualStringField.Factory.newInstance();
Value val = desc.addNewValue();
val.setStringValue("Cost Center 2.10");
val.setSearchValue("Cost Center 2.10");
val.setLocale("en");
val.setSearchTarget(".");
val.setSearchType(SearchableMultilingualStringField.Value.SearchType.SEARCH_AND_VALUE);
udsElement.setDescription(desc);

// Set the UDFs in the requisition
req.addNewJobInformation().addNewJobInformation().setUDFs(udfs);

try
{
    // Invoke the requisition create service.
     pRequisitionService.merge(mergeDoc);
}
catch (RemoteException re)
{
     re.printStackTrace();
}
catch (WebServiceFault wsf)
{
     wsf.printStackTrace();
}
```

C#

```
Requisition req = new Requisition();

//Set the search key
req.ContestNumber = new searchableStringField();
req.ContestNumber.Value = "INT-REQ-SRC-CS-122354";
req.ContestNumber.searchType =
 searchableStringFieldSearchType.searchAndValue;
req.ContestNumber.searchTypeSpecified = true;
req.ContestNumber.searchValue = req.ContestNumber.Value;

//Creation of the Job Information
req.JobInformation = new RequisitionJobInformation();
req.JobInformation.JobInformation = new JobInformation();
```

```
// Creation of an UDF element in the requisition
UDFsUDF[] udfs = new UDFsUDF[1];
req.JobInformation.JobInformation.UDFs = udfs;
req.JobInformation.JobInformation.UDFs[0] = new UDFsUDF();
req.JobInformation.JobInformation.UDFs[0].name =
 "LUDS_5fLogistics_5fRWT_2bCS";


// Creation of the UDSElement
UDSElement udsElement = new UDSElement();
searchableMultilingualStringFieldValue[] desc = new
 searchableMultilingualStringFieldValue[1];
udsElement.Description = desc;
udsElement.Description[0] = new searchableMultilingualStringFieldValue();
udsElement.Description[0].locale = "en";
udsElement.Description[0].searchValue = "LUD CANM 1";
udsElement.Description[0].searchType =
 searchableStringFieldSearchType.search;
udsElement.Description[0].searchTypeSpecified = true;

// Add the UDSElement to the UDFs
req.JobInformation.JobInformation.UDFs[0].Items = new UDSElement[1];
req.JobInformation.JobInformation.UDFs[0].Items[0] = udsElement;

XmlSerializer serializer = new XmlSerializer(req.GetType());
StringWriter sw = new StringWriter();
serializer.Serialize(sw, req);
System.Windows.Forms.MessageBox.Show(sw.ToString());

try
{
    // Invoke the requisition create service.
    String result = requisitionService.create(req);
}
catch (Exception e)
{
    e.Message.ToString();
}
```

VB

```
Dim requisition As Requisition = New Requisition

' A "searchable" string field allows a) to edit a field value or b) to use
 a field
' value already existing in Taleo systems. In the following lines, we just
 want to set
' the values.

requisition.ContestNumber = New searchableStringField
requisition.ContestNumber.Value = "INT-REQ-SRC-VB-122354"

'To do a merge, add those lines.
requisition.ContestNumber.searchType =
 searchableStringFieldSearchType.searchAndValue
requisition.ContestNumber.searchTypeSpecified = True
requisition.ContestNumber.searchValue = requisition.ContestNumber.Value

'Creation of the Job Information
```

```
requisition.JobInformation = New RequisitionJobInformation
requisition.JobInformation.JobInformation = New JobInformation


'Creation of an UDF element in the requisition
Dim UDFs(0) As UDFsUDF
requisition.JobInformation.JobInformation.UDFs() = UDFs
requisition.JobInformation.JobInformation.UDFs(0) = New UDFsUDF
requisition.JobInformation.JobInformation.UDFs(0).name =
 "LUDS_5fLogistics_5fRWT_2bCS"

Dim UDF_UDSElement(0) As UDSElement
requisition.JobInformation.JobInformation.UDFs(0).Items = UDF_UDSElement

Dim UDF_UDSElement1 As New UDSElement
requisition.JobInformation.JobInformation.UDFs(0).Items(0) =
 UDF_UDSElement1
Dim UDF_UDSElement_Description(0) As searchableMultilingualStringFieldValue
UDF_UDSElement1.Description = UDF_UDSElement_Description
UDF_UDSElement1.Description(0) = New searchableMultilingualStringFieldValue
UDF_UDSElement1.Description(0).locale = "en"
UDF_UDSElement1.Description(0).searchValue = "LUD CANM 1"
UDF_UDSElement1.Description(0).searchType =
 searchableStringFieldSearchType.search
UDF_UDSElement1.Description(0).searchTypeSpecified = True

' Check the query.
Dim serializer As New XmlSerializer(GetType(Requisition))
Dim sw As New StringWriter()
serializer.Serialize(sw, requisition)
MsgBox(sw.ToString())

'
 ********************************************************************************
Try
    RequisitionService.merge(requisition)

Catch ex As Exception
    MsgBox(ex.ToString)
End Try


'
 ********************************************************************************
' 4 - get and handle results
'
 ********************************************************************************
' the requisitionSvc.create() does not return any result, and if no error
 is thrown,
' it means the requisition was successfully inserted into Taleo systems

Console.WriteLine("requisition successfully imported in Taleo system using
 requirement: " & requisition.ContestNumber.Value)
```

XML

```
<merge xmlns="http://www.taleo.com/ws/art750/2006/12">
  <requisition>
    <ContestNumber searchValue="INT-REQ-SRC-122354" searchTarget="."
 searchType="searchAndValue">INT-REQ-SRC-122354</ContestNumber>
    <JobInformation>
```

```
      <JobInformation>
        <UDFs>
          <UDF name="TST_5f22_5f00_5f30">
            <UDSElement>
              <Description>
                <value searchValue="Cost Center 2.10" locale="en"
 searchTarget="." searchType="searchAndValue">Cost Center 2.10</value>
              </Description>
            </UDSElement>
          </UDF>
        </UDFs>
      </JobInformation>
    </JobInformation>
  </requisition>
</merge>
```

Export Standard Types

The following sections describe the standard types used by the FindService Web service that allows client applications to query data.

The first section provides an overview of Selection Query language (SQ-XML) that allows to build efficient data queries against the Taleo data model. For a complete reference of this language, refer to the Selection Query Language, SQ-XML section.

The next section describes the mapping between Entity fields, as described in the Taleo product data dictionary, and the standard types used by the API. As opposed to the standard import types, you will use these types to handle the response to queries sent to the FindService Web service.

The last sections describe each of these standard types, providing usage samples using XML (SOAP messages), Java, and C# samples. These samples refer to the Taleo Enterprise Edition, Professional 7.5 data dictionary. The code snippets presented here are partial only and aim to demonstrate the usage of each Standard Type. For a complete code sample in Java or C# involving an export request, refer to the Quick Start section.

Building Export Queries Using Selection Query (SQ-XML)

Use the Selection Query language (SQ-XML) to construct simple but powerful queries for the `sqxmlquery` parameter in the *findEntities* and *findPartialEntitie*s calls of the `FindService`. Similar to the SELECT command in SQL, SQ-XML allows you to specify the source object (such as Account), a list of fields to retrieve, and conditions for selecting rows in the source object.

A meta model differs mainly from a relational data model in terms of the relationships created between its entities. As such, the Selection Query language differs from the SQL language mainly in the same manner. Since ultimately the Selection Query engine will translate all SQ-XML expressions into SQL statements to be executed against the physical model, Selection Query expressions are really very close to their SQL counterparts. Resources accustomed to creating SQL extraction scripts should easily grasp the workings of the Selection Query format. For SQL neophytes, the SQ-XML offers a simpler alternative for working with extraction instructions. This section will present the "equivalent" SQL statement of the described SQ-XML documents. Please note that this is done ONLY for reference purposes. In almost all cases, the application entities and fields do NOT have the same name as the underlying physical elements. We purposely use the application model terms to clearly distinguish between examples and the SQL statements that would be actually generated by the export service.

Note: This section does not document the export service itself because this is already documented in the Data Model of your Taleo product.

Overview

Building an export instruction is much more involved than in the case of the import feature. This section presents a simple example using the Professional 7.5 data model. The export instruction basically

needs to specify what type of entity is exported, which entities are to be selected, and what fields are to be extracted. Assuming we want to extract all requisitions that are currently open and posted in Taleo Enterprise 7.5 product, the instruction would look like the following:

```
<ns:query alias="Find Open and Posted Requisitions"
 projectedClass="Requisition">
    <ns:projections>
        <ns:projection>
            <ns:field path="ContestNumber"/>
        </ns:projection>
        <ns:projection>
            <ns:field path="JobInformation,Title"/>
        </ns:projection>
    </ns:projections>
    <ns:filterings>
        <ns:filtering>
            <ns:includedIn>
                <ns:field path="State,Number"/>
                <ns:list>
                    <ns:long>3<!--state=open--></ns:long>
                    <ns:long>13<!--state=posted--></ns:long>
                </ns:list>
            </ns:includedIn>
        </ns:filtering>
    </ns:filterings>
    <ns:sortings>
        <ns:sorting>
            <ns:field path="ContestNumber"/>
        </ns:sorting>
    </ns:sortings>
</ns:query>
```

The next sections cover the 4 most important parts of a SQ-XML query:

- Basics—The entity type to be extracted
- Projections—The fields to be extracted
- Filterings—The conditions applying to the query (reducing the scope of entities to be extracted)
- Sortings—The order which to sort the extracted results

Each section will provide an XML sample and an SQL equivalence of the export request. For similar samples in Java or C#, refer to the Quick Start (see Appendix - WebServices Client Sample Code.), which provides a complete sample of a requisition export in both programming languages.

Basics

An SQ-XML document typically starts with a query element. There are two required attributes to the query element: `projectedClass` and `alias`. The former represents the base entity from which the extraction will be built. The latter is a uniqe name throughout the expression that identifies the query.

A basic query starts like this:

```
<query alias="BasicQuery" projectedClass="User"/>
```

SQL Equivalent: `FROM User`

Projections

The first main elements of a query are the projections that represent what information is to be extracted for the selected entities. The projection elements can be defined as any value understood by the Selection Query language. The simplest case is to use a field of the projected class.

```
<query alias="SimpleProjection" projectedClass="User">
    <projections>
```

```
        <projection>
            <field path="UserName"/>
        </projection>
    </projections>
</query>
```

SQL Equivalent: `SELECT UserName FROM User`

It is possible to assign an alias to a projection; this serves two purposes. First, when a function is used, the Selection Query cannot deduce a default alias. Hence, it is required to explicitly specify one. Second, when sub queries are involved, sometimes aliases are required to distinguish projections. This is because the default alias is the entity field name; so if both the main query and a sub-query project the Email field, then one of them will need an alias.

```
<query alias="ProjectionWithAlias" projectedClass="User">
    <projections>
        <projection alias="Login">
            <field path="UserName"/>
        </projection>
    </projections>
</query>
```

SQL Equivalent: `SELECT UserName AS Login FROM User`

The real strength of the Selection Query language comes from the application model relations. When such relations exist for the target data, then projecting it becomes straightforward. For example, by selecting the Department relation of the User entity (which is one-to-one according to the schema), you can access all of the Department entity fields. When the path expression only specifies the relation, then it is the entity key that is projected.

```
<query alias="ProjectionWithRelations" projectedClass="User">
    <projections>
        <projection>
            <field path="UserName"/>
        </projection>
        <projection>
            <field path="Department"/>
        </projection>
        <projection>
            <field path="Department,Name"/>
        </projection>
    </projections>
</query>
```

SQL Equivalent: `SELECT UserName, DepartmentNo, Department.Name FROM User, Department WHERE User.DepartmentNo = Department.No`

When navigating a relation, you also have access to all the relations of the related entity. In the example below, since the Recruiter relation of the Department entity points to a User entity, then all the fields of that entity are again available.

```
<query alias="ProjectionWithDeepRelations" projectedClass="User">
    <projections>
        <projection>
            <field path="UserName"/>
        </projection>
        <projection>
            <field path="Department,Recruiter,UserName"/>
        </projection>
    </projections>
</query>
```

SQL Equivalent: `SELECT UserName, Recruiter.UserName FROM User, Department, User Recruiter WHERE User.DepartmentNo = Department.No AND Department.RecruiterNo = Recruiter.No`

Filtering elements

The next query element includes the filters that represent what entities are to be selected.

The filtering elements are grouped in sequence within the filterings element, although the sequence itself is not relevant. The various filtering elements are implicitly linked by an AND logical operator. The filtering elements can be defined as any filter understood by the Selection Query language; these are either logical operators or actual conditions. The simplest case is to use a standard equality condition. The equal is a binary operator and as such accepts two value child elements. One simple possibility is to use a field and a fixed value. We saw fields in the projection sections; the fixed values are of the normal types: numeric, string, etc.

```
<query alias="EqualityFilter" projectedClass="User">
    <projections>
        <projection>
            <field path="UserName"/>
        </projection>
    </projections>
    <filterings>
        <filtering>
            <equal>
                <field path="FirstName"/>
                <string>John</string>
            </equal>
        </filtering>
    </filterings>
</query>
```

SQL Equivalent: `SELECT UserName FROM User WHERE FirstName = 'John'`

In the previous example, the SQ-XML is slightly more complex than the SQL equivalent. However, once again, the power of the expression language resides in the application model relations that allow a simple modification to filter on other relationships such as the Department name.

```
<query alias="RelationFilter" projectedClass="User">
    <projections>
        <projection>
            <field path="UserName"/>
        </projection>
    </projections>
    <filterings>
        <filtering>
            <equal>
                <field path="Department,Name"/>
                <string>Finance</string>
            </equal>
        </filtering>
    </filterings>
</query>
```

SQL Equivalent: `SELECT UserName FROM User, Department WHERE User.DepartmentNo = Department.No AND Department.Name = 'Finance'`

Applying a single logical condition can be done directly with the proper element.

```
<query alias="AndFilter" projectedClass="User">
    <projections>
        <projection>
```

```
            <field path="UserName"/>
        </projection>
    </projections>
    <filterings>
        <filtering>
            <and>
                <equal>
                    <field path="FirstName"/>
                    <string>John</string>
                </equal>
                <equal>
                    <field path="LastName"/>
                    <string>Doe</string>
                </equal>
            </and>
        </filtering>
    </filterings>
</query>
```

SQL Equivalent: `SELECT UserName FROM User WHERE FirstName = 'John' AND LastName = 'Doe'`

However, applying several logical conditions must be done in an embedded manner, as most logical operator elements are binary (that is, accept only two child elements).

```
<query alias="MultipleAndFilters" projectedClass="User">
    <projections>
        <projection>
            <field path="UserName"/>
        </projection>
    </projections>
    <filterings>
        <filtering>
            <and>
                <and>
                    <equal>
                        <field path="FirstName"/>
                        <string>John</string>
                    </equal>
                    <equal>
                        <field path="LastName"/>
                        <string>Doe</string>
                    </equal>
                </and>
                <equal>
                    <field path="MiddleInitial"/>
                    <string>R</string>
                </equal>
            </and>
        </filtering>
    </filterings>
</query>
```

SQL Equivalent: `SELECT UserName FROM User WHERE FirstName = 'John' AND LastName = 'Doe' AND MiddleInitial = 'R'`

Sorting elements

The last main query element is the sorting instructions that represent in what order the selected entities will be shown.

The sorting elements are grouped in sequence within the sortings element. The sequence determines what sorting instructions are applied first. The sorting elements accept any value as child elements, but the main usage is with fields of the projected entity. The sorting elements also accept an ascending attribute that determines the orientation of the particular ordering. Just like in SQL, this attribute defaults to true.

```
<query alias="Sorting" projectedClass="User">
    <projections>
        <projection>
            <field path="UserName"/>
        </projection>
    </projections>
    <sortings>
        <sorting>
            <field path="LastName"/>
        </sorting>
        <sorting ascending="false">
            <field path="FirstName"/>
        </sorting>
    </sortings>
</query>
```

SQL Equivalent: `SELECT UserName FROM User ORDER BY LastName ASC, FirstName DESC`

Entity Fields vs. Standard Types Mapping

Each Entity Field is annotated in the data dictionary (Field Details section) with the following attributes:

- **Export:** The field can be exported.
- **Multilingual:** The field can be set to multiple values, one per language.

The API considers all field values as String, no matter the type specified in the data model, and use different Export Standard Types to handle the field attributes:

Export Standard Type (used to handle export responses)	Attributes
String	Export
MultilingualStringField	Export Multilingual

If the checkbox for export is cleared for a given entity field in the data dictionary, this means the field cannot be exported. If the field can be exported, the field is either mapped to a *MultilingualStringField* if the field is multilingual, or to a *String* otherwise, no matter the field type (i.e. Boolean, Integer, Float, DateTime).

String

This type is a standard `xs:string` field, as specified in the *Schema W3C reference* (http://www.w3.org/TR/xmlschema-2/#string). Because the API considers any field type as a String, you may be required to cast the received value to the proper type in your code (see sample).

Sample #1: Retrieve the requisition identifier of an exported Requisition

The Entity `Requisition` contains the Field `ContestNumber` and `HasBeenApproved` that both correspond to a String field. This sample focuses on the result handling (we assume the export query

was successfully executed by the Web service). It reads the first exported requisition contest number (requisition identifier) and whether this one was approved or not.

Java

```
FindPartialEntitiesResponseDocument response =
 pFindService.findPartialEntities(findDoc);
FindPartialEntitiesResponse rsp =
 response.getFindPartialEntitiesResponse();
Entities pagedResults = rsp.getEntities();
com.taleo.export.art750.Requisition requisition =
 (com.taleo.export.art750.Requisition) entities.getEntityArray(0);
String contestNumber = requisition.getContestNumber();
// following conversion from String to Boolean is required to get the value
 as a boolean
boolean hasBeenApproved =
 Boolean.valueOf(requisition.getHasBeenApproved()).booleanValue();
if (hasBeenApproved) {
    System.out.println("Approved Requisition: " + contestNumber);
}
```

C#

```
taleo.findSvc.Entities pagedResults =
 pfindSvc.findPartialEntities(mappingVersion, sqxmlquery, attrs);
taleo.findSvc.Requisition requisition = entities.Entity[0];
String contestNumber = requisition.ContestNumber;
// following conversion from String to Boolean is required to get the value
 as boolean
bool hasBeenApproved = Convert.ToBoolean(requisition.HasBeenApproved);
if (hasBeenApproved)
{
    Console.WriteLine("Approved Requisition: " + contestNumber);
}
```

XML

```
<ns1:findPartialEntitiesResponse xmlns:ns1="http://www.taleo.com/ws/
integration/toolkit/2005/07">
    <Entities pageIndex="1" pageCount="1" pagingSize="200" entityCount="1"
            xmlns:e="http://www.taleo.com/ws/art750/2006/12"
            xmlns="http://www.taleo.com/ws/integration/toolkit/2005/07">
        <e:Entity xsi:type="e:Requisition">
             <e:ContestNumber>Req001</e:ContestNumber>
             <e:HasBeenApproved>false</e:HasBeenApproved>
        </e:Entity>
    </Entities>
</ns1:findPartialEntitiesResponse>
```

MultilingualStringField

This type is a Taleo object very similar to the *String* type, but further allows to retrieve multilingual exported values. Multilingual values are provided individually by locale, each in a dedicated `value` object.

Sample #1: Retrieve the current job title of a Candidate

The `Candidate` entity contains a `CurrentJob` relation, whose referenced object contains a `CurrentJobJobTitle` field that corresponds to a *MultilingualStringField*. This sample focuses on the results handling (we assume the export query was successfully executed by the Web service). It reads the current job from the first exported Candidate number and outputs the job title for all available languages (here English, French, and German).

Java

```
FindPartialEntitiesResponseDocument response =
 pFindService.findPartialEntities(findDoc);
FindPartialEntitiesResponse rsp =
 response.getFindPartialEntitiesResponse();
Entities pagedResults = rsp.getEntities();
com.taleo.export.art750.Candidate candidate =
 (com.taleo.export.art750.Candidate) entities.getEntityArray(0);
com.taleo.export.art750.CurrentJob currentJob =
 candidate.getCurrentJob().getCurrentJob();
com.taleo.export.art750.MultilingualStringField title =
 currentJob.getCurrentJobJobTitle();
com.taleo.export.art750.MultilingualStringField.Value[] values =
 title.getValueArray();
for (int i = 0; i < values.length; i++ {
    com.taleo.export.art750.MultilingualStringField.Value value =
 values[i];
    String locale = value.getLocale();
    String localizedTitle = value.getStringValue();
    System.out.println("locale: " + locale + " - title: " +
 localizedTitle);
}
```

C#

```
taleo.findSvc.Entities pagedResults =
 pfindSvc.findPartialEntities(mappingVersion, sqxmlquery, attrs);
taleo.findSvc.Candidate candidate = entities.Entity[0];
taleo.findSvc.multilingualStringFieldValue[] titles =
 candidate.CurrentJob.CurrentJob.CurrentJobJobTitle;
foreach(taleo.findSvc.multilingualStringFieldValue title in titles)
{
    String locale = titles[0].locale;
    String localizedTitle = titles[0].Value;
    Console.WriteLine("Locale: " + locale + " - title: " + localizedTitle);
}
```

XML

