# Table of Contents

## Getting Started
- Introducing the Taleo API ................................................................. 1-1
- Integrate And Extend Taleo Solutions ............................................. 1-1
- Standard Compliance ...................................................................... 1-2
- Development Platforms ................................................................. 1-2
- API Support Policy ........................................................................... 1-2
- Taleo Web Services Namespaces ....................................................... 1-4
- Using Web Services .......................................................................... 1-6

## Quick Start
- Step 1: Have the Taleo Production Team Enable the Web Service Framework .................................................. 1-8
- Step 2: Obtain and Activate a Taleo Operator Account with Web Services Permissions ........................................ 1-8
- Step 3: Generate or Obtain the Web Services WSDL Files .......... 1-8
- Step 4: Import the WSDL Files Into Your Development Platform ................................................................. 1-9

## Standard Type Basics
- Import Standard Types .................................................................. 1-12
- Export Standard Types .................................................................... 1-25

## API Call Basics
- Characteristics of API Calls ......................................................... 1-34
- API Usage Limits and Metering ...................................................... 1-34

## Security and the API
- Basic HTTP Authentication ............................................................. 1-36
- WS-Security Authentication ............................................................ 1-36

## API Reference
- Data Model ..................................................................................... 2-1
- Selection Query Language .............................................................. 2-2

## Appendix
- Web Service Limits .......................................................................... 3-1
- Version 7.5 Namespace Limitations .................................................. 3-3
- Namespace Usage Case #1: ......................................................... 3-3
- Namespace Usage Case #2: ......................................................... 3-3
- Namespace Usage Case #3: ......................................................... 3-4
- Namespace Usage Case #4: ......................................................... 3-5
Export Query Performance Throttles ................................. 3-6
WebServices Client Sample Code ........................................ 3-7
Sample Java Code .............................................................. 3-7
C# Sample Code .................................................................. 3-11
XML Sample Code ............................................................... 3-15
Confidential Information

It shall be agreed by the recipient of the document (hereafter referred to as "the other party") that confidential information disclosed by Taleo through its documents shall be retained in confidence by the other party, and its respective employees, affiliates and/or subsidiaries, pursuant to the following terms and conditions:

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.
Getting Started
Introducing 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.

Use this section to get a quick introduction to the API:

• Integrate and Extend Taleo Solutions
• Standard Compliance
• Development Platforms
• API Support Policy
• Using Web Services

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:

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

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:

<table>
<thead>
<tr>
<th>Development Platform</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXIS2 v1.3 using XmlBeans and ADB Data Binding</td>
<td><a href="http://ws.apache.org/axis2/">http://ws.apache.org/axis2/</a></td>
</tr>
<tr>
<td>XFire 1.2.6 using XmlBeans Data Binding</td>
<td><a href="http://xfire.codehaus.org/">http://xfire.codehaus.org/</a></td>
</tr>
</tbody>
</table>

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 follow-
ing 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 are defining specific part of the document and are also used for versioning purpose.

Taleo Web Services can be defined into two categories of services: The import services and export services.

**Taleo 10 Namespaces**

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.taleo.com/ws/integration/toolkit/">http://www.taleo.com/ws/integration/toolkit/</a> [version]</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Namespace Description

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.taleo.com/ws/%5BproductNickname">http://www.taleo.com/ws/[productNickname</a>][nsVersion]/[service]</td>
<td>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 as 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: <a href="http://www.taleo.com/ws/tee800/2009/01/department">http://www.taleo.com/ws/tee800/2009/01/department</a></td>
</tr>
<tr>
<td><a href="http://www.taleo.com/ws/%5BproductNickname">http://www.taleo.com/ws/[productNickname</a>][nsVersion]/import</td>
<td>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: <a href="http://www.taleo.com/ws/tee800/2009/01/import">http://www.taleo.com/ws/tee800/2009/01/import</a></td>
</tr>
<tr>
<td><a href="http://www.taleo.com/ws/%5BproductNickname">http://www.taleo.com/ws/[productNickname</a>][nsVersion]</td>
<td>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. 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: <a href="http://www.taleo.com/ws/tee800/2009/01">http://www.taleo.com/ws/tee800/2009/01</a></td>
</tr>
</tbody>
</table>
Version 7.5 Namespaces

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.taleo.com/ws/integration/toolkit/%5Bversion">http://www.taleo.com/ws/integration/toolkit/[version</a>]</td>
<td>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.</td>
</tr>
<tr>
<td><a href="http://www.taleo.com/ws/%5BproductNickname">http://www.taleo.com/ws/[productNickname</a>][nsVersion]</td>
<td>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: <a href="http://www.taleo.com/ws/art750/2006/12">http://www.taleo.com/ws/art750/2006/12</a></td>
</tr>
</tbody>
</table>

Namespace Limitations:

The namespace strategy has evolved between the Taleo 7.50 and Taleo 10 version. The 7.50 one is missing some concept to really make each pieces of information independent of other. This will be described in greater detailed in the Taleo 7.50 Namespaces section.

