UDDI Version 2.04 API Specification
UDDI Published Specification, 19 July 2002

This version:

http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.pdf

Latest version:

http://uddi.org/pubs/ProgrammersAPI_v2.pdf

Editors (alphabetically):

Dave Ehnebuske, IBM
Barbara McKee, IBM
Dan Rogers, Microsoft

Contributors (alphabetically):

Tom Bellwood, IBM
Douglas Bryan, Accenture
Jeff Burinda, Wand
Tom Clement, Avinon
Vadim Draluk, BEA
Brian Eisenberg, Datachannel
Tom Glover, IBM
Andy Harris, i2 Technologies
Andrew Hately, IBM
Denise Ho, Ariba
Yin-Leng Husband, Compaq
Alan Karp, HP
Keisuke Kibakura, Fujitsu
Chris Kurt, Microsoft
Jeff Lancelle, Verisign
Sam Lee, Oracle
Sean MacRoibeaird, Sun
Anne Thomas Manes, Sun
Joel Munter, Intel
Tammy Nordan, Compaq
Chuck Reeves, Microsoft
Jared Rodriguez
Christine Tomlinson, Sun
Cafer Tosun, SAP
Claus von Riegen, SAP
Prasad Yendluri, WebMethods

Contents

1 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

2 Introduction

2.1 Document Overview

This document describes the programming interface and expected behaviors of all instances of the Universal Description, Discovery & Integration (UDDI) registry. The primary audience for this document is programmers who want to write software that will directly interact with a UDDI Operator Site[1]. Private implementations of the UDDI specification should provide support for the interface described here as well as the behaviors defined.

2.1.1 What is UDDI?

Universal Description, Discovery & Integration, or UDDI, is the name of a group of web-based registries that expose information about a business or other entity[2] and its technical interfaces (or API’s). These registries are run by multiple Operator Sites, and can be used by anyone who wants to make information available about one or more businesses or entities, as well as anyone that wants to find that information. There is no charge for using the basic services of these operator sites.[3]

By accessing any of the public Operator Sites, anyone can search for information about web services[4] that are made available by or on behalf of a business. The benefit of having access to this information is to provide a mechanism that allows others to discover what technical programming interfaces are provided for interacting with a business for such purposes as electronic commerce, etc. The benefit to the individual business is increased exposure in an electronic commerce enabled world.

The information that a business can register includes several kinds of simple data that help others determine the answers to the questions “who, what, where and how”. Simple information about a business – information such as name, business identifiers (D&B D-U-N-S Number®, etc.), and contact information answers the question “*Who?*” “*What?*” involves classification information that includes industry codes and product classifications, as well as descriptive information about the services that the business makes available. Answering the question “*Where?*” involves registering information about the URL or email address (or other address) through which each type of service is accessed[5]. Finally, the question “*How?”* is answered by registering references to information about interfaces and other properties of a given service. These service properties describe how a particular software package or technical interface functions. These references are called tModels in the documentation.

---

[1] Operator Site is a term used to describe an implementation of this specification that participates in the public network of UDDI sites that together operate under special contract.

[2] The term *business* is used in a general sense to refer to an operating concern or any other type of organization throughout this document. This use is not intended to preclude other organizational forms.

[3] Operator Sites are required to adhere to this specification as a minimal set of service behaviors. Operator Sites are permitted to exceed the capabilities described in this specification.

[4] *Web Service* is a term used to describe technical services that are exposed for either private or general use. Examples include purchasing services, catalog services, search services, shipping or postal services exposed over transports like HTTP or electronic mail.

[5] The information about the service point or address at which a service is exposed is sometimes referred to using the technical term *binding information*. These design specs refer to this using the term *bindingTemplate*.