```
<ns1:findPartialEntitiesResponse xmlns:ns1="http://www.taleo.com/ws/
integration/toolkit/2005/07">
    <Entities pageIndex="1" pageCount="1" pagingSize="200" entityCount="1"
            xmlns:e="http://www.taleo.com/ws/art750/2006/12"
            xmlns="http://www.taleo.com/ws/integration/toolkit/2005/07">
        <e:Entity xsi:type="e:Candidate">
            <e:CurrentJob>
                <e:CurrentJob>
                    <e:CurrentJobJobTitle>
                        <e:value locale="en">Software Developer</e:value>
                        <e:value locale="fr">Developeur logiciel</e:value>
                        <e:value locale="de">Software-Entwickler</e:value>
                    </e:CurrentJobJobTitle>
                </e:CurrentJob>
            </e:CurrentJob>
        </e:Entity>
    </Entities>
</ns1:findPartialEntitiesResponse>
```

API Call Basics

API calls represent specific operations that your client applications can invoke at runtime to perform tasks, for example:

- Query data in your organization
- Add, update, and delete data

Characteristics of API Calls

All API calls are:

- **Service Requests and Responses:** Your client application prepares and submits a service request to the Taleo Web service via the API, the Taleo Web service processes the request and returns a response, and the client application handles the response
- **Synchronous:** Once the API call is invoked, your client application waits until it receives a response from the service.
- **Committed Automatically:** An application transaction is created for every operation that writes to a Taleo object. If the operation completes successfully, this transaction is automatically committed. If an error occurs while performing the operation, the transaction is automatically rolled back. For example, if a client application attempts to create a new candidate that includes one application, and the application creation fails, neither the candidate nor the application will be created and an error will be returned to the client application.

API Usage Limits and Metering

To protect the Taleo systems and your organization's data reliability, security, and scalability, the Taleo API is subject to different usage limits and metering. The limits documented in the following sections are subject to change in future releases. A more detailed documentation of the Taleo web service usage limits and metering is available on demand. Contact your Customer Representative to request this document.

Global Usage Metering

Taleo may monitor every API call for metering, accounting, or troubleshooting purposes.

Global Usage Limits

The following limits apply for any incoming Web service request:

- **Maximum of 20 concurrent Web service calls:** If a client application invokes a Web service while 20 Web services are running, access will be denied to that Web service and a fault message is returned to the client application
- **Maximum of 25,000 Web service calls per day:** The counter begins at the first API call and is reset at midnight every day.

Export Usage Limits

The export service is generic and, as oppposed to other Web services that only handle one entity at a time, can be used to export a large amount of entities. Therefore, the following additional limits apply for export requests only:

- **Maximum of 200 records per export call:** If your request produces more than 200 records, you may consider using Taleo Connect Client (http://www.taleo.com/solutions/connect.php), which uses an asynchronous processing for large, time- and resource-consuming integration requests. Alternatively, if a synchronous invocation is required, you must use the pagination mechanism of the export Web service.

- **Maximum of 250,000 export records per day:** The counter begins at the first export API call and is reset at midnight every day.
- **Maximum response size of 2048 Kilobytes (2 MB) per export call:** If an API call produces a response that exceeds this size, the request is aborted and a fault message is returned to the client application. By the nature of Web service synchronous calls, a request should not produce a response larger than this size: in most cases, this is due to an incorrect request,ffor example, with invalid or missing request filterings.
- **Maximum of 90 seconds per export call:** If an API call takes longer than 90 seconds to complete, the request is aborted and a fault message is returned to the client application. Consider splitting the complex export request into several smaller and simpler ones. Additionally, validate the export complexity and if needed, add additional filterings that could reduce the request execution time.

Security and the API

Client applications that access your organization's Taleo data are subject to the same security protections that are used in the Taleo user interface. Additional protection is available for organizations that implement their own authentication and authorization mechanisms and want to authorize the user on the Taleo application.

Basic HTTP Authentication

Basic HTTP authentication is mandatory to access Taleo Web services if and only if no WS-Security token (WSS SOAP header) is part of the SOAP request. The user who logs in to Taleo Web services will benefit from its permission set when calling services.

WS-Security Authentication

Taleo is compliant with the SAML and Username Token Profiles 1.1 Specifications, which are both parts of the WS-Security 1.1 Standard.

Taleo highly recommands using WS-Security instead of HTTP basic authentication. Sending a WS-Security Token, preferably a SAML token, is a more secure alternative than HTTP basic authentication. Signing the token further ensures its integrity (not modified).

If a WS-Security token (WSS SOAP header) is part of the SOAP request, you must not use HTTP basic authentication, since both the authentication and the authorization is handled by the WS-Security implementation. If the SOAP request does not contain any WS-Security token (WSS SOAP header), you must use HTTP basic authentication.

Sending a SAML Token will allow to send a user with the permission set directly to the application. This is mostly used when customers implement their own authentication and authorization mechanisms and want to authorize the user on the Taleo application.

SAML Token Profile

The following section defines the parameters needed to be exchanged with clients in order to configure the SAML Token Profile 1.1 for securing SOAP messages for the Web services offering.

- **Supported Profile Version**

  Taleo supports SAML Token Profile version 1.1
- **Security**

  The SAML assertion MUST be signed conform to XML-DSIG standard. Taleo supports the following signature algorithm as specified in XML-SIG standard:

  - DSA signature: http://www.w3.org/2000/09/xmldsig#dsa-sha1
  - RSA signature: http://www.w3.org/2000/09/xmldsig#rsa-sha1
  - HMAC with a shared secret key signature: http://www.w3.org/2000/09/xmldsig#hmac-sha1

  The SOAP messages should always be transmitted in a protected channel using SSL or VPN (IPSec).
- **Name identifiers**

  The subject name identifier is the user ID that will be used to retrieve the user who is authenticated with the SAML assertion. SAML defines four different name formats for the `<NameIdentifier>` element which identifies the subject authenticated by the SAML assertion. The name identifier supported formats are those defined by SAML v1.1.

  - **Unspecified**

    URI: urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified.

The interpretation of the content of the <NameQualifier> element is left to individual implementations.

This identifier is used if the subject name is not conform to one of the formats described below. When this identifier is used, Taleo expects to retrieve the subject name as described in the following example:

```
<Subject>
    <NameIdentifier Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:unspecified">
        USERNAME
    </NameIdentifier>
</Subject>
```

- **Email Address**

  URI: urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress

  Indicates that the content of the <NameIdentifier> element is in the form of an email address, specifically "addr-spec" as defined in IETF RFC 2822 §3.4.1. An addr-spec uses the local-part@domain format. Note that an addr-spec has no phrase (such as a common name) before it, has no comment (text surrounded in parentheses) after it, and is not surrounded by "<" and ">".
- **X509 Subject Name**

  URI: urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName

  Indicates that the content of the <NameIdentifier> element is in the form specified for the contents of the <ds:X509SubjectName> element in the XML Signature Recommendation.
- **Windows Domain Qualified Name**

  URI: urn:oasis:names:tc:SAML:1.1:nameid-format:WindowsDomainQualifiedName

  Indicates that the content of the <NameIdentifier> element is a Windows domain qualified name. A Windows domain qualified user name is a string of the form "DomainName\UserName". The domain name and "\" separator MAY be omitted.

- **Parameters**
  - **IDP Alias**

    This is the identity provider that Taleo should trust in order to grant access to Taleo Web services offer. The IDP alias will be retrieved in the Issuer element. The IDP alias value is a string that identifies the SAML Authority who creates the assertions. That value MUST be in all the SAML Assertions that Taleo will receive from the client. This is the key element for Taleo for retrieving the security credentials needed to authenticate the transaction.
  - **Certificate**

    If the signature of the SAML assertion is done with public key algorithm as DSA or RSA algorithm, the certificate of the public key related to the private key used for signing should be provided to Taleo. Taleo will use that certificate in order to validate the trust of the signature.
  - **Secret Key**

    The shared secret key needed to compute the signature in case the algorithm chosen is HMAC-SHA1.
  - **Assertion Audience**

    This is an optional parameter that the client could decide to add in the SAML Assertion in order to raise the security level. This attribute value identifies who can consume the SAML assertion. In this case, the value should be Taleo service provider identifier (a list of the identifiers is defined below). If this element is initialized in the SAML assertion, then Taleo service provider

MUST validate the assertion audience value received in the SAML assertion against the one configured within that Taleo Service provider.

- **Taleo Service Provider Identifiers**
  - Enterprise 7.0, 7.5 (SAML 1.0/1.1; Liberty 1.1/1.2)

Alias	`http://www.taleo.com/professional`
ID	`Base64: rmNFUq7BvVoX7cp1Z0mw0b8rcEk=`
HEX	`ae634552aec1bd5a17edca756749b0d1bf2b7049`
Audience	`http://www.taleo.com/professional`

  - Career Section 7.0 (SAML 1.0/1.1; Liberty 1.1/1.2)

Alias	`http://www.taleo.com/careersection`
ID	`Base64: VtphLHWfjKGcIu4zsbLY9JmKTIk`
HEX	`56da612c759f8ca19c22ee33b1b2d8f4998a4c89`
Audience	`http://www.taleo.com/careersectionp` `Assertion Consumer Service SAML 1.0/1.1: http(s)://hostname/servlets/CareerSection?art_servlet_language=en&art_ip_action=SAML11AssertionConsumerService`

  - OnBoarding 7.0 (SAML 1.0/1.1; Liberty 1.1/1.2)

Alias	`http://www.taleo.com/onboarding`
ID	`Base64: A8y2J5r1oIMj71xR7HHMYOkr75k`
HEX	`N/A`
Audience	`http://www.taleo.com/onboarding` `Assertion Consumer Service SAML 1.0/1.1: http(s)://hostname/onboarding/SamlAssertionConsumerService`

  - Smartorg 7.5 (SAML 1.1)

Alias	`http://www.taleo.com/smartorg`
ID	`Base64: nVTd/8JQSEW2UPGQLYY7pcgTHb8=`
HEX	`9d54ddffc2504845b650f1902d863ba5c8131dbf`
Audience	`http://www.taleo.com/smartorg` `Inter Site Transfer Service SAML 1.1:`

	`http(s)://hostname/smartorg/Saml11InterSiteTransferService.jss`

- **Username Token Profile**
  - **Supported Profile Version**

    Taleo supports Username Token Profile 1.1.
  - **Security**

    Even if both *PasswordText* and *PasswordDigest* password types are supported, Taleo strongly encourages the use of the more secure *PasswordDigest* password type. When using PasswordDigest, the password **must** be SHA1-hashed and Base64-encoded. This is required for the password verification process since this one is never stored in-clear in the Taleo system.

    The SOAP messages should always be transmitted in a protected channel using SSL or VPN (IPSec).

API Reference

Data Model

The Taleo API lets you integrate and extend several Taleo Products. For each product, a data dictionary documents each entity, relations, and services exposed by the product. Because the list of Taleo Products supporting the API will grow over time, the data dictionary documents are not part of this document but can be downloaded separately. If you do not have access to these documents, contact your Customer Representative.

Selection Query Language

Selection Query (or short, SQ-XML) is a proprietary language based on XML that allows you to query data and export entities (fields, relations, etc) exposed in the Data Model of the product(s) used by your organization. The `FindService` Web service uses this XML syntax and is exposed by all Taleo products. The Building Export Queries using Selection Query (SQ-XML) section of this document provides an overview of that syntax. A complete documentation of the SQ-XML can be downloaded separately. If you do not have access to these documents, contact your Customer Representative.

The SQ-XML schema is part of the FindService WSDL File and documents each element and attribute that is part of the schema (see Step 4: Import the WSDL File Into Your Development Platform for detailed instructions). If your SOAP Development Platform or XML Editor supports WSDL and XSD auto-completion, you can get each element and attribute of the Schema associated to the SQ-XML online-documented. Commercial solutions like XML Spy (www.altova.com), Stylus Studio (www.stylusstudio.com) or oXygen (www.oxygenxml.com) provide such an auto-completion feature.

Errors

Internal Server Error

An Internal Server Error (500) occurs when the server encounters an unexpected condition that prevents it from fulfilling a request.

Currently any error without an HTTP status is tagged as Internal Server Error (HTTP error code 500).

Bulk API

Overview

The Bulk API provides load and extract operations to transfer data asynchronously with a Taleo product.

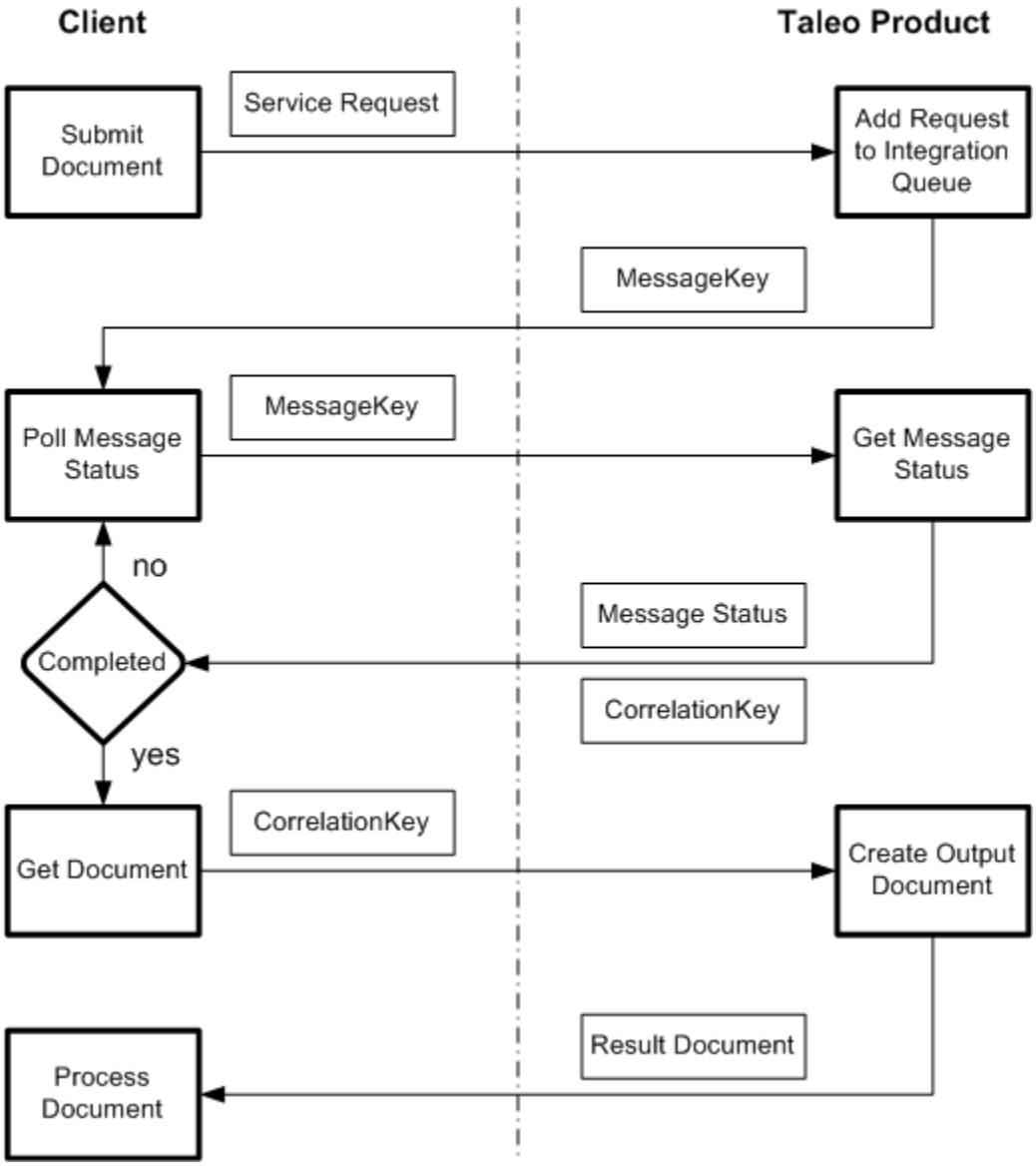
The Process

The Bulk API process applies to data import (load) and export (extract) operations.

The process consists of the following steps:

1. Send a request document to the Taleo product.
2. The response message contains the document identifier assigned by the integration toolkit .
3. Poll product on a regular interval to check the processing status of the document.
4. When processing state is complete, retrieve the result document.

Processing Flow



Taleo Connect Server

The Taleo Connect Server is a module embedded in Taleo products to enable standard system integration processes. Its main purpose is to process messages containing integration instruction such as create, read, update and delete operations (CRUD) on various entities of the business model. Taleo Connect Server does this in an asynchronous manner by first receiving the message and then placing it in a queue for processing when the proper system resources are available.

Following completion of the message, the response may be retrieved from the customer zone using the management web services.

Terminology

Each web service invocation handled by Taleo Connect Server is considered a message.

Messages conveying requests will be managed by Taleo Connect Server; this means that the message information will be completely persisted by Taleo Connect Server for subsequent processing.

The overall message information is divided into multiple message attributes and a single document. The former are information elements related to the communication aspect of the messaging and not the application specific business aspects of the request.

The document represents all information relating to the application specific request being conveyed by the message. In a sense, the document can be viewed as the single (sometimes large) parameter to the Taleo Connect Server service that receives asynchronous requests.

It is further divided into multiple document attributes and single document content. Such a distinction is required as document attributes are used at the request level by the application service, while the document content must support streaming in the context of very large requests. From a purely conceptual point of view, both the document attributes and the document content are parameters to the application service processing the request.

Lexicon (alphabetically)

- Document: Application specific information contained in a managed message.
- Document attribute: Application specific request level information.
- Document content: Application specific raw data.
- Document record: Subdivision of the document content.
- Manage: Behavior of Taleo Connect Server that asynchronously processes messages by temporarily persisting the request until a response has been created then sending back this response to the proper target.
- Message: All information provided as part of the web service invocation.
- Message attribute: Messaging (communication) related information.

WS-Addressing Standard

The WS-Addressing standard has been selected for implementation of the serialization of the message communication information. The choice was motivated by its simplicity in respect to other alternatives, and especially its very tight binding to our actual business needs.

WS-Addressing Implementation

Our implementation of the standard follows the specification as closely as possible, while concentrating on our specific business needs. The only supported Message Exchange Pattern (MEP) is Request-Reply.

- <wsa:MessageID>: A request message MUST have a [message id] property. This value is used to identify the message and correlate it with its corresponding response message.

- <wsa:ReplyTo>: A request message MUST have a [reply endpoint] property. This value indicates how Taleo Connect Server returns the corresponding response message. Currently, only one value is allowed: http://www.taleo.com/ws/integration/toolkit/2005/07/addressing/queue.
- <wsa:Action>: A request message MUST have an [action] property. This value indicates how Taleo Connect Server processes the included document.

Message Processing

At this lowest level, message instances are characterized by several internal information elements:

Name	XML Tag	Description
Message Number	<MessageKey>	A number that identifies a message instance for internal purposes and that is unique for a given Taleo Connect Server instance within a host application.
Message Target	<Target>	A flag indicating how the asynchronous controller should process this message. Possible value is QUEUE (outbound) = 4.
Message State	<State>	The current state of the message. Possible values are NEW=1, INCOMING=2, READY=3, INPROGRESS=4, COMPLETED=5, INTERRUPTED=6, SUSPENDING=7, SUSPENDED=8, INERROR=9, STOPPING=10.
Message Format	<Format>	The format to use for the envelope of the message. Possible values are SOAP11=2 and SOAP12=3.
Correlation Number	<CorrelationKey>	An optional reference to another message number that is created for message instances participating in a request-response Message Exchange Pattern.
Conversation Identifier	<ConversationID>	The identifier of the conversation to which this message is linked. This value has no business significance in this version of Taleo Connect Server.

The state machine flow is:

Name	Description
NEW	A new message instance is created. State =1.
INCOMING	During the creation of the message instance content. State =2.
READY	The message instance is ready for processing by the asynchronous controller. State =3.
INPROGESS	The message is currently being processed by the asynchronous controller. State =4.
COMPLETED	The asynchronous controller has completed the processing of the message instance. State =5.
INTERRUPTED	The message instance was being processed by the asynchronous controller, but the execution was not completed normally. State =6.
SUSPENDING	An external party has requested that the processing of this message instance be suspended. Can only occur if the message instance state is INPROGRESS. State =7.

Name	Description
SUSPENDED	The asynchronous controller has acknowledged the suspension request for this message instance. Can only occur if the message instance state is SUSPENDING. State =8.
INERROR	An unrecoverable error occurred while receiving the message. State =9.
STOPPING	The Taleo product has requested an immediate halt of this message processing for reasons independent of the Taleo Connect Server. State =10.

Any incoming message received by Taleo Connect Server is first created in a NEW state. During the streaming of the document content, the state is set to INCOMING. Finally, when the reception is done, the message is placed in a READY state.

Any messages present in the Taleo Connect Server queue in a READY state are eligible for processing by the asynchronous controller. The selection priority is given to outbound messages over inbound messages and then according to a FIFO logic.

Finally, the asynchronous controller simply sets the message state to COMPLETED and leaves the document in place for retrieval via web service at a later time.

Error Handling

There are three contexts to consider when discussing the implementation of error handling in the Taleo Connect Server.

1. **Web Service Errors:** Errors occurring during the invocation of a web service.
2. **Application Service Errors:** Application level business errors occurring during the asynchronous processing.
3. **Unexpected Technical Errors:** Unexpected technical errors occurring during the asynchronous controller activities.

The following information elements may be present in the error.

Name	SOAP 1.1 XML Tag	SOAP 1.2 XML Tag	Taleo XML (TXML)	Description
Error code	<faultcode>	<Code>	<Code>	Flag indicating if the error is due to the sender or the receiver of the request.
Error subcode	N/A	<SubCode>	<SubCode>	A language neutral identifier that is currently implemented as a fully-qualified element representing an English abbreviation of the general classification of the error. This is the same implementation as most of the WS-* standards.
Error message	<faultstring>	<Reason>	<Reason>	An English human-readable message that varies depending on the actual instance of the error.
Detail Message	<detail>	<Detail>	<Detail>	Detail information about the error.

Web Service Errors

In this context, any thrown exceptions are ultimately handled by the web service framework and are serialized as proper SOAP faults.

When the exception is thrown or handled by Taleo Connect Server itself, then the serialization looks like the following in SOAP version 1.1:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:Server</faultcode>
         <faultstring>Could not read XML stream.. Nested exception is
         com.ctc.wstx.exc.WstxUnexpectedCharException: Unexpected
         character '&lt;' (code 60) expected space, or '>' or "/>"
         at [row,col {unknown-source}]: [6,126]</faultstring>
         <detail>
            <WebServiceFault xmlns="http://www.taleo.com/ws/integration
            /toolkit/2005/07">
               <code>SystemError</code>
               <message>Unexpected character '&lt;' (code 60)
```

```
               expected space, or '>' or "/>"
               at [row,col {unknown-source}]: [6,126]</message>
            </WebServiceFault>
         </detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

And like this in SOAP version 1.2:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soap:Body>
      <soap:Fault>
         <soap:Code>
            <soap:Value>soap:Receiver</soap:Value>
            <soap:SubCode>
               <soap:Value xmlns:ns1="http://www.taleo.com/soap/
               fault">ns1:SystemError</soap:Value>
            </soap:SubCode>
         </soap:Code>
         <soap:Reason>
            <soap:Text>Could not read XML stream.. Nested exception is
            com.ctc.wstx.exc.WstxUnexpectedCharException: Unexpected
            character '&lt;' (code 60) expected space, or '>' or "/>"
            at [row,col {unknown-source}]: [6,126]</soap:Text>
         </soap:Reason>
         <soap:Detail>
            <WebServiceFault xmlns="http://www.taleo.com/ws/integration/
            toolkit/2005/07">
               <code>SystemError</code>
               <message>Unexpected character '&lt;' (code 60)
               expected space, or '>' or "/>"
               at [row,col {unknown-source}]: [6,126]</message>
            </WebServiceFault>
         </soap:Detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

Application Service Errors

When business or technical errors occur during the processing of the messages, the serialization is slightly different but contains the same information.

In the case of a handled error, a message will accompany the unique subcode to give the context of the problem.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
  <ns1:getDocumentByKeyResponse xmlns:ns1="http://www.taleo.com/ws/
  integration/toolkit/2011/05/management">
   <Document xmlns="http://www.taleo.com/ws/integration/toolkit/2011/05">
    <Attributes>
     <Attribute name="duration">0:00:00.019</Attribute>
     <Attribute name="count">0</Attribute>
     <Attribute name="mode">T-XML</Attribute>
     <Attribute name="version">http://www.taleo.com/ws/itk/
     prototype/2006/05</Attribute>
    </Attributes>
```

```
    <Content>
     <ExportErrors>
      <Error>
       <Code>ServerError</Code>
       <Subcode>itk:ExportBadSelectionQueryFormat</Subcode>
       <Reason/>
       <Detail>The unique identifier of the exception is
       6d6b9f00-8e12-11e0-885f-877305dceb5a</Detail>
      </Error>
     </ExportErrors>
    </Content>
   </Document>
  </ns1:getDocumentByKeyResponse>
 </soap:Body>
</soap:Envelope>
```

Unexpected Technical Errors

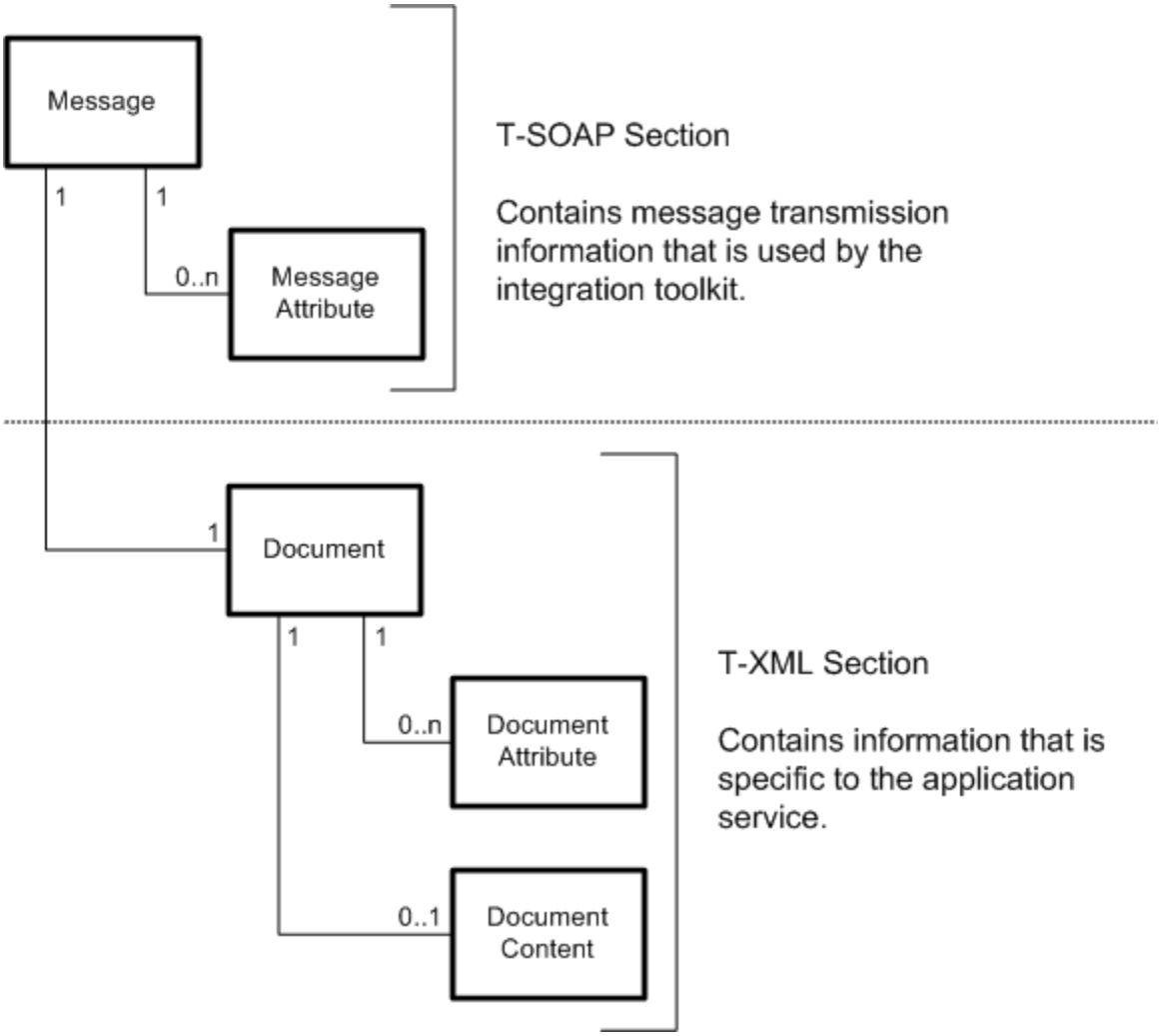
For unexpected technical errors, there is no way for Taleo Connect Server to provide the information directly to the customer. Indeed, these errors occur when:

- A communication failure occurs.
- A database failure occurs.
- A JVM failure occurs (out-of-memory, no more threads).
- An internal Taleo Connect Server error (that is NOT a business related error).