For more information, refer to the Version 7.5 Namespace Limitations Appendix.

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:
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.

- 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)
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 Customer Representative to request one.

Use this section to create a sample application in your development environment:

- Step 1: Have the Taleo Production Team Enable the Web Service Framework
- Step 2: Obtain and Activate a Taleo Operator Account with Web Services Permissions
- Step 3: Generate or Obtain the Web Services WSDL Files
- Step 4: Import the WSDL File Into Your Development Platform

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
Getting Started

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=javaPackage
```

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 types.
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:

```
```

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:\WSDL-Files\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.

In this section:

- Import Standard Types
- Export Standard Types

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:
If neither create, update, and search are checked for a given entity field in the data dicationary, 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**

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

**C#**

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

**XML**

```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**

```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#**

```csharp
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**

```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.searchableMultilingualStringFieldVal-
ueSearchType.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.searchableMultilingualStringFieldVal-
ueSearchType.searchAndValue;
candidate.EmailAddress.searchTypeSpecified = true
candidate.EmailAddress.searchValue = "jsmith@acme.com";
candidate.LastName.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

```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);
```

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[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.searchableMultilingualStringValueSearchType.search;
candidate.Applications[0].Requisition.Requisition.ContestNumber.searchTypeSpecified = true;
candidate.Applications[0].Requisition.Requisition.ContestNumber.searchValue = "Req001";

XML

```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 requisi-
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**

```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 = presel.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#**

```csharp
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**

```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/>
      </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**

```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#**

```c#
taleo.candidateSvc.Candidate candidate = new taleo.candidateSvc.Candidate();
candidate.EmailAddress = new taleo.candidateSvc.searchableStringField();
candidate.EmailAddress.searchType = taleo.candidateSvc.searchableMultilingualStringFieldValue.search;
candidate.EmailAddress.searchTypeSpecified = true
candidate.EmailAddress.searchValue = "jsmith@acme.com";
candidate.Currency = new taleo.candidateSvc.CandidateCurrency();
```

**XML**

```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**

```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#**

```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();
```

**XML**

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

```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.searchableMultilingualStringFieldValue.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].searchTypeSpecified = true;
candidate.Applications[0].ApplicationState.ApplicationState.Description[0].searchValue = "New";

XML

```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

```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.addValue();
value.setSearchType(SearchableMultilingualSearchOnlyField.Value.SearchType.SEARCH);
value.setSearchValue("$"); // US Dollar
value.setLocale("en");
```

C#

```csharp
taleo.candidateSvc.Candidate candidate = new taleo.candidateSvc.Candidate();
candidate.Currency = new taleo.candidateSvc.CandidateCurrency();
candidate.Currency.Currency.Symbol.searchType = taleo.candidateSvc.searchableSearchOnlyField.SearchType.search;
```

XML

```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, UnregisteredAttendee 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**

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

// Set the search key
SearchableStringField contestNumber = req.addNewContestNumber();
contestNumber.setStringValue("INT-REQ-SRC-122354");
contestNumber.setSearchValue("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 UDF
// Create a UDSElement
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);
}
```
Getting Started

Standard Type Basics

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);

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-VC-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 UDFs
requisition.JobInformation.JobInformation.UDFs() = UDFs
requisition.JobInformation.JobInformation.UDFs(0) = New UDFs
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

XML

<merge xmlns="http://www.taleo.com/ws/art750/2006/12">
  <requisition>
    <ContestNumber searchValue="INT-REQ-SRC-122354" searchTarget="" searchType="searchAnd-Value">INT-REQ-SRC-122354</ContestNumber>
  </requisition>
</merge>
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 findPartialEntities 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 one is already documented in the
**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:

```xml
<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 unique name throughout the expression that identifies the query.

A basic query starts like this:

```xml
<query alias="BasicQuery" projectedClass="User"/>
```
Getting Started

Standard Type Basics

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.