2.2 Compatible registries

This specification, coupled with the API schema (uddiAPI2.xsd) and the information in the UDDI Version 2.0 Data Structure Reference, defines a programming interface that is available according to the licensing terms defined in the beginning of this document. Software developers, businesses and others are encouraged to define products and tools that make use of both public and private UDDI registries. Developers who license this specification are further encouraged to build registries that are compatible with the UDDI specifications.

2.3 What are tModels?

In order for two or more pieces of software to be compatible with each other – that is, compatible enough to be able to exchange data for the purpose of achieving a desirable result – they must share some design goals and specifications in common. The registry information model that each site supports is based on this notion of shared specifications.

In the past, to build compatible software, two companies only had to agree to use the same specification, and then test their software. However, within a UDDI registry, businesses need a way to publish information about the specifications and versions of specifications that were used to design their advertised services. To accommodate the need to distinctly identify public specifications (or even private specifications shared only with select partners), information about the specifications themselves needs to be discoverable. This information about specifications – a classic metadata construct – is called a tModel within UDDI.

The tModel mechanism serves a useful purpose in discovering information about interfaces and other technical foundation concepts that are exposed for broad use by an individual service or registration instance. To get a clearer understanding, let's consider an example.

2.3.1 An example:

Suppose your business bought a software package that let you automatically accept electronic orders via your Internet connection. Using one of the public UDDI operator sites, you could "advertise" the availability of this electronic commerce capability so that your partners and customers could find out that you can accept orders electronically.

One of the reasons you chose this particular software package was its widespread popularity. In fact the salesperson that sold you the software made a point of highlighting a feature that gives your new software its broad appeal – the use and support of a widely used electronic commerce interface that accommodates automatic business data interchange.

As you installed and configured your new software, this software automatically consulted one of the public UDDI sites and identified compatible business partners. It did this by looking up each business you identified, and located those that had already advertised support for electronic commerce services that are compatible with your own.

The configuration software accomplishes this by taking advantage of the fact that a tModel has been registered within UDDI and a corresponding tModel key (called a tModelKey) gets assigned at the time of registration. This tModel represents the interface or specification for the electronic commerce capability. Individual partner capabilities are stored within UDDI as information about service bindings[6] – and each of these bindings references the tModel that represents the specific interface that your software understands.

In general, it's pretty safe to think of the tModel keys within a services binding description as a *fingerprint* that can be used to trace the compatibility origins of a given service. Since many such services will be constructed or pre-programmed to be compatible with a given, well-known interface, references to the tModel serve to identify the properties associated with a given service binding.

[6] The term "Service Binding" refers to technical descriptions that are used by programs to identify and eventually make a call to a specific web service.

For software companies and programmers, tModels provide a common point of reference that allows a technical interface to be registered, and compatible implementations of those interfaces to be easily identified. For businesses that use software, the benefit is greatly reduced work in determining which particular bindings exposed by a business partner are compatible with the software used in-house. Finally, for standards organizations, the ability to register information about a specification and then find implementations of web services that are compatible with a standard helps customers immediately realize the benefits of a widely used design.

2.4 Classification and Identification information

One of the immediate benefits of registering business information at one of the UDDI Operator Sites is the ability to specify one or more classifications, or category codes for your business. Many such codes exist – NAICS, UN/SPC, SIC Codes, etc. – and are widely used to classify businesses, industries, and product categories. Other (and there are many) classifications designate geographic information, or membership in a given organization.

The UDDI programming interface (API) defines a consistent way for businesses to add any number of classifications to their business registrations. This information, in turn, allows simple searching to be done on the information contained in the public registries. More importantly, registering information such as industry codes, product codes, geography codes and business identification codes (such as D&B D-U-N-S Numbers®) allows other search services to use this core classification information as a starting point to provide added-value indexing and classification while still referencing your information.

The UDDI version 2 specifications add the ability to accommodate validated classification and identification taxonomies. This new capability allows any company to extend the support that all UDDI operators use to manage validated taxonomies. In UDDI version 2, two types of taxonomies are supported that were not possible in UDDI version 1. These are unchecked and checked categorization and identification taxonomies.