In the above cases, on a best effort basis, the same error information is serialized to the message itself. This situation will be reflected by the state of the message in Taleo Connect Server (interrupted).

Taleo Connect Server management service exposes this information to external clients. Of course, if the database itself is down, then only the application logging will be available for diagnostic. This being said, when such a production incident happens, it is usually possible to correlate the system failure with the actual integration process problem.

Format of the request and result message



Request Message Example

```
<?xml version="1.0" encoding="UTF-8"?>

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:ns1="http://www.taleo.com/ws/integration/toolkit/2011/05/
management" xmlns:ns="http://www.taleo.com/ws/integration/toolkit/2011/05"
 xmlns:wsa="http://www.w3.org/2005/03/addressing">

<!--Start of Message-->
    <soapenv:Header>

        <!--Start of Message Attributes -->
        <wsa:MessageID>MSG0f4457ea-a865-4948-a675-d60fd1e48494</
wsa:MessageID>

        <wsa:ReferenceParameters>
            <!-- XXX -->
            <wsa:xPARAMx>xPARAMxVALUEx</wsa:xPARAMx>
        </wsa:ReferenceParameters>
```

```
        <wsa:ReplyTo>
            <wsa:Address>http://www.taleo.com/ws/integration/
toolkit/2005/07/addressing/queue</wsa:Address>
        </wsa:ReplyTo>

        <wsa:Action>http://www.taleo.com/ws/integration/toolkit/2005/07/
action/export</wsa:Action>
        <!--End of Message Attributes -->

    </soapenv:Header>


    <soapenv:Body>

        <!--Start of Integration Management Service Operation -->
        <ns1:submitDocument>

            <!--Start of Document -->
            <ns:Document>

                <!--Start of Document Attributes -->
                <ns:Attributes>
                    <ns:Attribute name="mode">T-XML</ns:Attribute>
                    <ns:Attribute name="version">http://www.taleo.com/ws/
itk/prototype/2006/05</ns:Attribute>
                </ns:Attributes>
                <!--End of Document Attributes -->


               <!--Start of Document Content -->
                <ns:Content>
                    <ExportQuery xmlns="http://www.taleo.com/ws/
integration/toolkit/2005/07/action/export">
                        <query projectedClass="Character" alias="Character"
 xmlns="http://itk.taleo.com/ws/query">
                            <projections>
                                <projection>
                                    <field path="Number"/>
                                </projection>
                                <projection>
                                    <field path="Character.Name"/>
                                </projection>
                            </projections>
                            <projectionSortings>
                                <projectionSorting>
                                    <field path="Quests,Number"/>
                                </projectionSorting>
                                <projectionSorting ascending="true">
                                    <field path="otherArmorPieces,Number"/>
                                </projectionSorting>
                            </projectionSortings>
                            <filterings>
                                <filtering>
                                    <lessThanOrEqual>
                                        <field path="Number"/>
                                        <string>9001004</string>
                                    </lessThanOrEqual>
                                </filtering>
                            </filterings>
                        </query>
                    </ExportQuery>
```

```
                </ns:Content>
               <!--End of Document Content -->

            </ns:Document>
            <!--End of Document -->

        </ns1:submitDocument>
        <!--End of Integration Management Service Operation -->

    </soapenv:Body>
<!--End of Message -->

</soapenv:Envelope>
```

Management Service

The management service API interacts with the integration messages within Taleo Connect Server.

WSDL

The ManagementService WSDL is available at the following URL: [protocol]//[zone]/[product]/soap?ServiceName=IntegrationManagementService&wsdl

Example: https://myzone.taleo.net/smartorg/soap?ServiceName=IntegrationManagementService&wsdl

Version 12B Changes

Some of the changes made in version 12B are breaking backwards compatibility. The impacts of the changes and new features are described below.

submitLargeDocument (Send)

Operation submitLargeDocument is introduced in version 12B. More information about the new operation can be found in section submitLargeDocument (Send).

Note: There is no impact when migrating to version 12B.

New Schema

To load an import result as an object a schema has been added to the WSDL.

```
xmlns:ai=http://www.taleo.com/ws/integration/toolkit/2005/07/action/import
```

```
 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 attributeFormDefault="unqualified" elementFormDefault="qualified"
 targetNamespace="http://www.taleo.com/ws/integration/toolkit/2005/07/
action/import">
   <xsd:element name="ImportResult">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="Record" type="ai:Record"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   <xsd:complexType name="Record">
    <xsd:sequence>
     <xsd:element name="Index" type="xsd:integer"/>
     <xsd:element name="Identifier" type="xsd:string"/>
     <xsd:element name="Status" type="xsd:string"/>
     <xsd:element name="TransactionType" type="xsd:string"/>
     <xsd:element name="ExceptionInfo" type="ai:ExceptionInfo"/>
     <xsd:element name="Return" type="ai:Return"/>
     <xsd:element name="Errors" type="ai:Errors" minOccurs="0"/>
     <xsd:element name="Warnings" type="ai:Warnings" minOccurs="0"/>
    </xsd:sequence>
   </xsd:complexType>
   <xsd:element name="Record" type="ai:Record"/>
   <xsd:complexType name="ExceptionInfo">
    <xsd:sequence>
     <xsd:element name="ExceptionMessage" type="xsd:string"/>
     <xsd:element name="ExceptionStackTrace" type="xsd:string"/>
    </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Return">
    <xsd:sequence>
     <xsd:any namespace="##any" minOccurs="0" maxOccurs="1"/>
```

```
    </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Errors">
    <xsd:sequence>
     <xsd:element name="Error" type="xsd:string" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Warnings">
    <xsd:sequence>
     <xsd:element name="Warning" type="xsd:string" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:complexType>
 </xsd:schema>
```

Note: There is no impact when migrating to version 12B.

.NET Streaming

To enable streaming with .NET another namespace has been included and a new type has been defined in version 12B. This is used to stream the content instead of having to load it in memory. When using wsdl2java to generate the JAVA code, no reference is made to the .NET namespace and a JAVA object is created.

```
<xsd:schema xmlns:tns="http://schemas.microsoft.com/Message"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
targetNamespace="http://schemas.microsoft.com/Message">
  <xs:simpleType name="StreamBody">
   <xs:restriction base="xs:base64Binary"/>
  </xs:simpleType>
</xsd:schema>
```

Note: There is no impact when migrating to version 12B.

getLargeDocumentByKeyResponse

The getLargeDocumentByKeyResponse operation takes a Document attribute instead of a File, and the type has been changed from base64Binary to StreamBody. However, the content remains the same.

Before 12B

```
<xsd:element name="getLargeDocumentByKeyResponse">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element maxOccurs="1" minOccurs="1" name="File"
nillable="true" type="mtom:base64Binary"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

12B

```
<xsd:element name="getLargeDocumentByKeyResponse">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element maxOccurs="1" minOccurs="1" name="Document"
nillable="true" type="nsm:StreamBody"
nsxmlmime:expectedContentTypes="application/octet-stream"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
```

Note: Change will be needed in the web services consumer code. When migrating to a 12B version, it will now be necessary to regenerate the stub and instantiate a new Document object instead of a File object.

attributeFormDefault

The attribute “attributeFormDefault” has been set to “unqualified” instead of “qualified”. This means that the attribute from the target namespace must not be qualified with a namespace. This change has been made since the IntegrationToolkit expects an unqualified element but the old WSDL was set with a qualified attribute instead. In version 12B, the default attribute has been adjusted to be consistent with the ITK. Here is an example of the attribute:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 attributeFormDefault="unqualified" elementFormDefault="qualified"
 targetNamespace="http://www.taleo.com/ws/integration/toolkit/2011/05">
```

Note: It will be necessary to remove any occurrence of a qualified attribute for an element and replace it by an unqualified attribute when migrating to version 12B.

CompressedContent

The element “CompressedContent” that was available in the previous versions has been removed in version 12B.

```
<xsd:element name="CompressedContent" type="xsd:string">
 <xsd:annotation>
  <xsd:documentation>The document content in a base 64 encoded zipped
 representation.</xsd:documentation>
  </xsd:annotation>
 </xsd:element>
```

Note: Element “CompressedContent” must be removed when migrating to version 12B.

Attributes Added

ApplicationCode, OwnerUserName, AttributeType have been added.

```
<xsd:element name="ApplicationCode" type="xsd:string" minOccurs="0"/>
 <xsd:element name="OwnerUserName" type="xsd:string" minOccurs="0"/>
 <xsd:element name="AttributeType" type="xsd:string"/>
```

Note: There is no impact when migrating to version 12B.

Attributes Modified

To resolve a reflection problem with .NET, MessageID and Action attributes have been modified.

Before 12B

```
<xsd:element name="MessageID" type="wsa:AttributedURIType"/>
```

```
<xsd:element name="Action" type="wsa:AttributedURIType"/>
```

12B

```
<xsd:element name="MessageID" type="wsa:TaleoMessageID"/>
 <xsd:complexType name="TaleoMessageID">
  <xsd:simpleContent>
   <xsd:extension base="wsa:AttributedURIType"/>
  </xsd:simpleContent>
 </xsd:complexType>
```

```
<xsd:element name="Action" type="wsa:TaleoAction"/>
 <xsd:complexType name="TaleoAction">
  <xsd:simpleContent>
```

```
  <xsd:extension base="wsa:AttributedURIType"/>
 </xsd:simpleContent>
</xsd:complexType>
```

Note: Change will be needed in the web services consumer code. When migrating to version 12B, it will now be necessary to regenerate the stub and instantiate a new TaleoMessageID instead of a MessageID and a new TaleoAction instead of an Action.

Prefix

The following prefixes are used to identify elements:

Prefix	Namespace	Usage
ai	http://www.taleo.com/ws/integration/toolkit/2005/07/action/import	Import result document structure.
nsxmlmime	http://www.w3.org/2005/05/xmlmime	Mime type of attachments use to transport large content
ns2005_07	http://www.taleo.com/ws/integration/toolkit/2005/07	ITK namespace containing the standard definition of web service fault.
ns2011_05	http://www.taleo.com/ws/integration/toolkit/2011/05	ITK namespace containing the definition of the bulk API message structure
nsm	http://schemas.microsoft.com/Message	Required to be compatible with .NET C# streaming
tns	http://www.taleo.com/ws/integration/toolkit/2011/05/management	Integration Management Service Namespace
wsa	http://www.w3.org/2005/03/addressing	WS-Addressing
wsdlsoap	http://schemas.xmlsoap.org/wsdl/soap/	SOAP Binding
xsd	http://www.w3.org/2001/XMLSchema	XML Schema namespace

submitDocument (Send)

Description

This operation allows a user to submit a document to the asynchronous job controller. This message must contain a **small** integration document (less than 50kb). If it is too big, the content of the document will not be streamed by the web service stack (client and server side).

Syntax

ns2011_05:IntegrationMessage : submitDocument(wsa:MessageID, wsa:ReplyTo, wsa:Action, ns2011_05:Document)

Direction	SOAP Part	Name
In	Header	wsa:MessageID
In	Header	wsa:ReplyTo
In	Header	wsa:Action
In	Body	ns2011_05:Document
Out	Body	ns2011_05:IntegrationMessage

Table 1: Request Parameters

Name	Description
wsa:MessageID	This value is used to identify the message and to correlate it with its corresponding response message.
wsa:Action	This value indicates how Taleo Connect Server processes the included document.
wsa:ReplyTo	This value indicates how Taleo Connect Server returns the corresponding response message. Currently, only one value is allowed: http://www.taleo.com/ws/integration/toolkit/2005/07/addressing/queue
ns2011_05:Document	A document is a generic container that holds the information elements of the message. Type: Complex Type
ns2011_05:Document.Attributes	The attributes represent document level information elements. Type: Complex Type
ns2011_05:Document.Attributes.Attribute	An attribute is a document level information element. Type: Complex Type
ns2011_05:Document.Attributes.Attribute.name	The attribute name uniquely identifies it among the attributes collection of a document. Type: xsd:string
ns2011_05:Document.Content	The document content represents the record level information elements. This type of information in the document content is dependent on the processing context. As specified by the schema, the Content element always contains exactly one child element which is the root of the content. Each immediate child of this element is considered a record in the Taleo Connect terminology.

Name	Description
	Type: xsd:any

Table 2: Response Elements

Name	Description
ns2011_05:IntegrationMessage	An integration message is the main entity handled by the Taleo Connect technology. Managed messages represent a wrapper around an integration document containing the actual integration instructions that specifies the connectivity and communication information. Type: Complex Type
ns2011_05:IntegrationMessage.MessageKey	The message key is a globally unique identifier generated by the Taleo Connect Server that is only to be used internally within the Taleo product instance. Type: xsd:integer
ns2011_05:IntegrationMessage.CorrelationKey	The correlation key is the message key of the related message in integration process (request or response depending on the nature of the message), this is only to be used internally within the Taleo product instance. Type: xsd:integer
ns2011_05:IntegrationMessage.MessageID	The message identifier is an external driven identifier for the message that is provided by the caller for request messages and generated by the Taleo Connect server for response type messages. When generated by the Taleo Connect server, the identifier value is guaranteed to be globally unique. The same restriction should apply when a caller supplies such a value. Type: xsd:string
ns2011_05:IntegrationMessage.ConversationID	The conversation identifier is an additional identifier offered to link message together in a non-restrictive manner. There are no business rules attached to this value in the Taleo Connect server other than to set this value with the value of the request message conversation identifier when creating a response message (if such a value is present). Type: xsd:string
ns2011_05:IntegrationMessage.State	The message state indicates its processing state within the Taleo Connect server. Type: xsd:integer
ns2011_05:IntegrationMessage.Target	The message target indicates how the Taleo Connect server is to process the message. The specific values are described in the main Taleo Connect documentation. Type: xsd:integer

Name	Description
ns2011_05:IntegrationMessage.Format	The message format indicates the envelope format of the message. The specific values are described in the main Taleo Connect documentation. Type: xsd:integer
ns2011_05:IntegrationMessage.ServiceName	Type: xsd:string
ns2011_05:IntegrationMessage.MethodName	Type: xsd:string
ns2011_05:IntegrationMessage.ServiceID	Type: xsd:string
ns2011_05:IntegrationMessage.ApplicationCode	For managed documents, this value indicates the business service invoked when processing the integration document. It is the value provided in the WS-Addressing action property. Type: xsd:string
ns2011_05:IntegrationMessage.Managed	The managed flag indicates whether a message contains an integration document. Type: xsd:boolean
ns2011_05:IntegrationMessage.CreationDate	Type: xsd:dateTime
ns2011_05:IntegrationMessage.LastModificationDate	Type: xsd:dateTime
ns2011_05:IntegrationMessage.Document	The document is a high-level representation of the integration document, except for the actual content. Type: complex type (see structure below)
ns2011_05:IntegrationMessage.Document.RecordCount	The total number of integration instructions present in the message (instructions can be viewed as results when the message is a response). Type: xsd:integer
ns2011_05:IntegrationMessage.Document. RecordIndex	The total number of instructions currently processed through the Taleo Connect server. The range of this value is 0 to n, where n is the RecordCount value. Type: xsd:integer
ns2011_05:IntegrationMessage.Document.Attributes.Name	Type: xsd:string
ns2011_05:IntegrationMessage.Document.Attributes.Value	Type: xsd:string

Name	Description
ns2011_05:IntegrationMessage. Attributes.Attribute.Namespace	Type: xsd:string
ns2011_05:IntegrationMessage. Attributes.Attribute.Name	Type: xsd:string
ns2011_05:IntegrationMessage. Attributes.Attribute.Value	Type: xsd:string
ns2011_05:IntegrationMessage. HistoryItems.HistoryItem.Sequence	The sequence of the history item. Type: xsd:integer
ns2011_05:IntegrationMessage. HistoryItems.HistoryItem.InitialState	Refers to the message state. Type: xsd:integer
ns2011_05:IntegrationMessage. HistoryItems.HistoryItem.FinalState	Refers to the message state. Type: xsd:integer
ns2011_05:IntegrationMessage. HistoryItems.HistoryItem.UserName	Type: xsd:string
ns2011_05:IntegrationMessage. HistoryItems.HistoryItem. CreationDate	Type: xsd:dateTime

Example:

Request

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
soap/envelope/" xmlns:ns1="http://www.taleo.com/ws/integration/
toolkit/2011/05/management" xmlns:ns2011_05="http://www.taleo.com/
ws/integration/toolkit/2011/05" xmlns:wsa="http://www.w3.org/2005/03/
addressing">
    <soapenv:Header>
        <wsa:MessageID>MSG0f4457ea-a865-4948-a675-d60fd1e48494</
wsa:MessageID>
        <wsa:ReferenceParameters>
            <!-- XXX -->
            <wsa:xPARAMx>xPARAMxVALUEx</wsa:xPARAMx>
        </wsa:ReferenceParameters>
        <wsa:ReplyTo>
            <wsa:Address>http://www.taleo.com/ws/integration/
toolkit/2005/07/addressing/queue</wsa:Address>
        </wsa:ReplyTo>
        <wsa:Action>http://www.taleo.com/ws/integration/toolkit/2005/07/
action/export</wsa:Action>
    </soapenv:Header>
    <soapenv:Body>
        <ns1:submitDocument>
            <ns2011_05:Document>
                <ns2011_05:Attributes>
```