```xml
<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.

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

SQL Equivalent: SELECT UserName AS Login FROM User

Another typical use case for projections is to apply a function to a given field. Most of the functions listed in the SQ schema respect the signatures of their SQL equivalents. In all cases, the schema indicates the number of parameters required for the function. Note that here an alias is required because of the use of a function.

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

SQL Equivalent: SELECT LOWER(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.

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

**SQL Equivalent:**

```sql
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.

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

**SQL Equivalent:**

```sql
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.

```xml
<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.

```xml
<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.

```xml
<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).

```xml
<query alias="MultipleAndFilters" 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>
```
<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>
</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 value, 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:
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**

```java
import com.taleo.export.art750.Requisition;

String contestNumber = requisition.getContestNumber();
// following conversion from String to Boolean is required to get the value as boolean
boolean hasBeenApproved = Boolean.valueOf(requisition.getHasBeenApproved()).booleanValue();
if (hasBeenApproved) {
    System.out.println("Approved Requisition: "+ contestNumber);
}
```

**C#**

```csharp
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**

```xml
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

```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#

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

XML

```xml
  <Entities pagepageIndex="1" pageCount="1" pagingSize="200" entityCount="1"
    xmlns="http://www.taleo.com/ws/art750/2006/12"
    <e:Entity xsi:type="e:Candidate">
      <e:CurrentJob>
```
<e:CurrentJob>
  <e:CurrentJobJobTitle>
    <e:value locale="en">Software Developer</e:value>
    <e:value locale="fr">Développeur logiciel</e:value>
    <e:value locale="de">Software-Entwickler</e:value>
  </e:CurrentJobJobTitle>
</e:CurrentJob>
</e:Entity>
</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

In this section:

- Characteristics of API Calls
- API Usage Limits and Metering

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. Asynchronous calls are not supported using the API but are supported using the Taleo Connect Client. For more information about Taleo Connect Client ([http://www.taleo.com/solutions/connect.php](http://www.taleo.com/solutions/connect.php)), please contact your Customer Representative.

- **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 opposed 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, for example, with invalid or missing request filterings.

- **maximum of 90 seconds (1'30") 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.

In this section:

- Basic HTTP Authentication
- WS-Security Authentication

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 recommends 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 signa-
Security and the API

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, which are:

- Unspecified
- Email Address
- X509 Subject Name
- Windows Domain Qualified Name

**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:

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

**Email Address**


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**


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**


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)**

<table>
<thead>
<tr>
<th>Alias</th>
<th><a href="http://www.taleo.com/professional">http://www.taleo.com/professional</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Base64: rmNFUq7BvVoX7cp1Z0mw0b8rcEk=</td>
</tr>
<tr>
<td>HEX</td>
<td>ae634552aec1bd5a17edca756749b0d1bf2b7049</td>
</tr>
<tr>
<td>Audience</td>
<td><a href="http://www.taleo.com/professional">http://www.taleo.com/professional</a></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Alias</th>
<th><a href="http://www.taleo.com/careersection">http://www.taleo.com/careersection</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Base64: VtphLHWfjKGcIu4zsbLY9JmKT1k</td>
</tr>
<tr>
<td>HEX</td>
<td>56da612c759f8ca19c22ee33b1b2d8f4998a4c89</td>
</tr>
<tr>
<td>Audience</td>
<td><a href="http://www.taleo.com/careersection">http://www.taleo.com/careersection</a></td>
</tr>
<tr>
<td></td>
<td>Assertion Consumer Service SAML 1.0/1.1: http(s)://hostname/servlets/CareerSection?art_servlet_language=en&amp;art_ip_ action=SAML11AssertionConsumerService</td>
</tr>
</tbody>
</table>
OnBoarding 7.0 (SAML 1.0/1.1; Liberty 1.1/1.2)

<table>
<thead>
<tr>
<th>Alias</th>
<th><a href="http://www.taleo.com/onboarding">http://www.taleo.com/onboarding</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Base64: A8y2J5r1oIMj71xR7HHMYOkr75k</td>
</tr>
<tr>
<td>HEX</td>
<td>N/A</td>
</tr>
<tr>
<td>Audience</td>
<td><a href="http://www.taleo.com/onboarding">http://www.taleo.com/onboarding</a></td>
</tr>
<tr>
<td></td>
<td>Assertion Consumer Service SAML 1.0/1.1:</td>
</tr>
<tr>
<td></td>
<td>http(s)://hostname/onboarding/</td>
</tr>
<tr>
<td></td>
<td>SamlAssertionConsumerService</td>
</tr>
</tbody>
</table>