Unchecked taxonomies are used for categorization and identification without the need for UDDI to perform a specific call-out to a validation service. Organizations that choose to make a particular taxonomy available for categorization or identification can register a taxonomy and use that taxonomy as unchecked. Unchecked taxonomies are registered by simply registering a new tModel, and classifying that tModel as either an identifier or as a categorization taxonomy.

Checked taxonomies are used when the publisher of a taxonomy wishes to make sure that the categorization code values or identifiers registered represent accurate and validated information. UDDI version 2 supports third parties that wish to create new checked taxonomies of identifiers and categorizations.

3 Design & Architecture

The UDDI programmer's API is designed to provide a simple request/response mechanism that allows discovery of businesses, services and technical service binding information.

3.1 Design Principles

The primary principle guiding the design of this programmer's API was simplicity. Care has been taken to avoid complexity, overlap, and also to provide direct access to the appropriate levels of registered information with a minimum of programming overhead and round tripping.

3.1.1 Security

Accessing UDDI programmatically is accomplished via API calls defined in this programmer's reference. Two types of APIs are defined. A publisher's API is provided for interactions between programs and the registry for the purpose of storing or changing data in the registry. An inquiry API is provided for programs that want to access the registry to read information from the registry.

Authenticated access is required to use the publishers API. Each Operator Site is responsible for selecting and implementing an authentication protocol that is compatible with the publishers API, as well as providing a new user sign-up mechanism. Before using any of the publisher API functions, the caller is responsible for signing up with one or more Operator Sites or compatible registries and establishing user credentials.

The Inquiry and Publishers API functions are exposed as SOAP messages over HTTP. HTTPS (specifically SSL 3.0) is used for all publisher API calls in order to assure wire privacy. No authentication is required to make use of the Inquiry API functions.

3.1.2 Versioning

In any programmers API, as well as any message set, versioning issues arise as time passes. Changes to an API over time can result in requests being misunderstood or processed incorrectly unless one can determine whether the version of the API being provided matches the version of the API used by a requesting party.

In order to facilitate a proper and controlled version match, the entire API defined by this programmer's reference is version stamped. Since the API itself is based on XML messages transmitted in SOAP envelopes over HTTP[7], this version stamp takes the form of an XML attribute.

All of the messages defined in this API must be transmitted with an accompanying application version attribute. This attribute is named "*generic*[8]" and is present on all messages. Each time this specification is modified, an ensuing requirement is placed on all Operator Sites to support generic 1, the current generic and at least the previous generic, if any. Compatible registries are encouraged to support at a minimum the generic 1 version of the UDDI API.

The use of generic value 1.0 with the UDDI version 2.0 namespace, or generic value 2.0 with the UDDI version 1 namespace is not considered to be a normal use of the versioning mechanism. Individual operators are permitted to interpret mixed versioning information as an error condition.

[7] HTTP is used as a general term here. HTTPS is used exclusively for all of the calls defined in the publishers API.

[8] Versioning of application behavior is accommodated via the *generic* attribute independently from the structures defined in the accompanying schema. In general, this form of versioning is preferable because it is easier to specify a new behavior against the same structures than to try and get data structure definitions to reflect business rules. Versioning the actual schema structures would present considerable technical difficulties after more than a small number of deployed applications existed.

3.1.3 SOAP Messaging

SOAP is a method for using Extensible Markup Language (XML) in message and remote procedure call (RPC) based protocols. SOAP has been jointly defined and submitted to the World Wide Web consortium (W3C) as a note.

UDDI uses SOAP in conjunction with HTTP to provide a simple mechanism for passing XML messages to Operator Sites using a standard HTTP-POST protocol. Unless specified, all responses will be returned in the normal HTTP response document. As of version 2, there are still no interactions that deviate from this general rule.