```
                    <ns2011_05:Attribute name="mode">T-XML</
ns2011_05:Attribute>
                    <ns2011_05:Attribute name="version">http://
www.taleo.com/ws/itk/prototype/2006/05</ns2011_05:Attribute>
                </ns2011_05:Attributes>
                <ns2011_05:Content>
                    <ExportQuery xmlns="http://www.taleo.com/ws/
integration/toolkit/2005/07/action/export">
                        <query projectedClass="Character"
 alias="Character" xmlns="http://itk.taleo.com/ws/query">
                            <projections>
                                <projection>
                                    <field path="Number"/>
                                </projection>
                                <projection>
                                    <field path="Character.Name"/>
                                </projection>
                            </projections>
                            <projectionSortings>
                                <projectionSorting>
                                    <field path="Quests,Number"/>
                                </projectionSorting>
                                <projectionSorting ascending="true">
                                    <field
 path="otherArmorPieces,Number"/>
                                </projectionSorting>
                            </projectionSortings>
                            <filterings>
                                <filtering>
                                    <lessThanOrEqual>
                                        <field path="Number"/>
                                        <string>9001004</string>
                                    </lessThanOrEqual>
                                </filtering>
                            </filterings>
                        </query>
                    </ExportQuery>
                </ns2011_05:Content>
            </ns2011_05:Document>
        </ns1:submitDocument>
    </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
  <ns1:submitDocumentResponse xmlns:ns1="http://www.taleo.com/ws/
  integration/toolkit/2011/05/management">
   <IntegrationMessage xmlns="http://www.taleo.com/ws/
   integration/toolkit/2011/05">
    <MessageKey>1031501710688</MessageKey>
   </IntegrationMessage>
  </ns1:submitDocumentResponse>
 </soap:Body>
</soap:Envelope>
```

submitLargeDocument (Send)

Description

The submitLargeDocument operation allows a User to submit a document to the asynchronous job controller.

The Document element in the SOAP body is of type base64Binary. If the size of the document is less than 50 kb, you can directly put its base64 encoded content in the Document element. If you send a larger one, it will be refused by the TCS.

The Document can also be sent in attachment using the MTOM attachment (Message Transmission Optimization Mechanism) technology. With an attachment you are not limited to 50kb so make sure to stream your document instead of loading it in memory. On the TCS side, the document will be directly streamed to the TCS persistence storage.

Syntax

ns2011_05:IntegrationMessage : submitLargeDocument(wsa:MessageID, wsa:ReplyTo, wsa:Action, tns:Document)

Direction	SOAP Part	Name
In	Header	wsa:MessageID
In	Header	wsa:ReplyTo
In	Header	wsa:Action
In	Body	tns:Document
Out	Body	ns2011_05:IntegrationMessage

Table 3: Request Parameters

Name	Description
wsa:MessageID	This value is used to identify the message and to correlate it with its corresponding response message.
wsa:ReplyTo	This value indicates how Taleo Connect Server is to process the included document.
wsa:Action	This value indicates how Taleo Connect Server is to return the corresponding response message. Currently, only one value is allowed: http://www.taleo.com/ws/integration/toolkit/2005/07/addressing/queue
tns:Document	This element contains either a base64 encoded ns2011_05:Document or a reference to a MTOM attachment. Type: nsm:StreamBody

Table 4: ns2011_05:Document structure

Name	Description
ns2011_05:Document	A document is a generic container that holds the information elements of the message.

Name	Description
	Type: Complex Type
ns2011_05:Document.Attributes	The attributes represent document level information elements. Type: Complex Type
ns2011_05:Document.Attributes.Attribute	An attribute is a document level information element. Type: Complex Type
ns2011_05:Document.Attributes.Attribute.name	The attribute name uniquely identifies it among the attributes collection of a document. Type: xsd:string
ns2011_05:Document.Content	The document content represents the record level information elements. This type of information in the document content is dependent on the processing context. As specified by the schema, the Content element always contains exactly one child element which is the root of the content. Each immediate child of this element is considered a record in the Taleo Connect terminology. Type: xsd:any

Table 5: Response Elements

Name	Description
ns2011_05:IntegrationMessage	An integration message is the main entity handled by the Taleo Connect technology. Managed messages represent a wrapper around an integration document containing the actual integration instructions that specifies the connectivity and communication information. Type: Complex Type
ns2011_05:IntegrationMessage.MessageKey	The message key is a globally unique identifier generated by the Taleo Connect Server that is only to be used internally within the Taleo product instance. Type: xsd:integer
ns2011_05:IntegrationMessage.CorrelationKey	The correlation key is the message key of the related message in integration process (request or response depending on the nature of the message), this is only to be used internally within the Taleo product instance. Type: xsd:integer
ns2011_05:IntegrationMessage.MessageID	The message identifier is an external driven identifier for the message that is provided by the caller for request messages and generated by the Taleo Connect server for response type messages. When generated by the Taleo Connect server, the identifier value is guaranteed to be globally unique. The same restriction should apply when a caller supplies such a value.

Name	Description
	Type: xsd:string
ns2011_05:IntegrationMessage.ConversationID	The conversation identifier is an additional identifier offered to link message together in a non-restrictive manner. There are no business rules attached to this value in the Taleo Connect server other than to set this value with the value of the request message conversation identifier when creating a response message (if such a value is present). Type: xsd:string
ns2011_05:IntegrationMessage.State	The message state indicates its processing state within the Taleo Connect server. Type: xsd:integer
ns2011_05:IntegrationMessage.Target	The message target indicates how the Taleo Connect server is to process the message. The specific values are described in the main Taleo Connect documentation. Type: xsd:integer
ns2011_05:IntegrationMessage.Format	The message format indicates the envelope format of the message. The specific values are described in the main Taleo Connect documentation. Type: xsd:integer
ns2011_05:IntegrationMessage.ServiceName	Type: xsd:string
ns2011_05:IntegrationMessage.MethodName	Type: xsd:string
ns2011_05:IntegrationMessage.ServiceID	Type: xsd:string
ns2011_05:IntegrationMessage.ApplicationCode	For managed documents, this value indicates the business service invoked when processing the integration document. It is the value provided in the WS-Addressing action property. Type: xsd:string
ns2011_05:IntegrationMessage.Managed	The managed flag indicates whether a message contains an integration document. Type: xsd:boolean
ns2011_05:IntegrationMessage.CreationDate	Type: xsd:dateTime
ns2011_05:IntegrationMessage.LastModificationDate	Type: xsd:dateTime
ns2011_05:IntegrationMessage.Document	The document is a high-level representation of the integration document, except for the actual content.

Name	Description
	Type: complex type (see structure below)
ns2011_05:IntegrationMessage.Document.RecordCount	The total number of integration instructions present in the message (instructions can be viewed as results when the message is a response). Type: xsd:integer
ns2011_05:IntegrationMessage.Document.RecordIndex	The total number of instructions currently processed through the Taleo Connect server. The range of this value is 0 to n, where n is the RecordCount value. Type: xsd:integer
ns2011_05:IntegrationMessage.Document.Attributes.Name	Type: xsd:string
ns2011_05:IntegrationMessage.Document.Attributes.Value	Type: xsd:string
ns2011_05:IntegrationMessage.Attributes.Attribute.Namespace	Type: xsd:string
ns2011_05:IntegrationMessage.Attributes.Attribute.Name	Type: xsd:string
ns2011_05:IntegrationMessage.Attributes.Attribute.Value	Type: xsd:string
ns2011_05:IntegrationMessage.HistoryItems.HistoryItem.Sequence	The sequence of the history item. Type: xsd:integer
ns2011_05:IntegrationMessage.HistoryItems.HistoryItem.InitialState	Refers to the message state. Type: xsd:integer
ns2011_05:IntegrationMessage.HistoryItems.HistoryItem.FinalState	Refers to the message state. Type: xsd:integer
ns2011_05:IntegrationMessage.HistoryItems.HistoryItem.UserName	Type: xsd:string
ns2011_05:IntegrationMessage.HistoryItems.HistoryItem.CreationDate	Type: xsd:dateTime

Example:

Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
soap/envelope/" xmlns:add="http://www.w3.org/2005/03/addressing"
 xmlns:ns="http://www.taleo.com/ws/integration/toolkit/2011/05/
management" xmlns:xm="http://www.w3.org/2005/05/xmlmime">
```

```
   <soapenv:Header>
      <add:MessageID>${msgId}</add:MessageID>
      <add:ReplyTo>
         <add:ReferenceParameters>
            <add:xPARAMx>xPARAMxVALUEx</add:xPARAMx>
         </add:ReferenceParameters>
         <add:Address>http://www.taleo.com/ws/integration/
toolkit/2005/07/addressing/queue</add:Address>
      </add:ReplyTo>
      <add:Action>http://www.taleo.com/ws/integration/toolkit/2005/07/
action/import</add:Action>
   </soapenv:Header>
   <soapenv:Body>
      <ns:submitLargeDocument>
         <ns:Document xm:contentType="application/?">cid:
${Properties#newCid}</ns:Document>
      </ns:submitLargeDocument>
   </soapenv:Body>
</soapenv:Envelope>
```

Example:

Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
   <soap:Body>
      <ns1:submitLargeDocumentResponse xmlns:ns1="http://www.taleo.com/
ws/integration/toolkit/2011/05/management">
         <IntegrationMessage xmlns="http://www.taleo.com/ws/integration/
toolkit/2011/05">
            <MessageKey>38101791958</MessageKey>
         </IntegrationMessage>
      </ns1:submitLargeDocumentResponse>
   </soap:Body>
</soap:Envelope>
```

getMessageByKey (Poll)

Description

Returns a complete report on the status of a message providing, among others, the key and identifier of the message, its current state, its state history and the document information.

Syntax

ns2011_05:IntegrationMessage : getMessageByKey(tns:messageKey)

Direction	SOAP Part	Name
In	Body	tns:messageKey
Out	Body	ns2011_05:IntegrationMessage

Table 6: Request Parameters

Name	Description
tns:messageKey	Key of the message as returned by the SubmitDocument operation response.

Name	Description
	Type: xsd:string Default:

Table 7: Response Elements

Name	Description
ns2011_05:IntegrationMessage	An integration message is the main entity handled by the Taleo Connect technology. Managed messages represent a wrapper around an integration document containing the actual integration instructions that specifies the connectivity and communication information. Type: Complex Type
ns2011_05:IntegrationMessage.MessageKey	The message key is a globally unique identifier generated by the Taleo Connect Server that is only to be used internally within the Taleo product instance. Type: xsd:integer
ns2011_05:IntegrationMessage.CorrelationKey	The correlation key is the message key of the related message in integration process (request or response depending on the nature of the message), this is only to be used internally within the Taleo product instance. Type: xsd:integer
ns2011_05:IntegrationMessage.MessageID	The message identifier is an external driven identifier for the message that is provided by the caller for request messages and generated by the Taleo Connect server for response type messages. When generated by the Taleo Connect server, the identifier value is guaranteed to be globally unique. The same restriction should apply when a caller supplies such a value. Type: xsd:string
ns2011_05:IntegrationMessage.ConversationID	The conversation identifier is an additional identifier offered to link message together in a non-restrictive manner. There are no business rules attached to this value in the Taleo Connect server other than to set this value with the value of the request message conversation identifier when creating a response message (if such a value is present). Type: xsd:string
ns2011_05:IntegrationMessage.State	The message state indicates its processing state within the Taleo Connect server. Type: xsd:integer
ns2011_05:IntegrationMessage.Target	The message target indicates how the Taleo Connect server processes the message. The specific values are described in the main Taleo Connect documentation. Type: xsd:integer

Name	Description
ns2011_05:IntegrationMessage.Format	The message format indicates the envelope format of the message. The specific values are described in the main Taleo Connect documentation. Type: xsd:integer
ns2011_05:IntegrationMessage.ServiceName	Type: xsd:string
ns2011_05:IntegrationMessage.MethodName	Type: xsd:string
ns2011_05:IntegrationMessage.ServiceID	Type: xsd:string
ns2011_05:IntegrationMessage.ApplicationCode	For managed documents, this value indicates the business service invoked when processing the integration document. It is the value provided in the WS-Addressing action property. Type: xsd:string
ns2011_05:IntegrationMessage.Managed	The managed flag indicates whether a message contains an integration document. Type: xsd:boolean
ns2011_05:IntegrationMessage.CreationDate	Type: xsd:dateTime
ns2011_05:IntegrationMessage.LastModificationDate	Type: xsd:dateTime
ns2011_05:IntegrationMessage.Document	The document is a high-level representation of the integration document, except for the actual content. Type: complex type (see structure below) Type: complex type (see structure below)
ns2011_05:IntegrationMessage.Document.RecordCount	The total number of integration instructions present in the message (instructions can be viewed as results when the message is a response). Type: xsd:integer
ns2011_05:IntegrationMessage.Document.RecordIndex	The total number of instructions currently processed through the Taleo Connect server. The range of this value is 0 to n, where n is the RecordCount value. Type: xsd:integer
ns2011_05:IntegrationMessage.Document.Attributes.Name	Type: xsd:string
ns2011_05:IntegrationMessage.Document.Attributes.Value	Type: xsd:string

Name	Description
ns2011_05:IntegrationMessage. Attributes.Attribute.Namespace	Type: xsd:string
ns2011_05:IntegrationMessage. Attributes.Attribute.Name	Type: xsd:string
ns2011_05:IntegrationMessage. Attributes.Attribute.Value	Type: xsd:string
ns2011_05:IntegrationMessage. HistoryItems.HistoryItem.Sequence	The sequence of the history item. Type: xsd:integer
ns2011_05:IntegrationMessage. HistoryItems.HistoryItem.InitialState	Refers to the message state. Type: xsd:integer
ns2011_05:IntegrationMessage. HistoryItems.HistoryItem.FinalState	Refers to the message state. Type: xsd:integer
ns2011_05:IntegrationMessage. HistoryItems.HistoryItem.UserName	Type: xsd:string
ns2011_05:IntegrationMessage. HistoryItems.HistoryItem. CreationDate	Type: xsd:dateTime

Example:

Request

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:ns="http://www.taleo.com/ws/integration/
toolkit/2011/05/management">
 <soapenv:Header/>
 <soapenv:Body>
  <ns:getMessageByKey soapenv:encodingStyle="http://schemas.xmlsoap.org/
  soap/encoding/">
   <messageKey xsi:type="xsd:string" xs:type="type:string"
   xmlns:xs="http://www.w3.org/2000/XMLSchema-instance">38301791352
   </messageKey>
  </ns:getMessageByKey>
 </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
```

```
  <ns1:getMessageByKeyResponse xmlns:ns1="http://www.taleo.com/ws/
  integration/toolkit/2011/05/management">
   <IntegrationMessage xmlns="http://www.taleo.com/ws/integration/
toolkit/2011/05">
    <MessageKey>38301791352</MessageKey>
    <CorrelationKey>40401791352</CorrelationKey>
    <MessageID>ben1243</MessageID>
    <ConversationID/>
    <State>4</State>
    <Target>1</Target>
    <Format>2</Format>
    <ServiceName>IntegrationManagementService</ServiceName>
    <MethodName>submitDocument</MethodName>
    <ServiceID>http://www.taleo.com/ws/integration/toolkit/2005/07/
    action/export</ServiceID>
    <ApplicationCode>integrationtoolkit</ApplicationCode>
    <CreationDate>2011-10-28T14:38:24.000-04:00</CreationDate>
    <LastModificationDate>2011-10-28T14:38:24.000-04:00
    </LastModificationDate>
    <OwnerUserName>sysint</OwnerUserName>
    <Document>
     <RecordCount>1</RecordCount>
     <RecordIndex>1</RecordIndex>
     <Attributes>
      <Attribute>
       <Name>version</Name>
       <Value>http://www.taleo.com/ws/itk/prototype/2006/05</Value>
      </Attribute>
      <Attribute>
       <Name>mode</Name>
       <Value>T-XML</Value>
      </Attribute>
     </Attributes>
    </Document>
    <Attributes>
     <Attribute>
      <Namespace>http://www.taleo.com/ws/integration/toolkit/2005/07
      </Namespace>
      <Name>MiddlewarePort</Name>
      <Value/>
      <AttributeType>1</AttributeType>
     </Attribute>
     <Attribute>
      <Namespace>http://www.taleo.com/ws/integration/toolkit/2005/07
      </Namespace>
      <Name>MiddlewareProtocol</Name>
      <Value/>
      <AttributeType>1</AttributeType>
     </Attribute>
     <Attribute>
      <Namespace>http://www.w3.org/2005/03/addressing</Namespace>
      <Name>xPARAMx</Name>
      <Value>xPARAMxVALUEx</Value>
      <AttributeType>5</AttributeType>
     </Attribute>
     <Attribute>
      <Namespace>http://www.taleo.com/ws/integration/toolkit/2005/07
      </Namespace>
      <Name>MiddlewarePath</Name>
      <Value/>
      <AttributeType>1</AttributeType>
```

```
     </Attribute>
     <Attribute>
      <Namespace>http://www.taleo.com/ws/integration/toolkit/2005/07
      </Namespace>
      <Name>MiddlewareHost</Name>
      <Value/>
      <AttributeType>1</AttributeType>
     </Attribute>
     <Attribute>
      <Namespace>http://www.w3.org/2005/03/addressing</Namespace>
      <Name>ReplyTo</Name>
      <Value>http://www.taleo.com/ws/integration/toolkit/2005/07/
      addressing/queue</Value>
      <AttributeType>4</AttributeType>
     </Attribute>
    </Attributes>
    <HistoryItems>
     <HistoryItem>
      <Sequence>1</Sequence>
      <InitialState>1</InitialState>
      <FinalState>1</FinalState>
      <UserName>sysint</UserName>
      <CreationDate>2011-10-28T14:38:24.000-04:00</CreationDate>
     </HistoryItem>
     <HistoryItem>
      <Sequence>2</Sequence>
      <InitialState>1</InitialState>
      <FinalState>2</FinalState>
      <UserName>sysint</UserName>
      <CreationDate>2011-10-28T14:38:24.000-04:00</CreationDate>
     </HistoryItem>
     <HistoryItem>
      <Sequence>3</Sequence>
      <InitialState>2</InitialState>
      <FinalState>3</FinalState>
      <UserName>sysint</UserName>
      <CreationDate>2011-10-28T14:38:24.000-04:00</CreationDate>
     </HistoryItem>
     <HistoryItem>
      <Sequence>4</Sequence>
      <InitialState>3</InitialState>
      <FinalState>4</FinalState>
      <UserName>sysint</UserName>
      <CreationDate>2011-10-28T14:38:24.000-04:00</CreationDate>
     </HistoryItem>
    </HistoryItems>
   </IntegrationMessage>
  </ns1:getMessageByKeyResponse>
 </soap:Body>
</soap:Envelope>
```

getMessageByIdentifier (Poll)

Description

Returns a complete report on the status of a message providing, among others, the key and identifier of the message, its current state, its state history and the document information.

Syntax

ns2011_05:IntegrationMessage : getMessageByIdentifier(tns:identifier)

Direction	SOAP Part	Name
In	Body	tns:identifier
Out	Body	ns2011_05:IntegrationMessage

Table 8: Request Parameters

Name	Description
tns:identifier	Identifier of the message as set into the messageID element of the request file. Type: xsd:string Default:

Table 9: Response Elements

Name	Description
ns2011_05:IntegrationMessage	An integration message is the main entity handled by the Taleo Connect technology. Managed messages represent a wrapper around an integration document containing the actual integration instructions that specifies the connectivity and communication information. Type: Complex Type
ns2011_05:IntegrationMessage.MessageKey	The message key is a globally unique identifier generated by the Taleo Connect Server that is only to be used internally within the Taleo product instance. Type: xsd:integer
ns2011_05:IntegrationMessage.CorrelationKey	The correlation key is the message key of the related message in integration process (request or response depending on the nature of the message), this is only to be used internally within the Taleo product instance. Type: xsd:integer
ns2011_05:IntegrationMessage.MessageID	The message identifier is an external driven identifier for the message that is provided by the caller for request messages and generated by the Taleo Connect server for response type messages. When generated by the Taleo Connect server, the identifier value is guaranteed to be globally unique. The same restriction should apply when a caller supplies such a value. Type: xsd:string
ns2011_05:IntegrationMessage.ConversationID	The conversation identifier is an additional identifier offered to link message together in a non-restrictive manner. There are no business rules attached to this value in the Taleo Connect server other than to set this value with the value of the request message conversation identifier when creating a response message (if such a value is present).

Name	Description
	Type: xsd:string
ns2011_05:IntegrationMessage.State	The message state indicates its processing state within the Taleo Connect server. Type: xsd:integer
ns2011_05:IntegrationMessage.Target	The message target indicates how the Taleo Connect server processes the message. The specific values are described in the main Taleo Connect documentation. Type: xsd:integer
ns2011_05:IntegrationMessage.Format	The message format indicates the envelope format of the message. The specific values are described in the main Taleo Connect documentation. Type: xsd:integer
ns2011_05:IntegrationMessage.ServiceName	Type: xsd:string
ns2011_05:IntegrationMessage.MethodName	Type: xsd:string
ns2011_05:IntegrationMessage.ServiceID	Type: xsd:string
ns2011_05:IntegrationMessage.ApplicationCode	For managed documents, this value indicates the business service invoked when processing the integration document. It is the value provided in the WS-Addressing action property. Type: xsd:string
ns2011_05:IntegrationMessage.Managed	The managed flag indicates whether a message contains an integration document. Type: xsd:boolean
ns2011_05:IntegrationMessage.CreationDate	Type: xsd:dateTime
ns2011_05:IntegrationMessage.LastModificationDate	Type: xsd:dateTime
ns2011_05:IntegrationMessage.Document	The document is a high-level representation of the integration document, except for the actual content. Type: complex type (see structure below) Type: complex type (see structure below)
ns2011_05:IntegrationMessage.Document.RecordCount	The total number of integration instructions present in the message (instructions can be viewed as results when the message is a response). Type: xsd:integer

Name	Description
ns2011_05:IntegrationMessage.Document.RecordIndex	The total number of instructions currently processed through the Taleo Connect server. The range of this value is 0 to n, where n is the RecordCount value. Type: xsd:integer
ns2011_05:IntegrationMessage.Document.Attributes.Name	Type: xsd:string
ns2011_05:IntegrationMessage.Document.Attributes.Value	Type: xsd:string
ns2011_05:IntegrationMessage.Attributes.Attribute.Namespace	Type: xsd:string
ns2011_05:IntegrationMessage.Attributes.Attribute.Name	Type: xsd:string
ns2011_05:IntegrationMessage.Attributes.Attribute.Value	Type: xsd:string
ns2011_05:IntegrationMessage.HistoryItems.HistoryItem.Sequence	The sequence of the history item. Type: xsd:integer
ns2011_05:IntegrationMessage.HistoryItems.HistoryItem.InitialState	Refers to the message state. Type: xsd:integer
ns2011_05:IntegrationMessage.HistoryItems.HistoryItem.FinalState	Refers to the message state. Type: xsd:integer
ns2011_05:IntegrationMessage.HistoryItems.HistoryItem.UserName	Type: xsd:string
ns2011_05:IntegrationMessage.HistoryItems.HistoryItem.CreationDate	Type: xsd:dateTime

Example:

Request

```
<soapenv:Envelope
               xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:man="http://www.taleo.com/ws/integration/
toolkit/2011/05/management">
 <soapenv:Header/>
 <soapenv:Body>
  <man:getMessageByIdentifier>
   <man:identifier>MSG0f4457ea-a865-4948-a675-d60fd1e48494
   </man:identifier>
  </man:getMessageByIdentifier>
 </soapenv:Body>
```

```
</soapenv:Envelope>
```

Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
 <soap:Body>
  <ns1:getMessageByIdentifierResponse xmlns:ns1="http://www.taleo.com/ws/
  integration/toolkit/2011/05/management">
   <IntegrationMessage xmlns="http://www.taleo.com/ws/integration/
   toolkit/2011/05">
    <MessageKey>1031501710688</MessageKey>
    <CorrelationKey>1031601710688</CorrelationKey>
    <MessageID>MSG0f4457ea-a865-4948-a675-d60fd1e48494</MessageID>
    <ConversationID/>
    <State>5</State>
    <Target>1</Target>
    <Format>2</Format>
    <ServiceName>IntegrationManagementService</ServiceName>
    <MethodName>submitDocument</MethodName>
    <ServiceID>http://www.taleo.com/ws/integration/toolkit/2005/07/
    action/export</ServiceID>
    <ApplicationCode>integrationtoolkit</ApplicationCode>
    <CreationDate>2011-06-03T08:32:20.000-04:00</CreationDate>
    <LastModificationDate>2011-06-03T08:32:21.000-04:00
    </LastModificationDate>
    <OwnerUserName>sysint</OwnerUserName>
    <Document>
     <RecordCount>1</RecordCount>
     <RecordIndex>1</RecordIndex>
     <Attributes>
      <Attribute>
       <Name>version</Name>
       <Value>http://www.taleo.com/ws/itk/prototype/2006/05</Value>
      </Attribute>
      <Attribute>
       <Name>mode</Name>
       <Value>T-XML</Value>
      </Attribute>
     </Attributes>
    </Document>
    <Attributes>
     <Attribute>
      <Namespace>http://www.taleo.com/ws/integration/toolkit/2005/07
      </Namespace>
      <Name>MiddlewarePort</Name>
      <Value/>
      <AttributeType>1</AttributeType>
     </Attribute>
     <Attribute>
      <Namespace>http://www.taleo.com/ws/integration/toolkit/2005/07
      </Namespace>
      <Name>MiddlewareProtocol</Name>
      <Value/>
      <AttributeType>1</AttributeType>
     </Attribute>
     <Attribute>
      <Namespace>http://www.w3.org/2005/03/addressing</Namespace>
      <Name>xPARAMx</Name>
```

```
      <Value>xPARAMxVALUEx</Value>
      <AttributeType>5</AttributeType>
     </Attribute>
     <Attribute>
      <Namespace>http://www.taleo.com/ws/integration/toolkit/2005/07
      </Namespace>
      <Name>MiddlewarePath</Name>
      <Value/>
      <AttributeType>1</AttributeType>
     </Attribute>
     <Attribute>
      <Namespace>http://www.taleo.com/ws/integration/toolkit/2005/07
      </Namespace>
      <Name>MiddlewareHost</Name>
      <Value/>
      <AttributeType>1</AttributeType>
     </Attribute>
     <Attribute>
      <Namespace>http://www.w3.org/2005/03/addressing</Namespace>
      <Name>ReplyTo</Name>
      <Value>http://www.taleo.com/ws/integration/toolkit/2005/07/
      addressing/queue</Value>
      <AttributeType>4</AttributeType>
     </Attribute>
    </Attributes>
    <HistoryItems>
     <HistoryItem>
      <Sequence>1</Sequence>
      <InitialState>1</InitialState>
      <FinalState>1</FinalState>
      <UserName>sysint</UserName>
      <CreationDate>2011-06-03T08:32:20.000-04:00</CreationDate>
     </HistoryItem>
     <HistoryItem>
      <Sequence>2</Sequence>
      <InitialState>1</InitialState>
      <FinalState>2</FinalState>
      <UserName>sysint</UserName>
      <CreationDate>2011-06-03T08:32:20.000-04:00</CreationDate>
     </HistoryItem>
     <HistoryItem>
      <Sequence>3</Sequence>
      <InitialState>2</InitialState>
      <FinalState>3</FinalState>
      <UserName>sysint</UserName>
      <CreationDate>2011-06-03T08:32:20.000-04:00</CreationDate>
     </HistoryItem>
     <HistoryItem>
      <Sequence>4</Sequence>
      <InitialState>3</InitialState>
      <FinalState>4</FinalState>
      <UserName>sysint</UserName>
      <CreationDate>2011-06-03T08:32:20.000-04:00</CreationDate>
     </HistoryItem>
     <HistoryItem>
      <Sequence>5</Sequence>
      <InitialState>4</InitialState>
      <FinalState>5</FinalState>
      <UserName>sysint</UserName>
      <CreationDate>2011-06-03T08:32:21.000-04:00</CreationDate>
     </HistoryItem>
```