Smartorg 7.5 (SAML 1.1)

<table>
<thead>
<tr>
<th>Alias</th>
<th><a href="http://www.taleo.com/smartorg">http://www.taleo.com/smartorg</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Base64: nVTd/8JQSEW2UPGQLYY7pcgTHb8=</td>
</tr>
<tr>
<td>HEX</td>
<td>9d54ddffc2504845b650f1902d863ba5c8131dbf</td>
</tr>
<tr>
<td>Audience</td>
<td><a href="http://www.taleo.com/smartorg">http://www.taleo.com/smartorg</a></td>
</tr>
<tr>
<td></td>
<td>Inter Site Transfer Service SAML 1.1:</td>
</tr>
<tr>
<td></td>
<td>http(s)://hostname/smartorg/</td>
</tr>
<tr>
<td></td>
<td>Saml11InterSiteTransferService.jss</td>
</tr>
</tbody>
</table>

Username Token Profile

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

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.
Appendix
# Web Service Limits

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

<table>
<thead>
<tr>
<th>Since</th>
<th>Default Value</th>
<th>Comment</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5 SP2</td>
<td>20</td>
<td>The maximum number of integration threads allowed by JVM.</td>
<td>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.</td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>-1 (no minimum)</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>200</td>
<td>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.</td>
<td>The maximum number of records is check before serializing results.</td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>2048000 (2 MB)</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>90 (1 minute 30 sec)</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>25000</td>
<td>Corresponds to an average of 17 calls/minute.</td>
<td>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.</td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>250000</td>
<td>Corresponds to an average of 10 exported entities/call for 17 calls/minute (more/call if less calls/minute)</td>
<td>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 will not run because this would exceed the daily limit.</td>
</tr>
<tr>
<td>Since</td>
<td>Default Value</td>
<td>Comment</td>
<td>Behavior</td>
</tr>
<tr>
<td>--------</td>
<td>---------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Always</td>
<td>N/A</td>
<td>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 Visual Basic language as it is not case sensitive.</td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

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

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

```xml
<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:

```xml
<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:

```xml
<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.

<table>
<thead>
<tr>
<th>Since</th>
<th>Default Value</th>
<th>Comment</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5 SP2</td>
<td>5</td>
<td>The maximum deepness of relations that can be specified in an export query (max.relation.deepness)</td>
<td></td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>-1</td>
<td>The maximum number of relations that can be specified in an export query (max.relation.count)</td>
<td></td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>1</td>
<td>The maximum number of subqueries that can be specified in an export query (max.subquery.count)</td>
<td></td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>5</td>
<td>The maximum number of fields that can be specified to filter an export query (max.filtering.field.count)</td>
<td></td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>5</td>
<td>The maximum number of fields that can be specified to group fields in an export query (max.grouping.field.count)</td>
<td>A grouping field represents a value used to group query results. In SQL, it will be pushed into the GROUP BY clause.</td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>5</td>
<td>The maximum number of fields that can be specified to join other queries in an export query (max.joining.field.count)</td>
<td>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.</td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>1</td>
<td>The maximum number of sub-queries that can be specified in an export query (max.subquery.level.count)</td>
<td></td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>100</td>
<td>The maximum number of fields that can be projected in an export query (max.projection.field.count)</td>
<td>A projection field represents a value returned by a query. In SQL, it will be pushed into the SELECT clause.</td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>5</td>
<td>The maximum number of projection fields that can be specified to filter an export query (max.projection.filtering.field.count)</td>
<td>A projection filtering field represents a filter used for projection filtering. In SQL, it will be pushed into the WHERE clause of the query that loads the selection elements.</td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>-1</td>
<td>The maximum number of query sets that can be specified in an export query (max.queryset.count)</td>
<td>A query set represents a set of queries. An operator is applied to sub-queries to build the resulting view.</td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>0</td>
<td>The maximum number of ranking fields that can be specified in an export query (max.ranking.field.count)</td>
<td>A ranking field represents a filter used for ranking. Ranking checks which entities match a criterion without excluding those that don’t.</td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>2</td>
<td>The maximum number of queries that can be specified in an export query. This number includes the parent query (max.query.count)</td>
<td></td>
</tr>
<tr>
<td>7.5 SP2</td>
<td>3</td>
<td>The maximum number of fields that can be specified to sort fields in an export query (max.sorting.field.count)</td>
<td>A sorting field represents a value used to order results. In SQL, it will be pushed into a ORDER BY clause.</td>
</tr>
</tbody>
</table>
WebServices 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 you 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 systems and has that candidate apply on that requisition.
3. Outputs the email address of the successfully imported candidate.

Notes: 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.
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.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;

/**
 * <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)
Usage: java com.taleo.demo.TaleoSamplesXmlBeans [URL] [username] [password]
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
/ **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.

Notes: 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.

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
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.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;

// 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.ContestNumber = new taleo.CandidateService.searchableStringField();
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
    // 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].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
stateField.path = "State,Number";
stateField.Items = new object[2];
stateField.Items[0] = stateField;
stateField.Items[1] = stateList;
// set the field and the list to the includedIn filter
statefilter.Items = new object[2];
statefilter.Items[0] = stateField;
statefilter.Items[1] = stateList;

// 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";

// invoke service
Console.WriteLine("Fetched page: " + pagedResults.pageIndex + "/" + pagedResults.pageCount + " of open/posted Taleo requisitions");
return pagedResults;
}
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.

