

Oracle

Talent Acquisition Cloud Using Web Services

17 (update 17.4)

Contents

Preface	i
<hr/>	
1 Getting Started	1
Taleo Web Services API	1
Quick Start	6
Standard Type Basics	9
API Call Basics	21
Security and the API	22
2 API Reference	23
Data Model	23
Selection Query Language	23
3 Appendix	25
Web Service Limits	25
Version 7.5 Namespace Limitations	26
Export Query Performance Throttles	28
Compatibility	29

Preface

This preface introduces information sources that can help you use the application and this guide.

Using Oracle Applications

To find guides for Oracle Applications, go to the Oracle Help Center at <http://docs.oracle.com/>.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the [Oracle Accessibility Program website](#).

Contacting Oracle

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit [My Oracle Support](#) or visit [Accessible Oracle Support](#) if you are hearing impaired.

Comments and Suggestions

Please give us feedback about Oracle Applications Help and guides! You can send an e-mail to: talent_acquisition_doc_feedback_ww_grp@oracle.com.

1 Getting Started

Taleo Web Services API

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

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

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

Integrate and Extend Taleo Solutions

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

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

- Integrate Taleo with your organization: The API enables seamless transfer between Taleo Enterprise, 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 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 Oracle Taleo Solutions, visit <http://www.oracle.com/us/products/applications/taleo/overview/index.html> or contact Oracle.

Standard Compliance

The API is implemented to comply with the following specifications:

Standard Name	Website
Simple Object Access Protocol (SOAP) 1.1	http://www.w3.org/TR/2000/NOTE-SOAP-20000508

Standard Name	Website
Web Service Description Language (WSDL) 1.1	http://www.w3.org/TR/2001/NOTE-wsdl-20010315
Hypertext Transfer Protocol (HTTP) 1.1	http://www.w3.org/Protocols/rfc2616/rfc2616.html
WS-I Basic Profile 1.1	http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html
Web Services Security 1.1	http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

There are many different styles of SOAP messages; the two most common today are `rpc/encoded` and `document/literal`. The 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:

- AXIS2 v1.3 using XmlBeans and ADB Data Binding
- XFire 1.2.6 using XmlBeans Data Binding

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

API Support Policy

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

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

Backward Compatibility

Taleo strives to make backward compatibility easy when using the 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, 10 and later).

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.

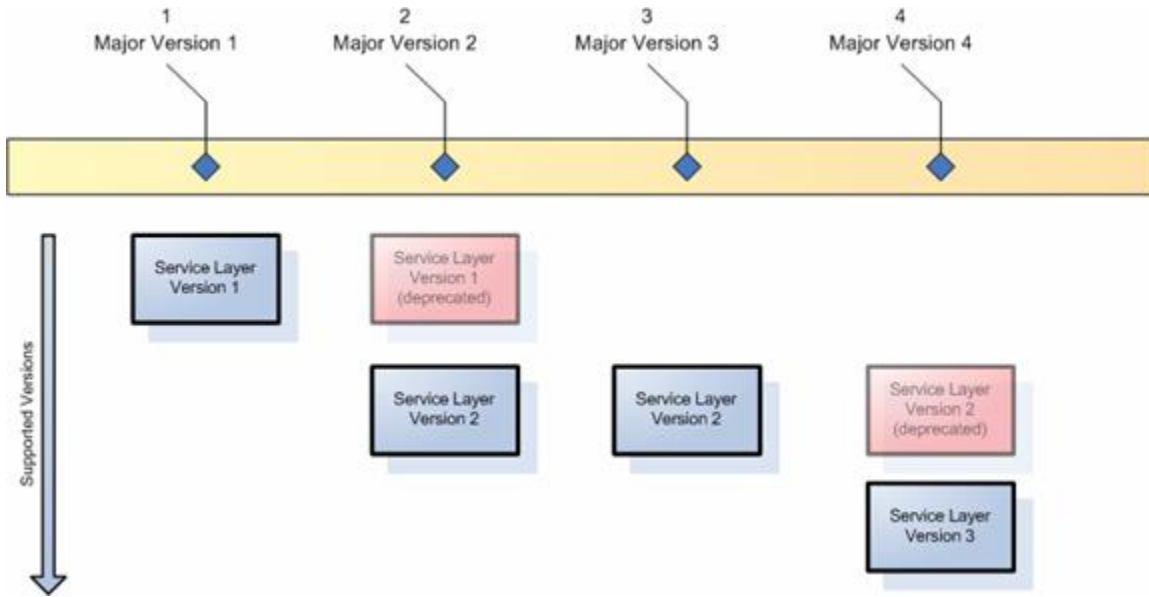
Taleo does not guarantee that an application written against one API version will work with future API versions. Changes in method signatures and data representations are often required as we continue to enhance the API. However, we strive to keep the API consistent from version to version with minimal,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

For example, an application written using the Taleo Recruiting 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 10 and later), using that same API. However, the same application may not work with later

versions of the product (i.e. Taleo Enterprise 10) without modifications to the application, using the latest available product API (i.e. Taleo Enterprise 10 and later).