```
    </HistoryItems>
   </IntegrationMessage>
  </ns1:getMessageByIdentifierResponse>
 </soap:Body>
</soap:Envelope>
```

getLargeDocumentByKey (Retrieve)

Description

Returns the actual document content of a message as attachment. When invoked for a response type message, the result of the integration process is retrieved. If invoked for a request type message, an echo of the originally submitted request is retrieved.

The returning of large document is an attachment containing a ns2011_05:Document element. The document is returned using the MTOM (Message Transmission Optimization Mechanism) technology for transporting data in an attachment manner.

Syntax

tns:Document : getLargeDocumentByKey (tns:messageKey)

Direction	SOAP Part	Name
In	Body	tns:messageKey
Out	Body	tns:Document

Table 10: Request Parameters

Name	Description
tns:messageKey	Identifier of the response document, corresponds to the tns:IntegrationMessage.CorrelationKey value returned by the getMessageByKey or getMessageByIdentifier operations. Type: xsd:string

Table 11: Response Elements

Name	Description
tns:Document	This element contains a base64 encoded ns2011_05:Document. Type: nsm:StreamBody

Table 12: ns2011_05:Document Structure

Name	Description
ns2011_05:Document	A document is a generic container that holds the information elements of the message. Type: Complex Type
ns2011_05:Document.Attributes	The attributes represent document level information elements. Type: Complex Type

Name	Description
ns2011_05:Document.Attributes.Attribute	An attribute is a document level information element. Type: Complex Type
ns2011_05:Document.Attributes.Attribute.Name	The attribute name uniquely identifies it among the attributes collection of a document. Type: xsd:string
ns2011_05:Document.Content	The document content represents the record level information elements. This type of information in the document content is dependent on the processing context. As specified by the schema, the Content element always contains exactly one child element which is the root of the content. Each immediate child of this element is considered a record in the Taleo Connect terminology. This size of the information in this element may be large; the Taleo Connect server is tuned to handle this eventuality. Type: xsd:any

Example:

Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
                    soap/envelope/"
                    xmlns:man="http://www.taleo.com/ws/integration/
toolkit/2011/05/management">
 <soapenv:Header/>
 <soapenv:Body>
  <man:getLargeDocumentByKey>
   <man:messageKey>1031601710688</man:messageKey>
  </man:getLargeDocumentByKey>
  <man1:getLargeDocumentByKey xmlns:man1="http://www.taleo.com/ws/
  integration/toolkit/2011/05/management"/>
 </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
  <ns1:getLargeDocumentByKeyResponse xmlns:ns1="http://www.taleo.com/ws/
  integration/toolkit/2011/05/management">
   <ns1:out ns2:contentType="text/xml; charset=UTF-8" xmlns:ns2=
   "http://www.w3.org/2004/11/xmlmime">
    <Include href="cid:13071043411793820106945 47@http://www.w3.org/
    2005/05/xmlmime" xmlns="http://www.w3.org/2005/05/xop/include"/>
   </ns1:out>
  </ns1:getLargeDocumentByKeyResponse>
 </soap:Body>
</soap:Envelope>
```

Attachment Response

```
<Document xmlns="http://www.taleo.com/ws/integration/toolkit/2011/05">
 <Attributes>
```

```
  <Attribute name="duration">0:00:00.294</Attribute>
  <Attribute name="count">4</Attribute>
  <Attribute name="entity">Character</Attribute>
  <Attribute name="mode">T-XML</Attribute>
  <Attribute name="version">http://www.taleo.com/ws/itk/
  prototype/2006/05</Attribute>
 </Attributes>
 <Content>
  <ExportTXML xmlns="http://www.taleo.com/ws/integration/toolkit/2005/07/
  action/export" xmlns:e="http://www.taleo.com/ws/itk/prototype/2006/05">
   <e:Character>
    <e:Name>Percival</e:Name>
    <e:Number>9001001</e:Number>
   </e:Character>
   <e:Character>
    <e:Name>Egor</e:Name>
    <e:Number>9001002</e:Number>
   </e:Character>
   <e:Character>
    <e:Name>C. Borg</e:Name>
    <e:Number>9001003</e:Number>
   </e:Character>
   <e:Character>
    <e:Name>Zenon</e:Name>
    <e:Number>9001004</e:Number>
   </e:Character>
  </ExportTXML>
 </Content>
</Document>
```

getDocumentByKey (Retrieve)

Description

Returns the actual document content of a message. When invoked for a response type message, the result of the integration process is retrieved. If invoked for a request type message, an echo of the originally submitted request is retrieved.

Should only be used to get a small document. For a big document use the getLargeDocumentByKey operation.

Syntax

tns:Document : getDocumentByKey (tns:messageKey)

Direction	SOAP Part	Name
In	Body	tns:messageKey
Out	Body	tns:Document

Table 13: Request Parameters

Name	Description
tns:messageKey	Identifier of the response. document Correspond to the tns:IntegrationMessage.CorrelationKey value returned by the getMessageByKey or getMessageByIdentifier operations.

Name	Description
	Type: xsd:string

Table 14: Response Elements

Name	Description
tns:Document	This element contains a base64 encoded ns2011_05:Document Type: nsm:StreamBody

Table 15: ns2011_05:Document Structure

Name	Description
ns2011_05:Document	A document is a generic container that holds the information elements of the message. Type: Complex Type
ns2011_05:Document.Attributes	The attributes represent document level information elements. Type: Complex Type
ns2011_05:Document.Attributes.Attribute	An attribute is a document level information element. Type: Complex Type
ns2011_05:Document.Attributes.Attribute.Name	The attribute name uniquely identifies it among the attributes collection of a document. Type: xsd:string
ns2011_05:Document.Content	The document content represents the record level information elements. This type of information in the document content is dependent on the processing context. As specified by the schema, the Content element always contains exactly one child element which is the root of the content. Each immediate child of this element is considered a record in the Taleo Connect terminology. This size of the information in this element may be large; the Taleo Connect server is tuned to handle this eventuality. Type: xsd:any

Example:

Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
soap/envelope/" xmlns:man="http://www.taleo.com/ws/integration/
toolkit/2011/05/management">
 <soapenv:Header/>
 <soapenv:Body>
  <man:getDocumentByKey>
   <man:messageKey>1031601710688</man:messageKey>
  </man:getLargeDocumentByKey>
  <man1:getDocumentByKey xmlns:man1="http://www.taleo.com/ws/integration/
```

```
  toolkit/2011/05/management"/>
 </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
  <ns1:getDocumentByKeyResponse xmlns:ns1="http://www.taleo.com/ws/
integration/toolkit/2011/05/management">
   <Document xmlns="http://www.taleo.com/ws/
   integration/toolkit/2011/05">
    <Attributes>
     <Attribute name="duration">0:00:00.294</Attribute>
     <Attribute name="count">4</Attribute>
     <Attribute name="entity">Character</Attribute>
     <Attribute name="mode">T-XML</Attribute>
     <Attribute name="version">http://www.taleo.com/ws/itk/
     prototype/2006/05</Attribute>
    </Attributes>
    <Content>
     <ExportTXML xmlns="http://www.taleo.com/ws/integration/
     toolkit/2005/07/action/export"
     xmlns:e="http://www.taleo.com/ws/itk/prototype/2006/05">
      <e:Character>
       <e:Name>Percival</e:Name>
       <e:Number>9001001</e:Number>
      </e:Character>
      <e:Character>
       <e:Name>Egor</e:Name>
       <e:Number>9001002</e:Number>
      </e:Character>
      <e:Character>
       <e:Name>C. Borg</e:Name>
       <e:Number>9001003</e:Number>
      </e:Character>
      <e:Character>
       <e:Name>Zenon</e:Name>
       <e:Number>9001004</e:Number>
      </e:Character>
     </ExportTXML>
    </Content>
   </Document>
  </ns1:getDocumentByKeyResponse>
 </soap:Body>
</soap:Envelope>
```

API Usage Guidelines

Defining Polling Interval

When using the operation getMessageByKey or getMessageByIdentifier (polling), some guidelines must be followed in order to avoid a heavy load of communications with the zone.

Depending on the size of the document, the polling interval must be adjusted so no unnecessary polling is done on the zone. Each case should be analyzed and the interval should be proportional to the estimated time required for the completion of the operation.

The completion of an operation can be considered done when either one of these states has been reached (see Message Processing for more information about each state):

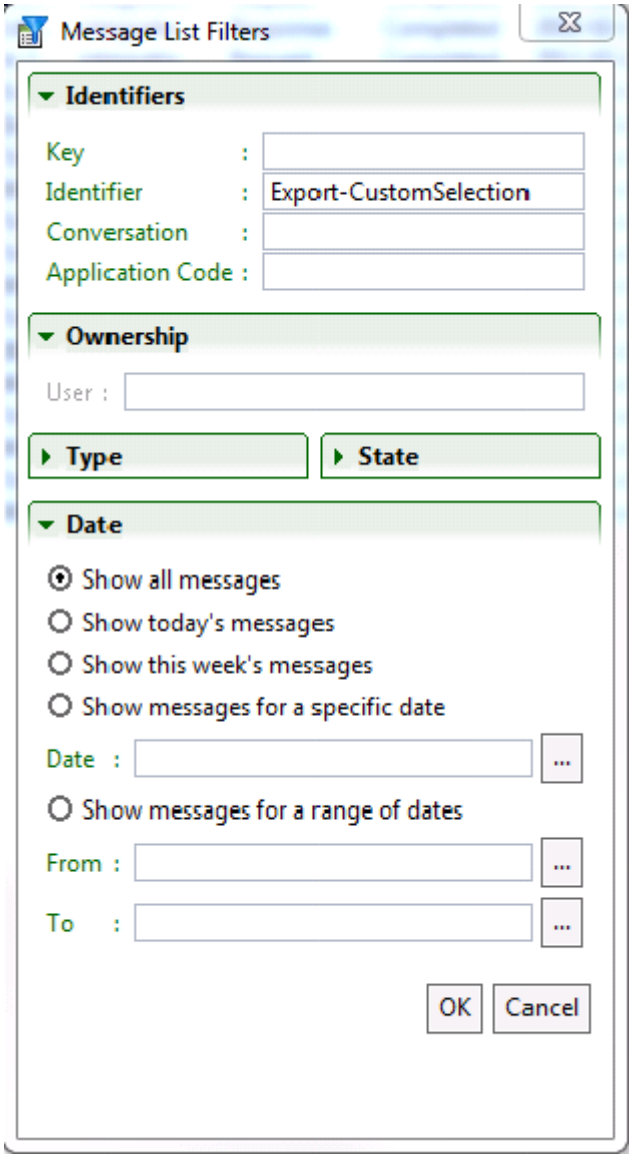
- COMPLETED
- INTERRUPTED
- INERROR
- SUSPENDED

MessageID and MessageKey Use

MessageID

When creating the request, it is also recommended to use a significant MessageID (wsa:MessageID in the SOAP header). This will help to easily retrieve the information in the Taleo Connect Client Console.

In the profile wizard it is possible to filter the documents for a specific MessageID (called “Identifier” in TCC) as shown below:



MessageKey

The MessageKey (“Key” in TCC)(ns2011_05:IntegrationMessage.MessageKey) associated with the request file will become the CorrelationKey ((ns2011_05:IntegrationMessage.CorrelationKey) in the response message and vice versa since this is how the files are related.

When doing the polling, the MessageKey should be used at first to retrieve the information. A validation will then need to be performed to verify if the CorrelationKey has a value. From the time a value is available in the CorrelationKey, it can be use to continue the polling. Using the CorrelationKey will allow to have more information in the response, i.e. the progression of the actions. Following is an extract of a response using the CorrelationKey that will allow monitoring of the progress:

```
<IntegrationMessage xmlns="http://www.taleo.com/ws/integration/
toolkit/2011/05">
…
<Document>
        <RecordCount>46900</RecordCount>
```

```
        <RecordIndex>6000</RecordIndex>
        <Attributes/>
</Document>
…
</IntegrationMessage>
```

Import Request Index and Identifier Use

When doing an import, the response file contains the result for each imported entities and each of them have an index. This information can be used to link the entities from the request file to the result in the response file. If more relevant information is needed to map the entities to the corresponding data in the request file, it is possible to use an identifier.

Although the index will always be available in the response file to relate to the request file, the identifier needs to be added for each entity as follow in the request file:

```
<ns:Content>
       <ImportDocument xmlns="http://www.taleo.com/ws/itk/
prototype/2006/05">
            <Character identifier="SignificativeValue">
               <Name>CharacterTest</Name>
                <Level>1</Level>
                <Class>
                    <CharacterClass>
                        <Description>
                            <value locale="en" searchType="search"
 searchValue="Fighter"/>
                        </Description>
                    </CharacterClass>
                </Class>
            </Character>
       </ImportDocument>
</ns:Content>
```

When adding an identifier to an entity, the response file will then contain an index as well as an identifier for each entity as follow:

```
<Content>
   <ImportResult xmlns="http://www.taleo.com/ws/integration/
toolkit/2005/07/action/import">
      <Record>
         <Index>1</Index>
         <Identifier>SignificativeValue</Identifier>
         <Status>success</Status>
         <TransactionType>character.create</TransactionType>
     </Record>
   </ImportResult>
</Content>
```

Document Values Construction

The bulk API is just an API to transport integration messages from the client to the zone and vice versa.

The following guidelines should help build the Document values of the request to send to the zone.

Here is the structure of a submitDocument operation. For clarity purpose, some namespaces have been removed (this is not a valid XML document).

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<SOAP-ENV:Envelope>
 <SOAP-ENV:Header>
  <wsa:MessageID/>
  <wsa:ReplyTo>
   <wsa:Address/>
   <wsa:ReferenceParameters/>
  </wsa:ReplyTo>
  <wsa:Action/>
 </SOAP-ENV:Header>
 <SOAP-ENV:Body>
  <m:submitDocument>
   <Document>
    <Attributes>
     <Attribute name="an attribute name"/>
    </Attributes>
    <Content>
     <Root>
      <Entity-operation xmlns="http://www.taleo.com/ws/product/version">
       <Param_1>
        …
       </Param_1>
      </Entity-operation>
      <Entity-operation xmlns="http://www.taleo.com/ws/product/version">
       <Param_1>
        …
       </Param_1>
      </Entity-operation>
     </Root>
    </Content>
   </Document>
  </m:submitDocument>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The bulk API WSDL and Schema are describing the XML structure up to the Document/Attributes and Document/Content element. What is in these elements is irrelevant for the Bulk API. But it is really important for the Bulk API user.

Document Attributes

The first section of the document contains the definition of the attributes related to the processing of the document content.

```
<SOAP-ENV:Body>
 <m:submitDocument>
  <Document>
   <Attributes>
    <Attribute name="name"/>
   </Attributes>
```

Here is the list of the possible attributes:

- **csvdelimiter:** String used to separate individual values.

Name	Value
Service Type	Export
Possible Values	N/A

Name	Value
Default Values	The comma character

- **csvheader:** Flag indicating if a header line should be shown.

Name	Value
Service Type	Export
Possible Values	true, false
Default Values	false

- **csvquote:** String to use to quote individual values

Name	Value
Service Type	Export
Possible Values	N/A
Default Values	The double quote character

- **largegraph:** Enables the flag indicating if objects must be loaded type by type instead of one big chunk. For huge loading graphs, it is preferable the load them type by type.

Name	Value
Service Type	Export
Possible Values	true, false
Default Values	false (new Query in TCC is default to true)

- **locale:** Default locale to use.

Name	Value
Service Type	Import/Export
Possible Values	Language abbreviation
Default Values	
Example	en

- **mode:** The export mode.

Name	Value
Service Type	Export
Possible Values	T-XML, CSV-ENTITY, CSV, XML
Default Values	

- **non.updatable.fields:** Directive to the integration operation on what to do when trying to update an non-updatable field.

Name	Value
Service Type	Import
Possible Values	error, warning, error.different
Default Values	error

Value	Equal XML and Entity value	Different XML and Entity value
error	When trying to update a non-updatable field, returns an error message Fails the import.	When trying to update a non-updatable field, returns an error message. Fails the import.
warning	When trying to update a non-updatable field, returns a warning message saying that both values have the same value. Does not fail the import but the entity value is not updated with the provided one.	When trying to update a non-updatable field, returns a warning message saying that both values do not have the same value. Does not fail the import but the entity value is not updated with the provided one.
error.different	When trying to update a non-updatable field, returns a warning message saying that both values have the same value. Does not fail the import but the entity value is not updated with the provided one.	When trying to update a non-updatable field, returns a warning message saying that both values do not have the same value. Fails the import.

- **pagingsize:** Number of records by page.

Name	Value
Service Type	Export
Possible Values	-1 (no paging), or a positive number
Default Values	-1

- **preparator:** Used to specified a service that will be called before the first record and after the last record.

Name	Value
Service Type	Import
Possible Values	See data dictionary
Default Values	
Example	olf

- **switch.system.maintenance:** Switch the zone to maintenance. In some product, it is required to put the zone is maintenance to execute an operation. Service that required that are documented in the product data dictionary (e.g.: the JobField entity of the Smartorg product in version 11a).

Name	Value
Service Type	Import
Possible Values	always
Default Values	

- **unknown.custom.fields:** Behavior when a UDF name is specified in an import request and not find in the zone.

Name	Value
Service Type	Import
Possible Values	warn, error, ignore
Default Values	error

- **version:** Data mapping version.

Name	Value
Service Type	Import, Export
Possible Values	See data dictionary in the Data Model Version section.
Default Values	
Example	http://www.taleo.com/ws/tee800/2009/01

Document Content

The second section of the document contains the document content that will be processed by a specified operation.

```
<SOAP-ENV:Body>
 <m:submitDocument>
  <Document>
   <Attributes>
    <Attribute name="version"/>
   </Attributes>
   <Content>
    <Root>
     <Entity-operation xmlns="http://www.taleo.com/ws/product/version">
      <Param_1>
       …
      </Param_1>
      <Param_2>
       …
      </Param_2>
     </Entity-operation>
     <Entity-operation xmlns="http://www.taleo.com/ws/product/version">
      <Param_1>
       …
      </Param_1>
      <Param_2>
       …
      </Param_2>
     </Entity-operation>
    </Root>
   </Content>
```

The first element under the <Content> element (Root) is one of the following:

Element	Description
<ImportEntities>	Root element containing a list of operations to perform on an entity.
<ExportQuery>	Root element containing a selection query.

ImportEntities

The element name under the ImportEntities one has the following form: *Entity-operation*

Entity

The *Entity* value corresponds to the entity type name on which the import will be done. The list of possible entities is documented in the product data dictionary document under the section “Service / Taleo Connect Client Services / Service Summary”

Here are some examples from the “Taleo Enterprise, Recruiting 11A Data Dictionary” document:

- Candidate
- Department
- Offer
- JobTemplate

An entity always starts with a capital letter and each word starts will a capital letter as well.

operation

The *operation* value is a bit tricky to construct. The following rules must be applied:

1. The possible operations are listed under the “Service / Taleo Connect Client Services / Service Details” section of the product data dictionary document.
2. The operation name from de data dictionary (lets call it Documented Operation) must be transformed into a T-XML operation name (lets call it XML Operation).
   a. The Documented Operation is composed of one or many word(s). The first word starts with a lowercase letter and each other words starts with an uppercase letter.
   b. In the XML Operation, each word must start with a lowercase and must be separated by an hyphen character (-).

Here are some examples:

Documented Operation	XML Operation
addComments	add-comments
attachFile	attach-file
attachRecruiterFileToApplication	attach-recruiter-file-to-application
bypass	bypass
create	create
match	match
merge	merge
mergeWithEmailControl	merge-with-email-control

Entity-operation

To summarize, here is an example of *Entity-operation* with the Candidate entity:

- Candidate-add-comments
- Candidate-attach-file
- Candidate-attach-recruiter-file-to-application
- Candidate-bypass
- Candidate-create

- Candidate-match
- Candidate-merge
- Candidate-merge-with-email-control

Operation Parameters

Under the *Entity-operation* element is the list of operation parameter(s).

Each parameter will be described in a XML element. To know the list of parameter(s) a service operation takes, refer to the “Service / Taleo Connect Client Services / Service Details” section of the product data dictionary document.

The TCS doesn’t make strong validation on the parameter name, what is important is the number of parameters and their order. So the best practice rule that you should follow is to use the same name as in the product data dictionary.

Here is an example with the Requisition Fill operation:

Operation
`fill (Requisition requisition, String comment)`
Fill a requisition If requisition is POSTED OR SCHEDULED, all postings are ended If none active offers exist, all are also ended Requisition is set to FILLED Requisition must be OPEN, POSTED OR SCHEDULED or UNPOSTED OR EXPIRED User must be allowed to view and edit requisition Action is not available if any active offer exists

XML:

```
<SOAP-ENV:Body>
 <submitDocument xmlns="http://www.taleo.com/ws/integration/
toolkit/2005/07">
  <Document>
   <Attributes>
    <Attribute name="locale">en</Attribute>
    <Attribute name="pagingsize">1</Attribute>
    <Attribute name="processing.batching.size">1</Attribute>
    <Attribute name="version">http://www.taleo.com/ws/tee800/2009/01</
Attribute>
    <Attribute name="default.operation">merge</Attribute>
   </Attributes>
   <Content>
    <ImportEntities>
     <Requisition-fill>
      <requisition>
       <ContestNumber searchType="search" searchValue="one" />
      </requisition>
      <comment>Fill the requisition</comment>
     </Requisition-fill>
    </ImportEntities>
   </Content>
  </Document>
 </submitDocument>
</SOAP-ENV:Body>
```

For more information about the construction of import request, see the Import Feature section

ExportQuery

The element under the <ExportQuery> element is one <query> element. The content of this element contains a Taleo Selection Query statement represented in a XML form.

For more information about how to build a query, see the Export Feature section.

Import Results Parsing

The import results respects a fixed data structure that is defined in the http://www.taleo.com/ws/integration/toolkit/2005/07/action/import schema present in the WSDL.

To see an example on how to process it, see the IntegrationManagementServiceMTOMSample sample code. The class IntegrationManagementServiceSample method retrieveLargeDocument invokes the ImportResultHandler class that demonstrate how to process an import result.

Again, the Bulk API can be used to process big incoming and outgoing documents. So be careful to NOT load in memory the entire content of the import result.

Here is an example of an Import Result:

In this case, the good way to process it is by loading in memory one Record element at a time.

The import result soap document with a reference to the attachment

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
   <soap:Body>
      <ns1:getLargeDocumentByKeyResponse xmlns:ns1="http://www.taleo.com/
ws/integration/toolkit/2011/05/management">
         <ns1:Document ns2:contentType="text/xml; charset=UTF-8"
 xmlns:ns2="http://www.w3.org/20045/1105/xmlmime">
            <Include href="cid:133286973731963702821870@http://
www.w3.org/2001/XMLSchema" xmlns="http://www.w3.org/2004/08/xop/include"/>
         </ns1:Document>
      </ns1:getLargeDocumentByKeyResponse>
   </soap:Body>
</soap:Envelope>
```

The attachment (with one record that failed to import)

```
<Document xmlns="http://www.taleo.com/ws/integration/toolkit/2011/05">
 <Attributes>
  <Attribute name="progress.interval">1</Attribute>
  <Attribute name="total.character">10</Attribute>
  <Attribute name="default.operation">merge</Attribute>
  <Attribute name="total.character.create">10</Attribute>
  <Attribute name="duration">0:00:01.261</Attribute>
  <Attribute name="internal.audit.session.id">287512617</Attribute>
  <Attribute name="internal.cluster.node.info">

 (product:integrationtoolkit,id:node253454_253452,http:&amp;quot;192.168.128.40:19654&amp;quot;,
integrationtoolkit,isCoordinator:false)</Attribute>
  <Attribute name="processing.batching.size">1</Attribute>
  <Attribute name="internal.cluster.node.id">node253454_253452</Attribute>
  <Attribute name="total.status.success">10</Attribute>
 </Attributes>
 <Content>
  <ImportResult xmlns="http://www.taleo.com/ws/integration/toolkit/2005/07/
action/import">
   <Record>
    <Index>1</Index>
    <Status>success</Status>
    <TransactionType>character.create</TransactionType>
   </Record>
```

```
   <Record>
    <Index>2</Index>
    <Status>error</Status>
    <TransactionType>character.create</TransactionType>
    <Errors>
     <Error>One of the values you have entered is not unique.</Error>
    </Errors>
   </Record>
   <Record>
    <Index>3</Index>
    <Status>success</Status>
    <TransactionType>character.create</TransactionType>
   </Record>
   <Record>
    <Index>4</Index>
    <Status>success</Status>
    <TransactionType>character.create</TransactionType>
   </Record>
   <Record>
    <Index>5</Index>
    <Status>success</Status>
    <TransactionType>character.create</TransactionType>
   </Record>
  </ImportResult>
 </Content>
</Document>
```

Export Results Parsing

The format of the export result really depends on the way you build the projections of the export request.