```xml
<!-- Requisition Export Request -->
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
      <itk:query alias="Find Open and Posted Requisitions" projectedClass="Requisition">
        <itk:projections>
          <itk:projection path="ContestNumber"/>
          <itk:projection path="JobInformation,Title"/>
        </itk:projections>
        <itk:filterings>
          <itk:filtering>
            <itk:includedIn>
              <itk:list>
                <itk:long>3</itk:long>
                <!--state=open-->
                <itk:long>13</itk:long>
                <!--state=posted-->
              </itk:list>
            </itk:includedIn>
          </itk:filtering>
        </itk:filterings>
        <itk:sortings>
          <itk:sorting path="ContestNumber"/>
        </itk:sortings>
      </itk:query>
    </itk:findPartialEntities>
  </soapenv:Body>
</soapenv:Envelope>

<!-- Requisition Export Response -->
  <soap:Body>
      <Entities pageIndex="1" pageCount="1" pagingSize="200" entityCount="3" xsi:type="ns1:Requisition">
        <e:ContestNumber>TSTREQ-001</e:ContestNumber>
      </Entities>
    </ns1:findPartialEntitiesResponse>
  </soap:Body>
</soap:Envelope>
```
<e:JobInformation>
  <e:JobInformation>
    <e:Title>
      <e:value locale="en">TST Requisition</e:value>
    </e:Title>
  </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:StackTrace>true</itk:StackTrace>
    </itk:Debug>
  </soapenv:Header>
  <soapenv:Body>
    <e:create>
      <e:candidate>
        <e:Address>123 Main Street</e:Address>
        <e:Address2>Apartmret 1</e:Address2>
        <e:City>Seattle</e:City>
        <e:CreationDate>2007-12-06T14:40:21.815-05:00</e:CreationDate>
        <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: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:candidate>
    </e:create>
  </soapenv:Body>
</soapenv: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/">
  <soapenv:Header/>
  <soapenv:Body>
    <e:update>
      <e:candidate>
        <e:EmailAddress searchType="search" searchValue="sample508514216711198@unique.email.address.com"/>
        <e:Applications>
          <e:PreselectionApplication>
            <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:PreselectionApplication>
          <e:Requisition>
            <e:ContestNumber searchType="search" searchValue="TSTREQ-001"/>
          </e:Requisition>
          <e:ApplicationDate>2005-01-01</e:ApplicationDate>
          <e:ApplicationLocale>en</e:ApplicationLocale>
          <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:PastelResume><![CDATA[John Smith (...)]]></e:PastelResume>
          </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>Any Other</e:OtherEmployerName>
              <e:OtherJobTitle>Super Software, Inc.</e:OtherJobTitle>
            </e:Experience>
          </e:Experiences>
        </e:Applications>
      </e:candidate>
    </e:update>
  </soapenv:Body>
</soapenv:Envelope>
Appendix

WebServices Client Sample Code

rience-->
  <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>
</e:Studies>
</e:InfoFeeder>
</e:InfoFeeder>
</e:Medium>
</e:ApplicationMedium>
</e:Description>
</e:InfoFeeder>
</e:Medium>
</e:PreselectionApplication>
</e:Applications>
</e:candidate>
</e:update>
</soapenv:Body>
</soapenv:Envelope>
<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>