API End of Life

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



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

Taleo Web Services Namespaces

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

Taleo 10 and Later Namespaces

Namespace	Description
<code>http://www.taleo.com/ws/integration/toolkit/[version]</code>	<p>The integration toolkit namespace.</p> <p>This namespace is used to define elements related to the integration toolkit framework.</p> <p>The integration toolkit framework is where the Taleo component web service infrastructure resides.</p> <p>It is used in the definition of elements such as the <code>WebServiceFault</code>.</p>

Namespace	Description
<code>http://www.taleo.com/ws/[productCode][nsVersion]/[service]</code>	<p>The service namespace.</p> <p>This namespace is used to define elements related to the web service itself.</p> <p>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.</p> <p>Its name is composed of the product code (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).</p> <p>Example:</p> <p>WebService: DepartmentService</p> <p>Namespace:</p> <p><code>http://www.taleo.com/ws/tee800/2009/01/department</code></p>
<code>http://www.taleo.com/ws/[productCode][nsVersion]/import</code>	<p>The product import data model.</p> <p>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.</p> <p>Its name is composed of the product code (i.e.: tee for Taleo Enterprise, so for SmartOrg, etc.) followed by the namespace version (i.e. 2009/01) and the import string.</p> <p>Example:</p> <p>Namespace:</p> <p><code>http://www.taleo.com/ws/tee800/2009/01/import</code></p>
<code>http://www.taleo.com/ws/[productCode][nsVersion]</code>	<p>The product export data model.</p> <p>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.</p> <p>It is used in the definition of elements such as the User, the Candidate, the Requisition and all the other Taleo entities.</p> <p>Its name is composed of the product code (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).</p> <p>Example:</p> <p>Namespace:</p> <p><code>http://www.taleo.com/ws/tee800/2009/01</code></p>

Version 7.5 Namespaces

Namespace	Description
<code>http://www.taleo.com/ws/integration/toolkit/[version]</code>	The integration toolkit namespace.

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

Namespace Limitations

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

Using Web Services

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

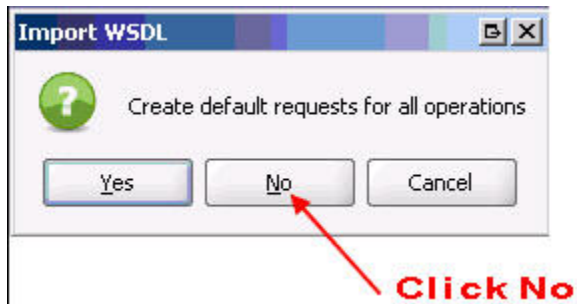
Multiple books and articles are available that describe in detail how to interact with the web services, SOAP, and WSDL standards. Some interesting starting points are:

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

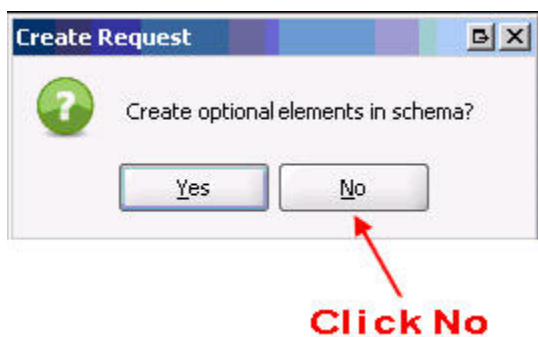
Using SOAPUI with Taleo WSDL

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

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



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



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

Quick Start

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

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

Step 1: Obtain and Activate a Taleo User Account

To access the API you need to have an activated Taleo User Account. 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 2: Obtain the Web Services WSDL Files

To access the 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>.

To obtain the WSDL File for Your Organization

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

To generate the WSDL file for your organization:

- Log in to your account using the URL specified in the data dictionary corresponding to your Taleo product.
- You should see a list of Web services available for this product. If the Web service you are looking for is not in the list, you may not have enough privileges to access it, you may be using the wrong URL for the Product, or you are searching for a deprecated Web service that has been removed or replaced by another one since the last major version.
- Right-click the Web service name to display your browser's save options, and save the WSDL to a local directory.