To see an example on how to process it, see the IntegrationManagementServiceMTOMSample sample code. The class IntegrationManagementServiceSample method retrieveLargeDocument invoke the ExportResultHandler class that demonstrate how to process an export result.

Again, the Bulk API can be used to process big incoming and outgoing documents. So be careful to NOT load in memory the entire content of the export result.

Here is an example of an export query and its result. In this case, the good way to process it is by loading in memory one Character element at time.

The export request document:

```
<ns:Document xmlns:ns="http://www.taleo.com/ws/integration/
toolkit/2011/05">
    <ns:Attributes>
            <ns:Attribute name="mode">T-XML</ns:Attribute>
            <ns:Attribute name="version">http://www.taleo.com/ws/itk/
prototype/2006/05</ns:Attribute>
    </ns:Attributes>
    <ns:Content>
        <ExportQuery xmlns="http://www.taleo.com/ws/integration/
toolkit/2005/07/action/export">
            <query projectedClass="Character" alias="Character"
 xmlns="http://itk.taleo.com/ws/query">
                <projections>
                    <projection>
                        <field path="Number"/>
                    </projection>
                    <projection>
                        <field path="Character.Name"/>
                    </projection>
```

```
                </projections>
                <projectionSortings>
                    <projectionSorting>
                        <field path="Quests,Number"/>
                    </projectionSorting>
                    <projectionSorting ascending="true">
                        <field path="otherArmorPieces,Number"/>
                    </projectionSorting>
                </projectionSortings>
                <filterings>
                    <filtering>
                        <containsIgnoreCase>
                            <field path="Name"/>
                            <string>8c03ac06-3da7-48c5-acad-1bd7b679b31e</
string>
                        </containsIgnoreCase>
                    </filtering>
                </filterings>
            </query>
        </ExportQuery>
    </ns:Content>
</ns:Document>
```

The export result soap document with a reference to the attachment:

```
<Document xmlns="http://www.taleo.com/ws/integration/toolkit/2011/05">
 <Attributes>
  <Attribute name="duration">0:00:00.296</Attribute>
  <Attribute name="count">10</Attribute>
  <Attribute name="entity">Character</Attribute>
  <Attribute name="mode">T-XML</Attribute>
  <Attribute name="version">http://www.taleo.com/ws/itk/prototype/2006/05</
Attribute>
 </Attributes>
 <Content>
  <ExportTXML xmlns="http://www.taleo.com/ws/integration/toolkit/2005/07/
action/export" xmlns:e="http://www.taleo.com/ws/itk/prototype/2006/05">
   <e:Character>
    <e:Name>Import-CreateCharacter0-8c03ac06-3da7-48c5-acad-1bd7b679b31e</
e:Name>
    <e:Number>12401791944</e:Number>
   </e:Character>
   <e:Character>
    <e:Name>Import-CreateCharacter1-8c03ac06-3da7-48c5-acad-1bd7b679b31e</
e:Name>
    <e:Number>12501791944</e:Number>
   </e:Character>
   <e:Character>
    <e:Name>Import-CreateCharacter2-8c03ac06-3da7-48c5-acad-1bd7b679b31e</
e:Name>
    <e:Number>12601791944</e:Number>
   </e:Character>
   <e:Character>
    <e:Name>Import-CreateCharacter3-8c03ac06-3da7-48c5-acad-1bd7b679b31e</
e:Name>
    <e:Number>12701791944</e:Number>
   </e:Character>
   <e:Character>
    <e:Name>Import-CreateCharacter4-8c03ac06-3da7-48c5-acad-1bd7b679b31e</
e:Name>
    <e:Number>12801791944</e:Number>
```

```
   </e:Character>
   <e:Character>
    <e:Name>Import-CreateCharacter5-8c03ac06-3da7-48c5-acad-1bd7b679b31e</
e:Name>
    <e:Number>12901791944</e:Number>
   </e:Character>
  </ExportTXML>
 </Content>
</Document>
```

The attachment:

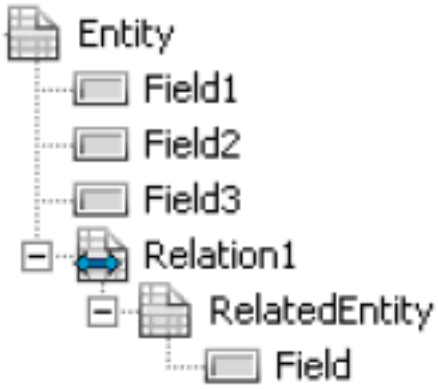
```
<Document xmlns="http://www.taleo.com/ws/integration/toolkit/2011/05">
 <Attributes>
  <Attribute name="duration">0:00:00.296</Attribute>
  <Attribute name="count">10</Attribute>
  <Attribute name="entity">Character</Attribute>
  <Attribute name="mode">T-XML</Attribute>
  <Attribute name="version">http://www.taleo.com/ws/itk/prototype/2006/05</
Attribute>
 </Attributes>
 <Content>
  <ExportTXML xmlns="http://www.taleo.com/ws/integration/toolkit/2005/07/
action/export" xmlns:e="http://www.taleo.com/ws/itk/prototype/2006/05">
   <e:Character>
    <e:Name>Import-CreateCharacter0-8c03ac06-3da7-48c5-acad-1bd7b679b31e</
e:Name>
    <e:Number>12401791944</e:Number>
   </e:Character>
   <e:Character>
    <e:Name>Import-CreateCharacter1-8c03ac06-3da7-48c5-acad-1bd7b679b31e</
e:Name>
    <e:Number>12501791944</e:Number>
   </e:Character>
   <e:Character>
    <e:Name>Import-CreateCharacter2-8c03ac06-3da7-48c5-acad-1bd7b679b31e</
e:Name>
    <e:Number>12601791944</e:Number>
   </e:Character>
   <e:Character>
    <e:Name>Import-CreateCharacter3-8c03ac06-3da7-48c5-acad-1bd7b679b31e</
e:Name>
    <e:Number>12701791944</e:Number>
   </e:Character>
   <e:Character>
    <e:Name>Import-CreateCharacter4-8c03ac06-3da7-48c5-acad-1bd7b679b31e</
e:Name>
    <e:Number>12801791944</e:Number>
   </e:Character>
   <e:Character>
    <e:Name>Import-CreateCharacter5-8c03ac06-3da7-48c5-acad-1bd7b679b31e</
e:Name>
    <e:Number>12901791944</e:Number>
   </e:Character>
  </ExportTXML>
 </Content>
</Document>
```

Mapping Feature

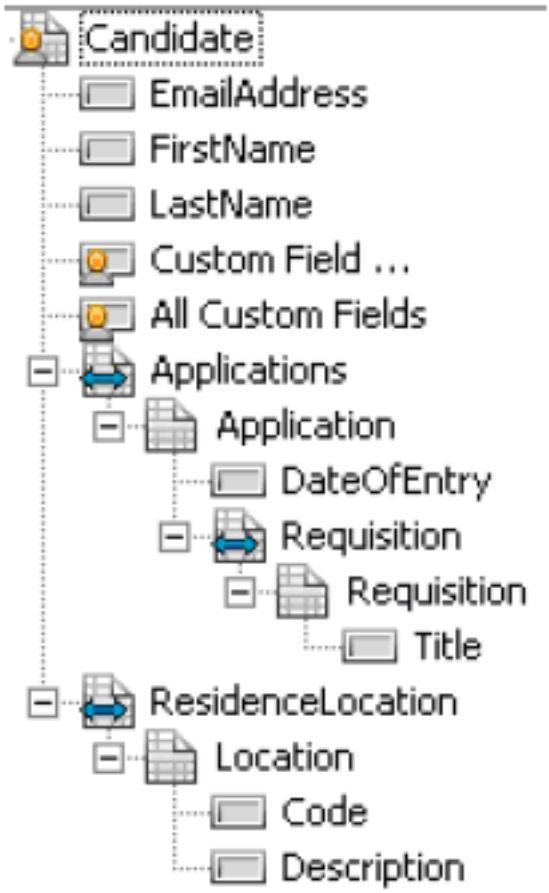
Product Business Model

All Taleo products are based on a highly structured business model that represents the data it handles. Business models are built with three concepts:

- Entity: A basic object unit that represents a general business concept.
- Field: A characteristic or attribute of an entity.
- Relation: A link between two entities.



We will be using a sample model in the following sections. Although inspired by the Taleo Recruiting business model, several simplifications have been made to keep the examples clear and manageable. The messages included in this document do not work against any actual Taleo product instances.



Again for simplicity, we will assume that a few business rules hold for our model:

- The EmailAddress field is unique throughout all Candidate entities.
- The Title is unique throughout all Requisition entities.
- The Code field is unique throughout all Location entities.
- The Requisition Title and Location Description fields are multi-lingual.
- A Candidate can have zero or more related Application entities, but at most one Application per Requisition.
- The Application DateOfEntry field may be null or empty.
- The Candidate has a single optional ResidenceLocation Location entity.

Versioning

Taleo products have evolving business models to reflect new features and improvements in the overall functionality. However, integration services need to be stable through time in order to ensure robust inter-system business processes. The Taleo Connect solution to this problem is a mapping feature that isolates the internal business model of Taleo products with a versioned XML representation.

Each release of a Taleo product contains a specific version of the integration mapping and thus a snapshot of the business model at that time. Any subsequent releases will also have a new mapping, but will fully support the previous mappings with no changes required on the client side.

When going through major product changes or refactoring, it is possible that certain requests must be modified because the underlying business model simply does not support the information. Taleo strives to keep these situations at a strict minimum.

Each version of a product's business model is identified by a URI that also serves as the namespace of the XML elements used to serialize the model entities. As with other namespaces, the URI does not represent an actual web URL, it is only used as a reference identifier. A few examples are:

- http://www.taleo.com/ws/integration/toolkit/2011/05/management
- http://www.taleo.com/ws/integration/toolkit/2011/05
- http://www.taleo.com/ws/itk/prototype/2006/05

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
  <ns1:submitDocumentResponse xmlns:ns1="http://www.taleo.com/ws/
integration/toolkit/2011/05/management">
   <IntegrationMessage xmlns="http://www.taleo.com/ws/integration/
toolkit/2011/05">
    <MessageKey>1232401710688</MessageKey>
   </IntegrationMessage>
  </ns1:submitDocumentResponse>
 </soap:Body>
</soap:Envelope>
```

In the following sections, we will use this URI as a sample model: http://www.taleo.com/ws/sample000/2006/06.

Documentation

The business models of the current Taleo products are documented within separate data dictionaries covering exhaustively the entities, fields and relations of the product along with their various characteristics (type, available to import and/to export, etc.).

Import Feature

The import feature processes integration instruction to create and/or update business entities within a Taleo product.

We will first see how to build an import instruction, then how to construct an import document and finally how to wrap the document in a T-SOAP message.

Import Instructions

Basic

Given the business model, writing a basic import instruction is quite straightforward. Assume we want to create a new candidate in our sample product, his name is **John Smith** and his e-mail is **jsmith@acme.com**, then the instruction would simple be:

```
<Candidate>
     <EmailAddress>jsmith@acme.com</EmailAddress>
     <FirstName>John</FirstName>
     <LastName>Smith</LastName>
</Candidate>
```

Create/Update

It is possible to write an instruction that will update an existing entity. In such a case, search attributes must be specified to determine the criteria to find the existing entity. The following instruction will look for the candidate based on the e-mail address and update the LastName field (leaving the FirstName field value as is):

```
<Candidate>
     <EmailAddress searchType=”search” searchValue=”jsmith@acme.com” >
     <LastName>Brown</LastName>
</Candidate>
```

The default behavior of an import instruction is to merge the information provided. That is, if the entity is found, then an update is made; if the entity is not found then it is created with the information provided.

In certain particular cases, the value used to determine the entity may also have to be updated, the syntax of the instruction would then be:

```
<Candidate>
     <EmailAddress searchType=”searchAndValue”
     searchValue=”jsmith@acme.com>jbrown@acme.com”
      </EmailAddress>
</Candidate>
```

Field types

There are four basic types used in Taleo business models: text, numeric, date and multilingual. Dates must be provided in the ISO-8601 format, for example:

```
<DateOfEnty>2006-06-01T14:15:00-04:00</DateOfEntry>
```

Multilingual values are provided individually by locale, for example:

```
<Location>
     <Code>USA</Code>
     <Description>
          <value locale=’en’>United States of America</value>
          <value locale=’fr’>États-Unis d’Amérique </value>
```

```
          <value locale='es'>Estados Unidos de América </value>
     </Description>
</Location>
```

Relations

When importing entities, most relations are lookups and the related entity is only linked to the main entity. This is the case of the *ResidenceLocation* relation of the *Candidate* entity. To determine the related entity, we re-use the same search attributes, but in a different context. For example, to specify that John Smith lives in the U.S.A, we would write:

```
<Candidate>
     <EmailAddress>jsmith@acme.com</EmailAddress>
     <FirstName>John</FirstName>
     <LastName>Smith</LastName>
     <ResidenceLocation>
          <Location>
               <Code searchType="search" searchValue="USA"/>
          </Location>
     </ResidenceLocation>
</Candidate>
```

Here we are not creating a new location with the code USA, simply finding this location in the product repository and linking it to the John Smith candidate we are creating.

A more complex example would update the John Smith candidate with another residence location by using the multilingual *Description* field.

```
<Candidate>
     <EmailAddress searchType="search" searchValue="jsmith@acme.com"/>
     <ResidenceLocation>
          <Location>
               <Description>
                    <value locale="en" searchType="search"
                    searchValue="Canada"/>
               </Description>
          </Location>
     </ResidenceLocation>
</Candidate>
```

The default behavior when importing relations is to append the new entities to the relation. However, in the case of single relations, appending a new entity replaces the existing one.

It is also possible that a related entity is heavily dependent on the main entity and is actually created/ updated during the instruction. This is the case of the *Applications* relation of the *Candidate* entity. As such, to get John Smith to apply on an Accountant job, we would write:

```
<Candidate>
     <EmailAddress searchType="search" searchValue=jsmith@acme.com/>
     <Applications>
          <Application>
               <Requisition>
                    <Title>
                         <value locale="en" searchType="search"
                         searchValue="Accountant"/>
                    </Title>
               </Requisition>
          </Application>
     </Applications>
</Candidate>
```

This creates an Application entity for John Smith on the Accountant job offer. There are two levels of the Requisition tag in this example to reflect both the relation on the related entity. It is accidental that in this case both names are the same (see Applications/Application and ResidenceLocation/Location for more natural examples).

If we now want to set the date of entry for John Smith for the accountant job, we need to update the proper Application entity. To do so, we must find it among all his other possible applications. Since there is no identifier in the Application entity itself, we must use the Requisition Title field to find the Application. This is possible by specifying a search Target among the search attributes:

```
<Candidate>
     <EmailAddress searchType=”search” searchValue=jsmith@acme.com/>
     <Applications>
          <Application>
               <DateOfEntry>2006-06-01T14:15:00-04:00</DateOfEntry>
               <Requisition>
                    <Requisition>
                         <Title>
                              <value locale=”en” searchType=”search”
                              searchValue=”Accountant” searchTarget=”..”/>
                         </Title>
                    </Requisition>
               </Requisition>
          </Application>
     </Applications>
</Candidate>
```

This instruction is saying:

1. Search for the Candidate entity with e-mail *jsmith@acme.com*.
2. Among the Application entities of the Applications relation, find the one that has a linked Requisition with an English Title value of *Accountant*.
3. Once this Application entity is found, set its DateOfEntry field to *June 1st, 2006*

The searchTarget attribute value may take the following values: . (default), .., ../.., ../../.., etc. representing respectively the current entity, the parent entity, the grand-grand-parent entity, etc. When using a search target, only the entities are considered, not the actual XML tags. Here are the entity tags in the preceding example:

```
<Candidate>
     <EmailAddress searchType=”search” searchValue=jsmith@acme.com/>
     <Applications>
          <Application>
               <DateOfEntry>2006-06-01T14:15:00-04:00</DateOfEntry>
               <Requisition>
                    <Requisition>
                         <Title>
                              <value locale=”en” searchType=”search”
                              searchValue=”Accountant” searchTarget=”..”/>
                         </Title>
                    </Requisition>
               </Requisition>
          </Application>
     </Applications>
</Candidate>
```

Custom Fields

Custom fields are handled in their own separates section since they can be included in the format business model XML schema. (The UDF acronym comes from the term User Defined Field).

```
<Candidate>
```

```
     <EmailAddress searchType=”search” searchValue=jsmith@acme.com/>
     <UDFs>
          <UDF name=”DriverLicenseNumber”>JS-552-87610</UDF>
     </UDFs>
</Candidate>
```

Custom fields can also be multilingual, in which case, they follow the same syntax as a standard field:

```
<Candidate>
     <EmailAddress searchType=”search” searchValue=jsmith@acme.com/>
     <UDFs>
          <UDF name=”CertificationInformation”>
               <value locale=”en”>Charted Accountant, level 2</value>
          </UDF>
     </UDFs>
</Candidate>
```

You can also import selection type custom fields by searching within the custom fields:

```
<Candidate>
     <EmailAddress searchType=”search” searchValue=jsmith@acme.com/>
     <UDFs>
          <UDF name=”EyeColor”>
               <Color>
                    <Name>
                         <value locale=”en” searchType=”search”
                         searchValue=”Blue”/>
                    </Name>
               </Color>
          </UDF>
     </UDFs>
</Candidate>
```

However, it is not possible to search for an entity based on a custom field. For example, this instruction is not legal, because we cannot find a candidate based on his driver license number (even if such a custom field is known to be unique in the product):

```
<Candidate>
     <!—This is NOT legal; the instruction will be refused -->
     <UDFs>
          <UDF name=”DriverLicenseNumber” searchType=”search”
 searchValue=”JS-552-87610” searchTarget=”..”/>
     </UDFs>
</Candidate>
```

In some Taleo products, it is possible to use any characters in the custom field code. XML standards enforce a stricter rule for attribute values, hence custom field names must be escaped using a Java language standard where a special (non-alphanumeric) character is replaced by its ASCII hexadecimal code prefixed with an underscore character (see http://www.asciitable.com). For example, if the custom field name is “Special Note (optional)” without the quotes, then the instruction would be:

```
<Candidate>
     <EmailAddress searchType=”search” searchValue=jsmith@acme.com/>
     <UDFs>
          <UDF name=”Special_20Note_20_28optional_29”>Very interested</UDF>
     </UDFs>
</Candidate>
```

Import Document

An import document is quite straightforward; the only required attribute is the version of the model used, in our case http://www.taleo.com/ws/sample000/2006/06. The document is also a good place to talk about the namespace use in the XML syntax. Basically, anything in the import instruction is places

in the namespace identified by the URI of the model. The root element grouping the instructions must also be in the namespace. Anything else is in the generic namespace of the Taleo Connect Server itself.

```
<Document xmlns=”http://www.taleo.com/ws/integration/toolkit/2005/07”>
     <Attributes>
          <Attribute name=”version”>http://www.taleo.com/ws/
sample000/2006/06</Attribute>
     </Attributes>
     <Content>
          <Candidates xmlns=http://www.taleo.com/ws/sample000/2006/06>
               <Candidate>
                    <EmailAddress>jsmith@acme.com</EmailAddress>
                     …
               </Candidate>
               …
          </Candidates>
     </Content>
</Document                        …
```

The version needs to be specified explicitly as an attribute because the content of the document gets stored separately from the document attributes. The version then becomes mandatory to correctly interpret the content XML (that is in the right namespace).

Import Message

The import message wraps the document within a standard SOAP envelope and adds some header elements to satisfy the T-SOAP specifications. The most important here is the action element that indicates to interpret the integration instructions as imports.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsa="http://www.w3.org/2005/03/addressing" xmlns:itk="http://
www.taleo.com/ws/integration/toolkit/2005/07">
 <soapenv:Header>
  <wsa:MessageID>Import-Candidate-2006061151500</wsa:MessageID>
  <wsa:ReplyTo>
   <wsa:Address>http://www.taleo.com/ws/integration/toolkit/2005/07/
   addressing/queue</wsa:Address>
  </wsa:ReplyTo>
  <wsa:Action>http://www.taleo.com/ws/itk/prototype/2006/05/action/import
  </wsa:Action>
 </soapenv:Header>
 <soapenv:Body>
  <itk:submitDocument>
   <Document xmlns="http://www.taleo.com/ws/integration/toolkit/2005/07">
            …
         </Document>
  </itk:submitDocument>
 </soapenv:Body>
</soapenv:Envelope>
```

Export Feature

The export feature processes as single integration instruction to extract business entities of information from a Taleo product.

We will first see how to build an export instruction, then how to construct an export document and finally how to wrap the document within a T-SOAP message.

Export Instructions

Building an export instruction is much more involved than in the case of the import feature. As such, the description of an export instruction in the SQ-XML format is described fully in a separate document.

We will present here a simple example using the sample model. We will mostly talk about the result provided when using the T-XML mode or format.

The export instruction basically needs to specify what type of entity is exported, which entities are to be selected and what fields are to be extracted. Assuming we want the basic information of our candidate from the previous section, the instruction would look like the following:

```
<query alias=”SampleQuery” projectedClass=”Candidate”>
     <projections>
          <projection>
             <field path=”FirstName”/>
             <field path=”LastName”/>
         </Projection>
     </projections>
     <filterings>
          <filtering>
               <equal>
                    <field path=”EmailAddress”/>
                     <string>jsmith@acme.com</string>
               </equal>
          </filtering>
     </filtering>
</query>
```

Export Document

The export document is similar to the one created for in the import feature, but there are more attributes to set to define the behavior of the instruction.

Export Message

The export message is identical to the one created for the import feature, except of course for the action used to process the document.

Designing a Data Extraction Process

Overview

The first step in any design process is to gather the requirements. This is no different when building a data extraction process. However, there are a few specific information points to gather:

- The frequency of the extractions.
- The average or expected size of the extractions.
- The type of communication with the customer.
- The complexity of the mapping required between the Taleo and customer formats.
- The type of data required from the Taleo application and its relation to the actual model.

Specific Considerations

The main objective of the overall export environment is to extract the data quickly from the application to ensure maximum availability and responsiveness to end users, and execute the transformation and routing activities in the customer's own environment.

Data Transformation

One choice regarding the projection of the constructed SQ-XML is whether to use the report type format to execute part or the whole of the customer specific transformation or simply extract the raw data and transform it at a later stage. The recommendation here is to use the Selection Query for transformation only if:

1. The transformation is simple (date formatting, concatenations, simple DECODE functions) AND
2. The transformation can be completely expressed in SQ-XML (thus saving a customer specific integration package altogether).

If the second condition fails, then a customer specific package will have to exist anyway, and it is thus optimal to group all the transformation information in it. One immediate gain is that the SQ-XML document will be much lighter and easier to read and maintain. If the first condition fails, then performance will degrade when executing the requests on the application and database servers.

Determining the Business Model Mapping

The export service application schema

Once the data to be extracted has been determined at a functional level, it must be expressed in terms of the business model. Included within the WSDL file of the export web service or as a separate XSD file, the actual schema of the business model is the definitive reference that determines what data is exposed and thus possibly extracted. The schema defines the entities and for each an exhaustive list of fields and relationships. Among other things, it is based on this schema that you can determine a projection suitable for the entity-based format.

Export Extraction Language

The export service used the Taleo Selection Query framework to express and execute data extractions. The corresponding extraction language supports an XML serialization that is used as the main parameter of the request message.

For a high-level perspective, the Selection Query language is designed as standard SQL but based on a business model and not a physical representation (data model). The main strength of SQ-XML is that all the business model relationships are implicitly assumed.

Export Result Document Formats

The current version of the export service offers two different output formats: XML and T-XML. The first two formats are used for "report"-type extractions where the data is represented as a flat result set, that is, as a series of identically structured records.

In simple cases, they all provide exactly the same information as shown below:

XML format:

```
<record>
   <field name=”FirstName”>John</field>
   <field name=”LastName”>Doe</field>
   <field name=”Email”>johndoe@taleo.com</field>
   <field name=”Department.Name”>Finance</field>
</record>
```

The T-XML format is used for entity type extractions and provides structured result data.

T-XML format:

```
<User>
   <FirstName>John</FirstName>
   <LastName>Doe</LastName >
   <Email>johndoe@taleo.com</Email>
```

```
   <Department>
      <Department>
         <Name>Finance</Name>
      </Department>
   </Department>
</User>
```

For complex cases, both format types serve a different purpose. In the case where multiple levels of information are requested, the T-XML format is best. Consider the case of exporting candidates with the work experiences attached to each application.

First in the “natural” T-XML format:

```
<Candidate>
   <Email>janedoe@taleo.com</Email>
   <FirstName>Jane</FirstName>
   <LastName>Doe</LastName >
   <Applications>
      <Application>
         <ContestNumber>Teacher-2235</ContestNumber>
         <WorkExperiences>
            <WorkExperience>
               <Company>Bishop’s university</Company>
                <StartDate>2000-01-15</StartDate>
                <EndDate>2002-05-31</EndDate>
              </WorkExperience>
            <WorkExperience>
               <Company>McGill’s university</Company>
                <StartDate>2004-08-01</StartDate>
                <EndDate>2004-12-15</EndDate>
              </WorkExperience>
         </WorkExperiences>
      <Application>
      <Application>
         <ContestNumber>Technician-998J</ContestNumber>
         <WorkExperiences>
            <WorkExperience>
               <Company>Biochemical Inc</Company>
                <StartDate>1998-10-15</StartDate>
                <EndDate>1999-12-31</EndDate>
              </WorkExperience>
            <WorkExperience>
               <Company>Brown chemicals</Company>
                <StartDate>1998-01-01</StartDate>
                <EndDate>1998-10-12</EndDate>
              </WorkExperience>
         </WorkExperiences>
      <Application>
   <Applications>
</Candidate>
```

However, when the data extracted does not follow the actual structure of the business model, it is not possible to use the T-XML format. The XML format report type has no such limitation.

For example, consider the case where all candidates having applied during Q1 must be extracted along with the number of applications made per month. Only the report-type format could support such a request, giving the following XML format.

```
<ExportXML>
   <record>
      <field name=”Email”>zblack@taleo.com</field>
      <field name=”AppsInJan”>0</field>
```

```
      <field name=”AppsInFeb”>4</field>
      <field name=”AppsInMar”>1</field>
   </record>
   <record>
      <field name=”Email”>mbrown@taleo.com</field>
      <field name=”AppsInJan”>0</field>
      <field name=”AppsInFeb”>0</field>
      <field name=”AppsInMar”>3</field>
   </record>
   <record>
      <field name=”Email”>rwhite@taleo.com</field>
      <field name=”AppsInJan”>5</field>
      <field name=”AppsInFeb”>0</field>
      <field name=”AppsInMar”>2</field>
   </record>
   …
</ExportXML>
```

Summary of result document types and formats

The following table summarized the result document types and formats:

Type	Format
Report	XML
Entity	T-XML

Export Query Documentation

Refer to the “Taleo Connect Selection Query Specification.pdf” document for more information on building export queries.

Technology Matrix

The following matrix presents the technologies on which Taleo has tested the BulkAPI.

The interoperability nature of web services should facilitate the usage of other languages, stub generators, web service stacks and bindings. But Taleo can't test all the possibilities that exist out there.

Acceptance to investigate problems reported with other technologies will be at Taleo discretion.

Language	Stub Generation	WS Stack	Binding	Supported by Taleo	Comment
SOAP UI	N/A	N/A	N/A	Yes	
Java	CXF WSDLTOJAVA	CXF	JAXB	Yes	
Java	CXF WSDLTOJAVA	CXF	XMLBean	Yes but not with Attachment	XMLBean doesn't support MTOM attachment
.NET C#		WS		No	
.NET C#	WseWsdl3.exe	WSE		No	Required works at the C programming level (class override)
.NET C#	Svcutil.exe	WCF		Yes	
Others				No	

Appendix

Web Service Limits

The following web service limits apply to all web services by default.

Since	Default Value	Comment	Behavior
7.5 SP2	20	The maximum number of integration threads allowed by JVM.	The Web Service policy manager ensures that no more than x web service calls runs simultaneously. When the maximum number of running web service call is reached, any new web service call will be rejected with an error message.
7.5 SP2	-1 (no minimum)	The minimum export rate that is tolerated by the integration toolkit. This setting enables the integration toolkit to stop the processing of an export if the current data export rate is too low. The data export rate indicates the number of entities exported in an hour.	When serializing export results (after each entity or after each CSV report row), check the export rate. If the export rate is lower than the allowed export rate, the export is cancelled.
7.5 SP2	200	The maximum number of records that can be returned by an export request sent to the integration export service. If a request would return more results, an error is returned instead.	The maximum number of records is check before serializing results.
7.5 SP2	2048000 (2 MB)	The maximum size (in bytes) of the response generated by an export request sent to the integration export service. If the service notices that the response size is larger than this value, an error is returned instead.	When serializing export results (after each entity or after each CSV report row), check the export size. If export size is greater than the allowed size, the export is cancelled.
7.5 SP2	90 (1 minute 30 sec)	The maximum time in seconds permitted for an export request. This setting enables the integration toolkit to stop an export request if it takes more than the specified amount of time.	When serializing export results (after each entity or after each CSV report row), check the elapsed time of the export. If the elapsed time is greater than the maximum export time, the export is cancelled.
7.5 SP2	25000 Corresponds to an average of 17 calls/ minute	The maximum number of web service calls that can be invoked daily.	The maximum is checked before invoking the service. If the daily invocation maximum is already reached, the service is not invoked and an error is returned to the user.
7.5 SP2	250000 Corresponds to an average of 10 exported entities/call for 17 calls/	The maximum number of records that can be exported daily.	The maximum is checked before extracting data. E.g.: If the limit is 250 000 records per day and there is already 200 000 exported records for the current day, an extract returning 50 001 records

Since	Default Value	Comment	Behavior
	minute (more/ call if less calls/minute)		will not run because this would exceed the daily limit.
Always	N/A	Certain WSDLs may contain two types where the only difference in their names is that one is all in small-case letters and the other contains upper-case letters. This peculiarity is a limiting factor for the Visal Basic language as it is not case sensitive.	When a customer tries to use this type of WSDL to generate a stub in Visual Basic, the generated class will not compile since two different classes will have the same name.

Version 7.5 Namespace Limitations

You must be aware of the following limitations generated by the way that the namespaces are used in Version 7.5 WSDLs. (Note that these limitations have been fixed in Taleo 10 SP1 and later).

Normally, in Java, these limitations can be worked around by assigning the different usages of a namespace to different packages and then eliminate any possible collision in the Java code. But with the XML Bean data binding, this solution is not possible. Here is an extract from the XML Bean documentation:

"Note: XMLBeans doesn't support using two or more sets of java classes (in different packages) mapped to schema types/elements that have the same names and target namespaces, using all in the same class loader. Depending on the direction you are using for the java classes to schema types mapping, some features might not work correctly. This is because even though the package names for the java classes are different, the schema location for the schema metadata (.xsb files) is the same and contains the corresponding implementing java class, so the JVM will always pick up the first on the classpath. This can be avoided if multiple class loaders are used."

Here is a list of know limitations that are caused by this XML Bean limitation:

Namespace Usage Case #1

The same integration toolkit namespace is used for two different contents.

The import services are using the integration toolkit namespace to specify a different content that the export ones.

Known Limitation :

The impact is that it might not be possible to use an import service and an export service in the same java program.

Namespace Usage Case #2

The import service parameters definitions are put in the data model namespace.

For example, the Department service create operation takes a tns:createRequest message in parameter that points to a create element defined in the data model:

```
<wsdl:message name="createRequest">
   <wsdl:part name="parameters" element="tns:create"/>
</wsdl:message>
```

In the data model, it points to a Department entity.

```
<xsd:element name="create">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="department" type="tns:Department"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

And if we take the same example but with the Position service:

The Position service create operation takes a tns:createRequest message in parameter that points to a create element defined in the data model:

```
<wsdl:message name="createRequest">
   <wsdl:part name="parameters" element="tns:create"/>
</wsdl:message>
```

In the data model, it points to a Position entity.

```
<xsd:element name="create">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="position" type="tns:Position"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Known Limitation :

The impact is that it might not be possible to use two different import services in the same java program.

Namespace Usage Case #3

The export service parameters definitions are put in the integration toolkit namespace.

For example, in the FindService, the findPartialEntities operation response is a findPartialEntitiesResponse element that points to a findPartialEntitiesResponse element defined in the integration toolkit namespace:

```
<wsdl:message name="findPartialEntitiesResponse">
   <wsdl:part name="parameters"
              element="tns:findPartialEntitiesResponse"/>
</wsdl:message>
```

This element points to the Entities one that is also define in the integration toolkit namespace:

```
<xsd:element name="findPartialEntitiesResponse">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element maxOccurs="1" minOccurs="1" name="Entities"
                      nillable="true" type="tns:Entities"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

And finally the Entities element points to the Entity element define in the data model:

```
<xsd:complexType name="Entities">
   <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" ref="ns1:Entity"/>
  </xsd:sequence>
</xsd:complexType>
```

Known Limitation :

The impact is that it is not possible to use the FindService with two different data model in the same Java program because the Entities element defined in the ITK point to a Entity element that is defined in the data model namespace. Since the Entity element can be defined in multiple data model, the Java program will not be able to determine which one to use.

This will be the case when a find must be done in the art product and another one in the smartorg product.

Another case is when a FindService is use to search in two different version of the same product. An example of this is a search done in Smartorg 7.5 and then done in Smartorg 10.

Namespace Usage Case #4

The data model define in an import service is different than the one define in an export service but they are both using the same namespace.

Therefore the definition of an entity for an import service might be different that the one for an export service. For example, the definition of a candidate entity in import is different than the one in an export.

Known Limitation :

In Java, it is not possible to cast the find service result to the correct object if the import service is imported before the export one.

Export Query Performance Throttles

The following export query performance throttles apply to the EXPORT web service only.

Since	Default Value	Comment	Behavior
7.5 SP2	5	The maximum depth of relations that can be specified in an export query (max.relation.deepness)	All performance throttles are checked before extracting the data.
7.5 SP2	-1	The maximum number of relations that can be specified in an export query (max.relation.count)	
7.5 SP2	1	The maximum number of subqueries that can be specified in an export query (max.subquery.count)	
7.5 SP2	5	The maximum number of fields that can be specified to filter an export query (max.filtering.field.count)	
7.5 SP2	5	The maximum number of fields that can be specified to group fields in an export query (max.grouping.field.count) A grouping field represents a value used to group query results. In SQL, it will be pushed into the GROUP BY clause.	
7.5 SP2	5	The maximum number of fields that can be specified to join other queries in an export query (max.jointing.field.count) A joint field represents a filter used to join a parent and a sub-query. In SQL, the filter will be pushed into the WHERE clause of the parent query.	
7.5 SP2	1	The maximum number of sub-queries that can be specified in an export query (max.subquery.level.count)	
7.5 SP2	100	The maximum number of fields that can be projected in an export query (max.projection.field.count) A projection field represents a value returned by a query. In SQL, it will be pushed into the SELECT clause.	

Web Services Client Sample Code

To walk through the sample code, we will use a sample business case. Let's assume you are a Taleo Partner and want to export a candidate from your application and import it into Taleo systems. Then have this candidate apply to a Taleo job offer (requisition).

Using Taleo API, you will use the following two Web services to implement this business case:

- **Professional 7.5 CandidateService:** This service allows your application to create and update a candidate in Taleo systems
- **Professional 7.5 FindService:** This service allows your application to export a requisition out of Taleo systems

Here is the work flow using the Taleo API:

1. Import a requisition. That is, export a requisition from Taleo systems so you will get a valid requisition identifier on which will apply your candidate.
2. Export your candidate. That is, import the candidate into Taleo systems and have the candidate apply on the requisition you previously imported in your application.

Once you have imported the WSDL files (here: `CandidateService` and `FindService`), you can begin building client applications that use the API. Use the following samples to create a basic client application. Comments embedded in the sample explain each section of code.

Sample Java Code

This section walks you through a sample Java client application that uses the Apache Axis2 SOAP client for Java. The purpose of this sample application is to show the required steps for logging into the login server and to demonstrate the invocation and subsequent handling of several API calls.

This sample application performs the following main tasks:

1. Exports all Taleo requisitions using pagination in order to retrieve the first Taleo requisition identifier from the last page of results.
2. Using this requisition identifier, imports a new candidate to Taleo system and has that candidate apply on that requisition.
3. Outputs the email address of the successfully imported candidate.

Note: The values used in this sample are hard coded in the code, but you could imagine having this data retrieved from your own system, i.e. your internal database.

```
/*
 * @(#)TaleoSamplesXmlBeans.java
 *
 * Copyright 1999-2006 by Taleo Corporation,
 * 330 St-Vallier East, Suite 400, Quebec city, Quebec, G1K 9C5, CANADA
 * All rights reserved
 */
package com.taleo.demo;
import com.taleo.art750.candidate.Candidate;
import com.taleo.art750.candidate.CandidateServiceStub;
import com.taleo.art750.candidate.CreateDocument;
import com.taleo.art750.candidate.CreateResponseDocument;
import com.taleo.art750.candidate.PreselectionApplication;
import com.taleo.art750.candidate.SearchableMultilingualSearchOnlyField;
import com.taleo.art750.candidate.SearchableMultilingualStringField;
import com.taleo.art750.candidate.SearchableStringField;
import com.taleo.art750.candidate.WebServiceFault;
import com.taleo.art750.candidate.Candidate.Applications;
import com.taleo.art750.candidate.Candidate.CurrentJob;
import com.taleo.art750.candidate.PreselectionApplication.ApplicationState;
```

```
import com.taleo.art750.candidate.PreselectionApplication.Requisition;
import com.taleo.art750.candidate.SearchableMultilingualStringField.Value;
import com.taleo.export.itk.Entities;
import com.taleo.export.itk.Field;
import com.taleo.export.itk.Filtering;
import com.taleo.export.itk.FindPartialEntitiesDocument;
import com.taleo.export.itk.FindPartialEntitiesResponseDocument;
import com.taleo.export.itk.FindServiceStub;
import com.taleo.export.itk.IncludedIn;
import com.taleo.export.itk.List;
import com.taleo.export.itk.Long;
import com.taleo.export.itk.Projection;
import com.taleo.export.itk.Query;
import com.taleo.export.itk.Sorting;
import com.taleo.export.itk.SqxmlQuery;
import com.taleo.export.itk.String2StringMap;
import com.taleo.export.itk.WebServiceFaultException0;
import
 com.taleo.export.itk.FindPartialEntitiesResponseDocument.FindPartialEntitiesResponse;
import com.taleo.export.itk.Query.Filterings;
import com.taleo.export.itk.Query.Projections;
import com.taleo.export.itk.Query.Sortings;
import com.taleo.export.itk.String2StringMap.Entry;
import java.rmi.RemoteException;
import java.security.InvalidParameterException;
import org.apache.axis2.AxisFault;
import org.apache.axis2.transport.http.HTTPConstants;
import org.apache.axis2.transport.http.HttpTransportProperties;
/**
 * <code>TaleoSamplesXmlBeans</code> is a demo application using the
 FindService and
 * CandidateService web services, available on the Taleo enterprise
 product, version 750 SP4+
 *
 * @author pmarsteau
 * @since 1.0
 */
public final class TaleoSamplesXmlBeans
{
    /**
     * Exports the latest Requisition ContestNumber (external number) and
 create one Candidate
     * applying to that requisition.
     *
     * @param pArgs parameters
     * @throws RemoteException if any remote exception occurs
     * @since 1.0
     */
    public static void main(String[] pArgs) throws RemoteException
    {
        if (pArgs.length == 3)
        {
            TaleoSamplesXmlBeans demoImpl = new TaleoSamplesXmlBeans();
            String contestNumber = demoImpl.exportRequisition(pArgs[0],
 pArgs[1], pArgs[2], 1);
            System.out.println("Successfully exported requisition number: "
 + contestNumber);
            long candidateNo =
                demoImpl.importCandidate(pArgs[0], pArgs[1], pArgs[2],
 contestNumber);
```

```
            System.out.println("Candidate successfully created with number:
 " + candidateNo);
        }
        else
        {
            throw new InvalidParameterException(
                "Invalid number of arguments specified (should be 3)\n"
                    + "Usage: java com.taleo.demo.TaleoSamplesXmlBeans
 [URL] [username] [password]");
        }
    }
    /**
     * Exports the ContestNumber of the last open and posted requisition
 available in the Taleo
     * database.
     *
     * @param pEndpoint the URL to the Taleo server in the form http://
host:port/servlets/soap
     * @param pUsername the username to login with (using BASIC
 authentication)
     * @param pPassword the password to login with (using BASIC
 authentication)
     * @param pPageIndex the page index to retrieve (see export pagination)
     * @return the resulting ContestNumber
     * @throws RemoteException if any error occurs while invoking the
 remote server
     * @since 1.0
     */
    private String exportRequisition(String pEndpoint, String pUsername,
 String pPassword,
        int pPageIndex) throws RemoteException
    {
        FindServiceStub stub = new FindServiceStub(pEndpoint + "?
ServiceName=FindService");
        HttpTransportProperties.Authenticator auth = new
 HttpTransportProperties.Authenticator();
        auth.setUsername(pUsername);
        auth.setPassword(pPassword);
        auth.setPreemptiveAuthentication(true); // activate preemptive
 authentication

 stub._getServiceClient().getOptions().setProperty(HTTPConstants.AUTHENTICATE,
 auth);
        FindPartialEntitiesDocument findDoc =
 FindPartialEntitiesDocument.Factory.newInstance();
        FindPartialEntitiesDocument.FindPartialEntities find =
 findDoc.addNewFindPartialEntities();
        // set mapping version being used
        find.setMappingVersion("http://www.taleo.com/ws/art750/2006/12");
        // create SQ query
        SqxmlQuery sqxmlquery = find.addNewQuery();
        Query query = sqxmlquery.addNewQuery();
        query.setAlias("Find Open and Posted Requisitions");
        query.setProjectedClass("Requisition");
        Projections projections = query.addNewProjections();
        Projection contestNumberProj = projections.addNewProjection();
        Field contestNumberField = contestNumberProj.addNewField();
        contestNumberField.setPath("ContestNumber");
        Projection jobInfoTitleProj = projections.addNewProjection();
        Field jobInfoTitleField = jobInfoTitleProj.addNewField();
        jobInfoTitleField.setPath("JobInformation,Title");
```

```
        Filterings filterings = query.addNewFilterings();
        Filtering filtering = filterings.addNewFiltering();
        IncludedIn includedIn = filtering.addNewIncludedIn();
        Field stateField = includedIn.addNewField();
        stateField.setPath("State,Number");
        List stateList = includedIn.addNewList();
        Long open = stateList.addNewLong();
        open.setLongValue(3); // open
        Long posted = stateList.addNewLong();
        posted.setLongValue(13); // posted
        Sortings sortings = query.addNewSortings();
        Sorting sorting = sortings.addNewSorting();
        Field contestNumberSortingField = sorting.addNewField();
        contestNumberSortingField.setPath("ContestNumber");
        // set attributes
        String2StringMap attrs = find.addNewAttributes();
        Entry entry = attrs.addNewEntry();
        entry.setKey("pageindex");
        entry.setValue(String.valueOf(pPageIndex));
        // invoke service
        FindPartialEntitiesResponseDocument response;
        try
        {
            response = stub.findPartialEntities(findDoc);
        }
        catch (WebServiceFaultException0 e)
        {
            System.err.println("Error while creating the candidate: "
                + e.getFaultMessage().getWebServiceFault().getMessage());
            throw new RuntimeException(e);
        }
        // get and handle results
        FindPartialEntitiesResponse rsp =
 response.getFindPartialEntitiesResponse();
        Entities entities = rsp.getEntities();
        // results are pagined, that is, are fetched page by page (max 200
 entities per page)
        // pagecount = total number of pages and
        // pageindex = current page index
        System.out.println("Fetched page: " + entities.getPageIndex() + "/"
            + entities.getPageCount());
        if (!entities.getPageCount().equals(entities.getPageIndex()))
        {
            // export next page
            return exportRequisition(pEndpoint, pUsername, pPassword, +
+pPageIndex);
        }
        else
        {
            com.taleo.export.art750.Requisition requisition =
                (com.taleo.export.art750.Requisition)
 entities.getEntityArray()[entities
                    .sizeOfEntityArray() - 1];
            return requisition.getContestNumber();
        }
    }
    /**
     * Import a candidate that will apply on the specified requisition
 contest number
     *
```

```
     * @param pEndpoint the URL to the Taleo server in the form http://
host:port/servlets/soap
     * @param pUsername the username to login with (using BASIC
 authentication)
     * @param pPassword the password to login with (using BASIC
 authentication)
     * @param pContestNumber the contest number of the requisition on which
 the candidate will apply
     * @return the Taleo unique identifier associated to the new created
 candidate
     * @throws RemoteException if any error occurs while invoking the
 remote server
     * @since 1.0
     */
    private long importCandidate(String pEndpoint, String pUsername, String
 pPassword,
        String pContestNumber) throws RemoteException
    {
        CandidateServiceStub stub =
            new CandidateServiceStub(pEndpoint + "?
ServiceName=CandidateService");
        HttpTransportProperties.Authenticator auth = new
 HttpTransportProperties.Authenticator();
        auth.setUsername(pUsername);
        auth.setPassword(pPassword);
        auth.setPreemptiveAuthentication(true); // activate preemptive
 authentication

 stub._getServiceClient().getOptions().setProperty(HTTPConstants.AUTHENTICATE,
 auth);
        CreateDocument createDoc = CreateDocument.Factory.newInstance();
        CreateDocument.Create create = createDoc.addNewCreate();
        Candidate candidate = create.addNewCandidate();
        // ****************************
        // Usage of a searchable fields
        // ============================
        // - first add a value
        // - set its value (to be set)
        // - set its search type (optional)
        // - set its search target (optional)
        // - set its search value (to be searched, optional)
        // if none of search type, search target and search value are
 specified, the value is set
        // without prior search. Some fields do not allow edition, but only
 being searched.
        // ****************************
        SearchableStringField address = candidate.addNewAddress();
        address.setStringValue("123 Main Street");
        SearchableStringField address2 = candidate.addNewAddress2();
        address2.setStringValue("Apartment 1");
        SearchableStringField city = candidate.addNewCity();
        city.setStringValue("Seattle");
        // ******************************
        // Usage of a multilingual fields
        // ==============================
        // - first add a value
        // - set its locale
        // - set its value
        // ******************************
        CurrentJob curJob = candidate.addNewCurrentJob();
```