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


Step 3: 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 pathToWsd1/Wsd1Filename -d xmlbeans -ns2p namespaceURL=javaPackageName
```

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

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

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

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


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

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

Instructions for Microsoft Visual Studio

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

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

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

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

To access an XML Web service from managed code:

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

To add a Web reference:

1. On the Project menu, choose Add Web Reference.
2. In the URL box of the Add Web Reference dialog box, type the URL to obtain the service description of the XML Web service you want to access, such as: `file:///c:\WSDLFiles\CandidateService.wsdl` or `https://hostname/servlets/soap?ServiceName=CandidateService&wsdl`.
3. Click Go to retrieve information about the XML Web service.
4. In the Web reference name box, rename the Web reference, such as `taleo.candidatesvc`, which is the namespace you will use for this Web reference.

5. Click Add Reference to add a Web reference for the target XML Web service. For more information, see the topic "Adding and Removing Web References" in the Visual Studio documentation.
6. Visual Studio retrieves the service description and generates a proxy class to interface between your application and the XML Web service.

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

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

Standard Type Basics

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

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

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

Import Standard Types

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

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

The next sections describe each of these standard types, providing usage samples using XML (SOAP messages). The code snippets presented here are partial only, and aim to demonstrate the usage of each standard type.

Entity Fields Definition vs. Import Standard Types Mapping

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

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

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

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

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

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

String

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

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

Sample #1: Set a Candidate prefix

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

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.

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).

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):

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).

XML

```
<Candidate>
```

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

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

If we now want to set the date of entry for John Smith for the "Req001" job, we need to update the proper **Application** entity. To do so, we must find it among all his other possible applications. Since there is no identifier in the **Application** entity itself, we must use the Requisition **contestNumber** field to find the application. This is possible by specifying a **searchTarget** among the search attributes.

XML

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

SearchableSearchOnlyField

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

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

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

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).

XML

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

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

The `candidate` entity contains an `Applications` relation, whose referenced object itself contains an `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.

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

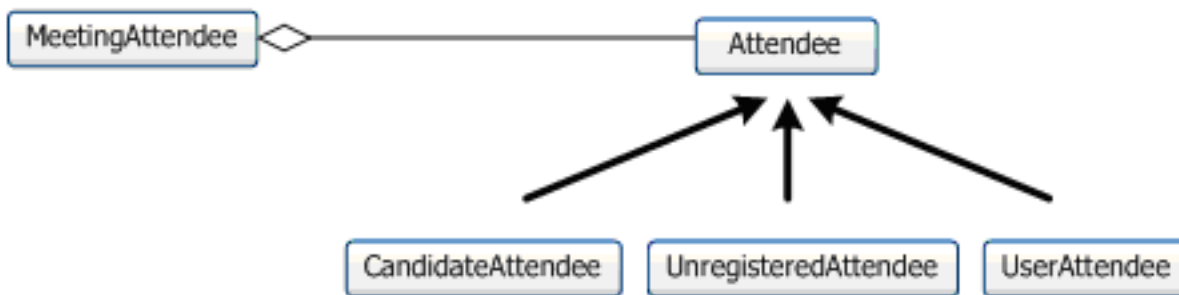
The `candidate` entity contains a `Currency` relation, whose referenced object contains a `Symbol` field that corresponds to a `SearchableMultilingualSearchOnlyField`. The following example will set the currency to be used for the candidate to the US dollar.

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:



In this example, 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

XML

```
<merge xmlns="http://www.taleo.com/ws/art750/2006/12">
  <requisition>
    <ContestNumber searchValue="INT-REQ-SRC-122354" searchTarget="." searchType="searchAndValue">INT-REQ-
SRC-122354</ContestNumber>
    <JobInformation>
    <JobInformation>
    <UDFs>
    <UDF name="TST_5f22_5f00_5f30">
    <UDSElement>
    <Description>
    <value searchValue="Cost Center 2.10" locale="en" searchTarget="." searchType="searchAndValue">Cost Center
2.10</value>
    </Description>
    </UDSElement>
    </UDF>
    </UDFs>
    </JobInformation>
    </JobInformation>
  </requisition>
</merge>
```

Export Standard Types

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

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

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

The last sections describe each of these standard types, providing usage samples using XML (SOAP messages). The code snippets presented here are partial only and aim to demonstrate the usage of each Standard Type.

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 is already documented in the Data Model of your Taleo product.

Overview

Building an export instruction is much more involved than in the case of the import feature. This section presents a simple example using the Recruiting 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 Edition 7.5 product, the instruction would look like the following:

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

```
</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

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:

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

SQL Equivalent: **FROM User**

Projections

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

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

SQL Equivalent: **SELECT UserName FROM User**

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

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

SQL Equivalent: `SELECT UserName AS Login FROM User`

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

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

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

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

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

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

Filtering elements

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

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

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

```
<field path="FirstName"/>
<string>John</string>
</equal>
</filtering>
</filterings>
</query>
```

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

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

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

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

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

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

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

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

```
<query alias="MultipleAndFilters" projectedClass="User">
```

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

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

Sorting elements

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

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

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

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

Entity Fields vs. Standard Types Mapping

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

- Export: The field can be exported.

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

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

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

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

String

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

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

The Entity `Requisition` contains the Field `ContestNumber` and `HasBeenApproved` that both correspond to a String field. This sample focuses on the result handling (we assume the export query was successfully executed by the Web service). It reads the first exported requisition contest number (requisition identifier) and whether this one was approved or not.

XML

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

MultilingualStringField

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

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

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

XML


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

API Call Basics

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

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

Characteristics of API Calls

All API calls are:

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

API Usage Limits and Metering

To protect the Taleo systems and your organization's data reliability, security, and scalability, the 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 per JVM: 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 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.

Basic HTTP Authentication

Basic HTTP authentication is mandatory to access Taleo Web services. The user who logs in to Taleo Web services will benefit from its permission set when calling services. Make sure this authentication mode is only used with a SSL (HTTP/S) connection.

Since Taleo Enterprise build 13A.5, the HTTPS protocol must be used to reach the webService "soap page" or "wsdl page". The HTTP protocol will now result in a 401 error.

For example:

- soap page: `https://[ZONENAME]/[PRODUCT]/soap`
- wsdl page: `https://[ZONENAME]/[PRODUCT]/soap?ServiceName=IntegrationManagementService&wsdl`

2 API Reference

Data Model

The 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.

Data Dictionaries can be found on: [My Oracle Support \(MOS\)](#).

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.

3 Appendix

Web Service Limits

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

Since	Default Value	Comment	Behavior
7.5 SP2	20	The maximum number of integration threads allowed by JVM.	The Web Service policy manager ensures that no more than x web service calls runs simultaneously. When the maximum number of running web service call is reached, any new web service call will be rejected with an error message.
7.5 SP2	-1 (no minimum)	The minimum export rate that is tolerated by the integration toolkit. This setting enables the integration toolkit to stop the processing of an export if the current data export rate is too low. The data export rate indicates the number of entities exported in an hour.	When serializing export results (after each entity or after each CSV report row), check the export rate. If the export rate is lower than the allowed export rate, the export is cancelled.
7.5 SP2	200	The maximum number of records that can be returned by an export request sent to the integration export service. If a request would return more results, an error is returned instead.	The maximum number of records is check before serializing results.
7.5 SP2	2048000 (2 MB)	The maximum size (in bytes) of the response generated by an export request sent to the integration export service. If the service notices that the response size is larger than this value, an error is returned instead.	When serializing export results (after each entity or after each CSV report row), check the export size. If export size is greater than the allowed size, the export is cancelled.
7.5 SP2	90 (1 minute 30 sec)	The maximum time in seconds permitted for an export request. This setting enables the integration toolkit to stop an export request if it takes more than the specified amount of time.	When serializing export results (after each entity or after each CSV report row), check the elapsed time of the export. If the elapsed time is greater than the maximum export time, the export is cancelled.
7.5 SP2	25000 Corresponds to an average of 17 calls/minute	The maximum number of web service calls that can be invoked daily.	The maximum is checked before invoking the service. If the daily invocation maximum is already reached, the service is not invoked and an error is returned to the user.
7.5 SP2	250000 Corresponds to an average of 10 exported entities/call for 17 calls/minute (more/call if less calls/minute)	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.
Always	N/A	Certain WSDLs may contain two types where the only difference in their names is that one is all in	When a customer tries to use this type of WSDL to generate a stub in Visual Basic, the generated class

Since	Default Value	Comment	Behavior
		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.	will not compile since two different classes will have the same name.

Version 7.5 Namespace Limitations

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

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

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

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

Namespace Usage Case #1

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

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

Known Limitation

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

Namespace Usage Case #2

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

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

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

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

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

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

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

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

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

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

Known Limitation

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

Namespace Usage Case #3

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

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

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

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

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

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

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

Known Limitation

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

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

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

Namespace Usage Case #4

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

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

Known Limitation

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

Export Query Performance Throttles

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

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

Since	Default Value	Comment	Behavior
		A projection field represents a value returned by a query. In SQL, it will be pushed into the SELECT clause.	