```
       com.taleo.art750.candidate.CurrentJob curJobElement =
curJob.addNewCurrentJob();
       SearchableMultilingualStringField jobTitle =
curJobElement.addNewCurrentJobJobTitle();
       Value titleValue = jobTitle.addNewValue();
       titleValue.setLocale("en");
       titleValue.setStringValue("Sr Developer");
       Applications applications = candidate.addNewApplications();
       PreselectionApplication presel =
applications.addNewPreselectionApplication();
       ApplicationState appstatecontainer =
presel.addNewApplicationState();
       com.taleo.art750.candidate.ApplicationState appstate =
           appstatecontainer.addNewApplicationState();
       SearchableMultilingualSearchOnlyField appstatedesc =
appstate.addNewDescription();

com.taleo.art750.candidate.SearchableMultilingualSearchOnlyField.Value
descvalue =
           appstatedesc.addNewValue();
       descvalue.setLocale("en");
       descvalue.setSearchValue("New");

descvalue.setSearchType(SearchableMultilingualSearchOnlyField.Value.SearchType.SEARCH);
       // set a unique Email address. Providing an email address of an
existing Candidate will
       // cause the service invocation to throw an exception
       SearchableStringField emailAddress =
candidate.addNewEmailAddress();

emailAddress.setStringValue("demoXmlBeans011@unique.email.adresss.com");
       // set a requisition on which this candidate is applying
       Requisition reqcontainer = presel.addNewRequisition();
       com.taleo.art750.candidate.Requisition requisition =
reqcontainer.addNewRequisition();
       SearchableStringField contestnumber =
requisition.addNewContestNumber();

contestnumber.setSearchType(SearchableStringField.SearchType.SEARCH);
       // set a valid requisition number
       contestnumber.setSearchValue(pContestNumber);
       try
       {
           // invoke service
           CreateResponseDocument createResponse = stub.create(createDoc);
           return createResponse.getCreateResponse().getNumber();
       }
       catch (WebServiceFault e)
       {
           System.err.println("Error while creating the candidate: "
               + e.getFaultMessage().getWebServiceFault().getMessage());
           throw new RuntimeException(e);
       }
       catch (AxisFault e)
       {
           System.err.println("Error while creating the candidate: " +
e.getMessage() + "\n"
               + e.getCause());
           throw e;
       }
   }
```

```
}
```

C# Sample Code

This section walks you through a sample C# client application. The purpose of this sample application is to show the required steps for logging in and to demonstrate the invocation and subsequent handling of several API calls.

This sample application performs the following main tasks:

1. Exports all Taleo requisitions using pagination in order to retrieve the first Taleo requisition identifier from the last page of results.
2. Using this requisition identifier, imports a new candidate to Taleo system and that candidate apply on that requisition.
3. Outputs the email address of the successfully imported candidate.

Note: The values used in this sample are hard coded in the code, but you could imagine having this data retrieved from your own system, i.e. your internal database.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Web.Services.Protocols;
namespace SamplesNamespace
{
    class Samples
    {
        static void Main(string[] args)
        {
            Samples samples = new Samples();
            samples.run();
        }
        private String getUserInput(String prompt)
        {
            Console.WriteLine(prompt);
            return Console.ReadLine();
        }
        private void run()
        {
            String endpoint = getUserInput("Enter Endpoint");
            String username = getUserInput("Enter Username");
            String password = getUserInput("Enter Password");
            String reqId;
            try
            {
                // export a requisition out of Taleo systems and retrieves
 its identifier
                reqId = exportRequisition(endpoint, username, password);
            }
            catch (SoapException e)
            {
                Console.WriteLine("Error while exporting the requisitions:
 " + e.Detail.InnerText);
                // abord process
                return;
            }
            try
            {
                importCandidate(endpoint, username, password, reqId);
            }
            catch (SoapException e)
            {
```

```
                Console.WriteLine("Error while creating the candidate: " +
e.Detail.InnerText);
            }
        }
        private void importCandidate(String pEndpoint, String pUsername,
String pPassword,
            String pReqID)
        {
            //
*****************************************************************************
            // 1 - The first part is to instantiate the Web services using
the URL/credentials
            //
*****************************************************************************
            String endpoint = pEndpoint + "?ServiceName=CandidateService";
            taleo.CandidateService.CandidateService candidateService = new
taleo.CandidateService.CandidateService();
            CredentialCache credCache = new CredentialCache();
            credCache.Add(new Uri(endpoint), "Basic", new
NetworkCredential(pUsername, pPassword));
            candidateService.Credentials = credCache;
            candidateService.Url = endpoint;
            candidateService.PreAuthenticate = true;

            //
*****************************************************************************
            // 2 - now create a document that will contain the
findPartialEntities() request
            //
*****************************************************************************
            taleo.CandidateService.Candidate candidate = new
taleo.CandidateService.Candidate();

            // a "searchable" string field allows a) to edit a field value
or b) to use a field
            // value already existing in Taleo systems. In the following
lines, we just want to set
            // the values.
            candidate.Address = new
taleo.CandidateService.searchableStringField();
            candidate.Address.Value = "123 Main Street";
            candidate.Address2 = new
taleo.CandidateService.searchableStringField();
            candidate.Address2.Value = "Apartment 1";
            candidate.City = new
taleo.CandidateService.searchableStringField();
            candidate.City.Value = "Seattle";

            // a "searchable multilingual" string field is identical to a
"searchable" string but
            // further allows to specify a specific locale. In the
following lines, we just want to
            // set the value using the english locale ("en")
            candidate.CurrentJob = new
taleo.CandidateService.CandidateCurrentJob();
            candidate.CurrentJob.CurrentJob = new
taleo.CandidateService.CurrentJob();
            candidate.CurrentJob.CurrentJob.CurrentJobJobTitle = new
taleo.CandidateService.searchableMultilingualStringFieldValue[1];
            candidate.CurrentJob.CurrentJob.CurrentJobJobTitle[0] = new
taleo.CandidateService.searchableMultilingualStringFieldValue();
```

```
            candidate.CurrentJob.CurrentJob.CurrentJobJobTitle[0].locale =
"en";
            candidate.CurrentJob.CurrentJob.CurrentJobJobTitle[0].Value =
"Sr Developer";
            candidate.Applications = new
taleo.CandidateService.PreselectionApplication[1];
            candidate.Applications[0] = new
taleo.CandidateService.PreselectionApplication();

            // The following lines uses also use a "searchable
multilingual" string field, but this
            // time, we want to set a value already existing in Taleo
systems instead of setting a
            // new value. The search type is therefore set to "SEARCH" and
the search value to the
            // value we are referring to.
            candidate.Applications[0].ApplicationState = new
taleo.CandidateService.PreselectionApplicationApplicationState();
            candidate.Applications[0].ApplicationState.ApplicationState =
new taleo.CandidateService.ApplicationState();

candidate.Applications[0].ApplicationState.ApplicationState.Description =
new taleo.CandidateService.searchableMultilingualSearchOnlyFieldValue[1];

candidate.Applications[0].ApplicationState.ApplicationState.Description[0]
= new taleo.CandidateService.searchableMultilingualSearchOnlyFieldValue();

candidate.Applications[0].ApplicationState.ApplicationState.Description[0].locale
= "en";

candidate.Applications[0].ApplicationState.ApplicationState.Description[0].searchType
=
taleo.CandidateService.searchableMultilingualSearchOnlyFieldValueSearchType.search;

candidate.Applications[0].ApplicationState.ApplicationState.Description[0].searchValue
= "New";

            // set a unique Email address. Providing an email address of an
existing Candidate will
            // cause the service invocation to throw an exception.
            candidate.EmailAddress = new
taleo.CandidateService.searchableStringField();
            String uniqueEmail = "sample" + DateTime.Now.Ticks +
"@unique.email.address.com";
            candidate.EmailAddress.Value = uniqueEmail;
            // set a requisition on which this candidate is applying and
set the specified
            // requisition identifier to it
            candidate.Applications[0].Requisition = new
taleo.CandidateService.PreselectionApplicationRequisition();
            candidate.Applications[0].Requisition.Requisition = new
taleo.CandidateService.Requisition();
            candidate.Applications[0].Requisition.Requisition.ContestNumber
= new taleo.CandidateService.searchableStringField();

candidate.Applications[0].Requisition.Requisition.ContestNumber.searchType
= taleo.CandidateService.searchableStringFieldSearchType.search;

candidate.Applications[0].Requisition.Requisition.ContestNumber.searchTypeSpecified
= true;
```

```
candidate.Applications[0].Requisition.Requisition.ContestNumber.searchValue
= pReqID;

            //
************************************************************************
            // 3 - invoke service
            //
************************************************************************
            candidateService.create(candidate);
            //
************************************************************************
            // 4 - get and handle results
            //
************************************************************************
            // the candidateSvc.create() does not return any result, and if
no error is thrown,
            // it means the candidate was successfully inserted into Taleo
systems
            Console.WriteLine("Candidate successfully imported in Taleo
system using email address: " + uniqueEmail);
        }
        private String exportRequisition(String pEndpoint, String
pUsername, String pPassword)
        {
            //
************************************************************************
            // 1 - The first part is to instantiate the Web services using
the URL/credentials
            //
************************************************************************
            String endpoint = pEndpoint + "?ServiceName=FindService";
            taleo.FindService.FindService findService = new
taleo.FindService.FindService();
            CredentialCache credCache = new CredentialCache();
            credCache.Add(new Uri(endpoint), "Basic", new
NetworkCredential(pUsername, pPassword));
            findService.Credentials = credCache;
            findService.Url = endpoint;
            findService.PreAuthenticate = true;
            //
************************************************************************
            // 2 - create the export request and invoke service
            //
************************************************************************
            // Taleo API has some export limits that prevent you to fetch
all elements at once, if
            // it exceeds 200 elements. To retrieve all elements, you will
need to paginate the
            // search by retrieving the results page by page
            int page = 1;
            taleo.FindService.Entities entities =
exportRequisition(findService, page);
            while (!entities.pageCount.Equals(entities.pageIndex))
            {
                entities = exportRequisition(findService, ++page);
            }
            //
************************************************************************
            // 4 - get and handle results
```

```
            //
 ********************************************************************************
            // retrieve first entity of the last fetched page
            taleo.FindService.Entity entity = entities.Entity[0];
            if
 (entity.GetType().Equals(typeof(taleo.FindService.Requisition)))
            {
                String reqId =
 ((taleo.FindService.Requisition)entity).ContestNumber;
                bool hasBeenApproved =
 Convert.ToBoolean(((taleo.FindService.Requisition)entity).HasBeenApproved);
                Console.WriteLine("Candidate will apply on requisition ID:
 " + reqId);
                return reqId;
            }
            else
            {
                throw new Exception("Expected to receive a Requisition");
            }
        }
        private taleo.FindService.Entities exportRequisition(
            taleo.FindService.FindService pFindSvc,
            int pPageIndex)
        {
            //
 ********************************************************************************
            // 1 - create a document that will contain the
 findPartialEntities() operation request
            //
 ********************************************************************************
            // 1a - set mapping version being used
            String mappingVersion = "http://www.taleo.com/ws/
art750/2006/12";
            // 1b - create SQ query
            taleo.FindService.sqxmlQuery sqxmlquery = new
 taleo.FindService.sqxmlQuery();
            sqxmlquery.query = new taleo.FindService.query();
            sqxmlquery.query.alias = "Find Open and Posted Requisitions";
            sqxmlquery.query.projectedClass = "Requisition";
            // query part 1: create the projections
            sqxmlquery.query.projections = new
 taleo.FindService.queryProjections();
            sqxmlquery.query.projections.projection = new
 taleo.FindService.projection[2];
            taleo.FindService.field contestnumber = new
 taleo.FindService.field();
            contestnumber.path = "ContestNumber";
            sqxmlquery.query.projections.projection[0] = new
 taleo.FindService.projection();
            sqxmlquery.query.projections.projection[0].Item =
 contestnumber;
            taleo.FindService.field jobInfoTitleProj = new
 taleo.FindService.field();
            jobInfoTitleProj.path = "JobInformation,Title";
            sqxmlquery.query.projections.projection[1] = new
 taleo.FindService.projection();
            sqxmlquery.query.projections.projection[1].Item =
 jobInfoTitleProj;
            // query part 2: create the filterings
            sqxmlquery.query.filterings = new
 taleo.FindService.queryFilterings();
```

```
            sqxmlquery.query.filterings.filtering = new
taleo.FindService.filtering[1];
            taleo.FindService.includedIn statefilter = new
taleo.FindService.includedIn();
            // create field on which to filter on
            taleo.FindService.field stateField = new
taleo.FindService.field();
            stateField.path = "State,Number";
            // create list of valid state numbers to filter on
            taleo.FindService.list stateList = new
taleo.FindService.list();
            // list contains 2 longs, 3 for open requisition, 13 for posted
requisitions
            taleo.FindService.@long[] listValues = new
taleo.FindService.@long[2];
            listValues[0] = new taleo.FindService.@long();
            listValues[0].Value = 3;
            listValues[1] = new taleo.FindService.@long();
            listValues[1].Value = 13;
            stateList.Items = listValues;
            // set the field and the list to the includedIn filter
            statefilter.Items = new object[2];
            statefilter.Items[0] = stateField;
            statefilter.Items[1] = stateList;
            // set the filter to the includedIn state filter
            sqxmlquery.query.filterings.filtering[0] = new
taleo.FindService.filtering();
            sqxmlquery.query.filterings.filtering[0].Item = statefilter;
            // query part 3: create the sortings
            sqxmlquery.query.sortings = new
taleo.FindService.querySortings();
            sqxmlquery.query.sortings.sorting = new
taleo.FindService.sorting[1];
            taleo.FindService.field contestNumberSortingField = new
taleo.FindService.field();
            contestNumberSortingField.path = "ContestNumber";
            // set the contest number sorting field to the sorting
            sqxmlquery.query.sortings.sorting[0] = new
taleo.FindService.sorting();
            sqxmlquery.query.sortings.sorting[0].Item =
contestNumberSortingField;
            // 1c - set attributes
            // attributes are used here to paginate the results in order to
            // avoid to exceed the export limits of the API
            taleo.FindService.string2stringMapEntry[] attrs = new
taleo.FindService.string2stringMapEntry[1];
            attrs[0] = new taleo.FindService.string2stringMapEntry();
            attrs[0].key = "pageindex";
            attrs[0].value = pPageIndex.ToString();
            //
*****************************************************************************
            // 2 - invoke service
            //
*****************************************************************************
            taleo.FindService.Entities pagedResults =
pFindSvc.findPartialEntities(mappingVersion, sqxmlquery, attrs);
            Console.WriteLine("Fetched page: " + pagedResults.pageIndex +
"/" + pagedResults.pageCount + " of open/posted Taleo requisitions");
            return pagedResults;
        }
    }
```

```
}
```

XML Sample Code

This section documents the XML requests and associated responses that would be exchanged between the client and Taleo applications when executing the C# or the Java Sample Code documented in the previous sections.

1. Exports all Taleo requisitions using pagination in order to retrieve the first Taleo requisition identifier from the last page of results.

```
<!-- Requisition Export Request -->
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/">
   <soapenv:Header/>
   <soapenv:Body>
      <itk:findPartialEntities xmlns:itk="http://www.taleo.com/ws/
integration/toolkit/2005/07">
         <itk:mappingVersion>http://www.taleo.com/ws/art750/2006/12</
itk:mappingVersion>
         <itk:query>
            <itk:query alias="Find Open and Posted Requisitions"
 projectedClass="Requisition">
               <itk:projections>
                  <itk:projection>
                     <itk:field path="ContestNumber"/>
                  </itk:projection>
                  <itk:projection>
                     <itk:field path="JobInformation,Title"/>
                  </itk:projection>
               </itk:projections>
               <itk:filterings>
                  <itk:filtering>
                     <itk:includedIn>
                        <itk:field path="State,Number"/>
                        <itk:list>
                           <itk:long>
                              3
                              <!--state=open-->
                           </itk:long>
                           <itk:long>
                              13
                              <!--state=posted-->
                           </itk:long>
                        </itk:list>
                     </itk:includedIn>
                  </itk:filtering>
               </itk:filterings>
               <itk:sortings>
                  <itk:sorting>
                     <itk:field path="ContestNumber"/>
                  </itk:sorting>
               </itk:sortings>
            </itk:query>
         </itk:query>
         <itk:attributes>
            <!--this specifies the server to return the first result page
 only (200 reqs max) -->
            <itk:entry>
               <itk:key>pageindex</itk:key>
               <itk:value>1</itk:value>
            </itk:entry>
         </itk:attributes>
```

```
      </itk:findPartialEntities>
   </soapenv:Body>
</soapenv:Envelope>
<!-- Requisition Export Response -->
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
   <soap:Body>
      <ns1:findPartialEntitiesResponse xmlns:ns1="http://www.taleo.com/
ws/integration/toolkit/2005/07">
         <Entities pageIndex="1" pageCount="1" pagingSize="200"
 entityCount="3" xmlns:e="http://www.taleo.com/ws/art750/2006/12"
 xmlns="http://www.taleo.com/ws/integration/toolkit/2005/07">
            <e:Entity xsi:type="e:Requisition">
               <e:ContestNumber>TSTREQ-001</e:ContestNumber>
               <e:JobInformation>
                  <e:JobInformation>
                     <e:Title>
                        <e:value locale="en">TST Requisition</e:value>
                     </e:Title>
                  </e:JobInformation>
               </e:JobInformation>
            </e:Entity>
            <e:Entity xsi:type="e:Requisition">
               <e:ContestNumber>1122</e:ContestNumber>
               <e:JobInformation>
                  <e:JobInformation>
                     <e:Title>
                        <e:value locale="en">Hourly_req</e:value>
                     </e:Title>
                  </e:JobInformation>
               </e:JobInformation>
            </e:Entity>
            <e:Entity xsi:type="e:Requisition">
               <e:ContestNumber>1234</e:ContestNumber>
               <e:JobInformation>
                  <e:JobInformation>
                     <e:Title>
                        <e:value locale="en">searchdisqualified</e:value>
                     </e:Title>
                  </e:JobInformation>
               </e:JobInformation>
            </e:Entity>
         </Entities>
      </ns1:findPartialEntitiesResponse>
   </soap:Body>
</soap:Envelope>
```

2. Using this requisition identifier imports a new candidate to the Taleo system.

```
<!-- Candidate Import Request -->
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:e="http://www.taleo.com/ws/art750/2006/12">
   <soapenv:Header>
      <itk:Debug xmlns:itk="http://www.taleo.com/ws/itk21/2005/07">
         <itk:StackTrace>true</itk:StackTrace>
      </itk:Debug>
   </soapenv:Header>
   <soapenv:Body>
      <e:create>
         <e:candidate>
            <e:Address>123 Main Street</e:Address>
```

```
            <e:Address2>Apartment 1</e:Address2>
            <e:City>Seattle</e:City>
            <e:CreationDate>2007-12-06T14:40:21.815-05:00</
e:CreationDate>
            <e:CurrentJob>
               <e:CurrentJob>
                  <e:CurrentJobJobTitle>
                     <e:value locale="en">Sr Developer</e:value>
                  </e:CurrentJobJobTitle>
                  <e:CurrentJobOtherEmployer>
                     <e:value locale="en">Super Software, Inc.</e:value>
                  </e:CurrentJobOtherEmployer>
               </e:CurrentJob>
            </e:CurrentJob>

 <e:EmailAddress>sample50851421671198@unique.email.address.com</
e:EmailAddress>
            <e:FirstName>Vic2</e:FirstName>
            <e:LastName>Smith</e:LastName>
            <e:Latitude>47.578213</e:Latitude>
            <e:Longitude>-122.333073</e:Longitude>
            <e:MobilePhone>home - 206.555.1235; work 425.325.8080</
e:MobilePhone>
            <e:Salary>100000</e:Salary>
            <e:WebSite>http://www.vicsmith.com</e:WebSite>
            <e:ZipCode>74554-234</e:ZipCode>
         </e:candidate>
      </e:create>
   </soapenv:Body>
</soapenv:Envelope>
<!-- Candidate Import Response -->
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
   <soap:Body>
      <ns1:createResponse xmlns:ns1="http://www.taleo.com/ws/
art750/2006/12">
         <ns1:Number>123456789</ns1:Number>
      </ns1:createResponse>
   </soap:Body>
</soap:Envelope>
```

3. Makes the candidate apply on the previously exported requisition (TSTREQ-001).

```
<!-- Candidate Update Request -->
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/
envelope/" xmlns:e="http://www.taleo.com/ws/art750/2006/12">
   <soapenv:Header/>
   <soapenv:Body>
      <e:update>
         <e:candidate>
            <e:EmailAddress searchType="search"
 searchValue="sample50851421671198@unique.email.address.com"/>
            <e:Applications>
               <e:PreselectionApplication>
                  <e:ApplicationState>
                     <e:ApplicationState>
                        <e:Description>
                           <!-- the application state on the TSTREQ-001
 requisition is New -->
                           <e:value searchType="search" searchValue="New"
 locale="en"/>
```

```
                        </e:Description>
                     </e:ApplicationState>
                  </e:ApplicationState>
                  <e:Requisition>
                     <e:Requisition>
                        <e:ContestNumber searchType="search"
 searchValue="TSTREQ-001"/>
                     </e:Requisition>
                  </e:Requisition>
                  <e:ApplicationDate>2005-01-01</e:ApplicationDate>
                  <e:ApplicationLocale>en</e:ApplicationLocale>
                  <e:ApplicationText>
                     <e:ApplicationText>
                        <!-- Data needs to be CDATA-escaped in XML, but
 must not be escaped in Java or .NET code -->
                        <e:AdditionalInformation><![CDATA[SUMMARY OF
 QUALIFICATIONS (...)]]></e:AdditionalInformation>
                        <e:CoverLetter><![CDATA[Hi, Thanks for passing
 along this information to me.  Let me tell you a little about myself:
 (...)]]></e:CoverLetter>
                        <e:PastedResume><![CDATA[John Smith (...)]]></
e:PastedResume>
                     </e:ApplicationText>
                  </e:ApplicationText>
                  <e:Experiences>
                     <e:Experience>
                        <e:CurrentEmployer>true</e:CurrentEmployer>
                        <!--Make sure you increment display sequences
 when adding more than one experience-->
                        <e:DisplaySequence>1</e:DisplaySequence>
                        <e:OtherEmployerName>Sr Developer</
e:OtherEmployerName>
                        <e:OtherJobTitle>Super Software, Inc.</
e:OtherJobTitle>
                     </e:Experience>
                     <e:Experience>
                        <e:CurrentEmployer>false</e:CurrentEmployer>
                        <!--Make sure you increment display sequences
 when adding more than one experience-->
                        <e:DisplaySequence>2</e:DisplaySequence>
                        <e:OtherEmployerName>Developer</
e:OtherEmployerName>
                        <e:OtherJobTitle>Super Software, Inc.</
e:OtherJobTitle>
                     </e:Experience>
                  </e:Experiences>
                  <e:Studies>
                     <e:Study>
                        <!--Make sure you increment display sequences
 when adding more than one studies-->
                        <e:DisplaySequence>1</e:DisplaySequence>
                        <e:OtherInstitutionName>Quinnipiac University</
e:OtherInstitutionName>
                        <e:OtherCourseTitle>BS CS</e:OtherCourseTitle>
                     </e:Study>
                  </e:Studies>
                  <e:InfoFeeder>
                     <e:InfoFeeder>
                        <e:Description>
                           <!--You may specify "Referrer" or "Recruiter"
 as searchValue-->
```

```
                              <value searchType="search"
 searchValue="Referrer" locale="en"/>
                           </e:Description>
                        </e:InfoFeeder>
                     </e:InfoFeeder>
                     <e:Medium>
                        <e:ApplicationMedium>
                           <e:Description>
                              <e:value searchType="search"
 searchValue="Import" locale="en"/>
                           </e:Description>
                        </e:ApplicationMedium>
                     </e:Medium>
                  </e:PreselectionApplication>
               </e:Applications>
            </e:candidate>
         </e:update>
      </soapenv:Body>
   </soapenv:Envelope>
   <!-- Candidate Update Response -->
   <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
   www.w3.org/2001/XMLSchema-instance">
      <soap:Body>
         <ns1:updateResponse xmlns:ns1="http://www.taleo.com/ws/
   art750/2006/12">
            <ns1:Number>123456789</ns1:Number>
         </ns1:updateResponse>
      </soap:Body>
   </soap:Envelope>
```