Compatibility

Taleo Connect Client compatibility with Taleo products is guaranteed one version up, when the same reference model is used. When the reference model used to create an integration point changes, compatibility is not guaranteed.

Architecture

Taleo Connect Client (TCC) communicates with Taleo Connect Server (TCS) that is embedded in every Taleo product. TCS uses internal reference models that describe each Taleo product entity available for integration. The reference models are related to the version of the business engine; the business engine contains sets of business rules that integration must follow. The reference models are not automatically compatible with each other. Taleo products are enhanced and modified from one version to the next. The business rules change to comply with the enhancements and modifications.

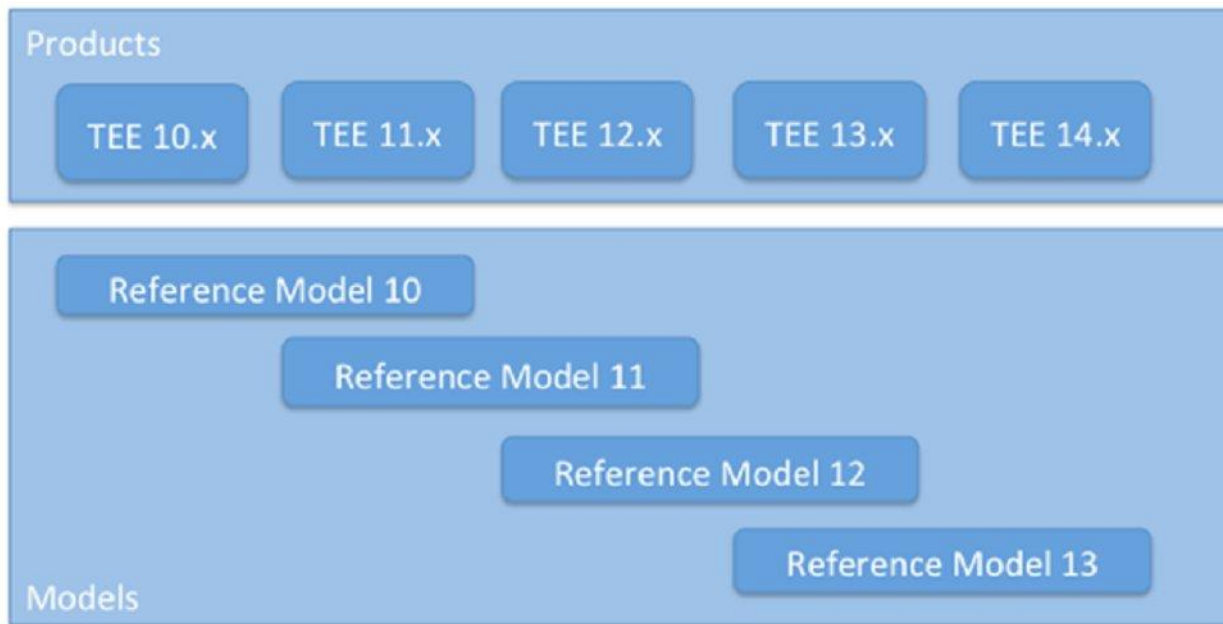
For information on the integration reference model changes for each release, refer to the following Notes.

- See: Taleo Enterprise - Taleo Integration Release Notes (MOS under Release Notes, Maintenance Packs, Express Packs, and Incidents Resolved History)

Example

An integration point built with reference model 11 X for a version 11.X product will continue to function after the product is upgraded to version 12 X, as long as the reference model remains unchanged. In this case, compatibility is guaranteed.

Product and model compatibility



If the reference model is modified, to take advantage of a new field for example, then compatibility is not guaranteed. In this case, the integration points must be re-tested and modified if necessary. A modification to a business rule may affect integration in a minimal way. Testing is required.

Note: Integrations built using the 7.5 reference model are expected to cease to be formally supported mid to late 2014, though formal communication will be made 12 months in advance of full compatibility support stopping. It is recommended in preparation to upgrade integration scripts currently on 7.5 to the latest data model version appropriate for your zone's version or at least begin the planning activities associated with this upgrade, subsequent testing and deployment.

The Recruiting/Professional and SmartOrg models have changed considerably from 7.5 upwards and it is recommended referencing the following guides for further information.

- See: Taleo Integration Migration Guide (MOS under Release Notes, Maintenance Packs, Express Packs, and Incidents Resolved History)
- See: Product Data dictionaries (MOS under Release Notes, Maintenance Packs, Express Packs, and Incidents Resolved History)