See the appendix on SOAP-specific implementations for more information on the way that Operator Sites use the SOAP schema as an envelope mechanism for passing XML messages.

3.1.4 XML conventions

The programming interface for UDDI is based on Extensible Markup Language (XML). See the appendix (XML usage details) for more information on specific XML constructs and limitations used in the specification of the programmer's interface.

3.1.5 Error Handling

The first line of error reporting is governed by the SOAP specification. SOAP fault reporting and fault codes will be returned for most invalid requests or any request where the intent of the caller cannot be determined.

If any application level error occurs in processing a request message, a dispositionReport structure will be returned to the caller inside of a SOAP fault report. Faults that contain disposition reports contain error information that includes descriptions and typed keys that can be used to determine the cause of the error. Refer to the appendix “Error Codes” for a general understanding of error codes. API-specific interpretations of error codes are described following each API reference page.

Many of the API constructs defined in this document allow one or more of a given type of information to be passed. These API calls each conceptually represent a request on the part of the caller. The general error handling treatment recommended for UDDI operators is to detect errors in a request prior to processing the request. Any errors in the request detected will invalidate the entire request, and cause a dispositionReport to be generated within a SOAP Fault structure (see appendix A).

In the case of an API call that involves passing multiples of a given structure, the dispositionReport will call out only the first detected error, and is not responsible for reporting multiple errors or reflecting intermediate “good” data. In situations where a specific reference within a request causes an error to be generated, the corresponding disposition/fault report will contain a clear indication of the key value that caused the rejection of the rejected request.

In general, UDDI Operators may return any UDDI error code needed to describe an error. The error codes specified within each API call description are characteristic of the API call, but other UDDI error codes may be returned in unusual circumstances or when doing so adds additional descriptive information.

3.1.6 White Space

With one exception, Operator Sites and compatible implementations will store white space contained in data exactly as it is provided. The exception is that, where the UDDI schema permits the appearance of leading and/or trailing white space, it will be removed from each field, element or attribute. White space SHOULD NOT be present where the UDDI schema does not allow it. Operator Sites and compatible implementations SHOULD reject requests that contain such white space. White space characters include carriage returns, line feeds, spaces, and tabs. UDDI Operators will not allow “name” fields (where entities are named) to be empty.

3.1.7 XML Encoding

For the purpose of this specification and all Operator Sites, consistency in handling of data is essential. For this reason, the default collation order for data registered within an Operator Site is binary even though this choice is meaningless for some languages, and effectively favors alphabetic languages. Similarly, XML allows for a large number of character set encoding choices. UDDI Operators are required to only support a single XML encoding – UTF-8, and will support all compatibility characters defined for UTF-8. See appendix B for more information related to the use of byte order marks and UTF-8 and the way the UDDI SOAP implementations convert all requests to Unicode prior to processing.

4 API Reference

This API reference is divided into 3 logical sections, each addressing a particular programming focus. These sections each cover the inquiry API, the publishing API, and appendices that describe specific concepts, technical details, UDDI extensions or added background information.

The special values within API syntax examples are shown in Italics. In most cases, the following reference applies to these values:

- *uuid_key*: Access keys within all defined data elements are represented as universal unique identifiers (these are sometimes called a GUID). The name of the element or attribute designates the particular key type that is required. These keys are always formatted according to an algorithm that is agreed upon by the UDDI Operator Council with the one exception being tModelKey values, which are prefixed with a URN qualifier in the format "uuid:" followed by the UUID value.
- *generic*: This special attribute is a required metadata element for all messages. It is used to designate the specification version used to format the SOAP message. In the 1.0 version of the specification, this value is required to be “1.0". In the 2.0 version of the specification, this value is required to be “2.0”. As of the date this specification, any other value (e.g. not “1.0” and not “2.0”) passed will result in an E_unsupported error.
- *xmlns*: This special attribute is a required metadata element for all messages. Technically, it isn’t an attribute, but is formally called a namespace qualifier. It is used to designate a universal resource name (URN) value that is reserved for all references to the schema. In the 1.0 version of the specification, this value is required to be “urn:uddi-org:api". In the 2.0 version of the specification, this value is required to be “urn:uddi-org:api_v2”.
- *findQualifiers*: This special element is found in the inquiry API functions that are used to search (i.e., the messages named find_binding, find_business, find_relatedBusinesses, find_service, and find_tModel). This passed argument is used to signal special behaviors to be used with searching. See the findQualifiers appendix and the documentation for the individual find API messages for more information.
- *maxRows*: This special qualifier is found in the inquiry API functions that are used to search (e.g. find_binding, find_business, find_service, and find_tModel). This argument is used to limit the number of results returned from a request. When an Operator Site or compatible instance returns data in response to a request that contains this caller-supplied limiting argument, the number of main result elements will not exceed the integer value passed. If a result set is truncated as a result of applying this limit, or if a result set is truncated because the search would otherwise exceed an operator-specific limit, the result will include the *truncated* attribute with a value of *true*.
- *truncated*: The truncated attribute indicates that the results returned do not represent the entire query result set. The actual limit set for applying this treatment is Operator Site policy specific, but in general should be a sufficiently large number so as to not normally be an issue. No behaviors such as paging mechanisms are defined for retrieving more data after a truncated limit. The intent is to support the average query, while at the same time allowing Operator Sites the leeway required to be able to manage adequate performance. UDDI is not designed to support large data sets required by some research uses.
- *categoryBag*: Searches can be performed based on a cross section of categories. Several categories are broadly supported by all Operator Sites and provide three categorization dimensions. These are industry type, product or service type, and geography. Searches involving category information can be combined to cross multiple

dimensions[9]. For this reason, these searches are performed by default matching on ALL of the categories supplied (e.g. logical AND). In general, the embedded category information serves as voluntary hints that depend on how the registering party has categorized themselves, but not to provide a full third party categorization facility.

- *identifierBag*: Searches involving identifiers are performed matching on any supplied identifier (e.g. D&B D-U-N-S Number®, etc) for any of the primary elements that have identifierBag elements. These searches allow broad identity matching by returning a match when any keyedReference set used to search identifiers matches a registered identifier. Version 2 provides for the definition of checked identifiers. This enhancement makes it possible to distinguish copycat information within UDDI from the registrations of the authentic business registration based on validated identifiers.
- *tModelBag*: This element is found in the inquiry messages named find_business, find_service, and find_binding. Searches that match a particular technical *fingerprint* use UUID values to search for bindingTemplates with matching tModelKey value sets. When used to search for web services (e.g. the data described by a bindingTemplate structure), the concept of tModel *fingerprints* allows for highly selective searches for specific combinations of keys. For instance, the existence of a web service that implements all of the parts of the UDDI specifications can be accomplished by searching for a combination of tModel key values that correspond to the full set of specifications (the UDDI specification, for instance, is divided into at least 3 different, separately deployable tModels). At the same time, limiting the number of tModelKey values passed in a search can perform broader searches that look for any web service that implements a specific sub-part of the full specification. All tModelKey values are always expressed using a Uniform Resource Identifier (URI) format that starts with the characters "uuid:" followed by a formatted Universally Unique Identifier (UUID) consisting of Hexadecimal digits arranged in the common 8-4-4-4-12 format pattern.

In all cases, the XML structures, attributes and element names shown in the API examples are derived from the Message API schema. For a full understanding of structure contents, refer to this schema as well as the UDDI data structure reference. It is suggested that tools that understand schemas be used to generate logic that populates the structures used to make the API calls against UDDI.

4.1 About UDDI Inquiry API functions

4.1.1 Three query patterns

The Inquiry API provides three forms of query that follow broadly used conventions which match the needs of software traditionally used with registries.

4.1.1.1 The browse pattern

Software that allows people to explore and examine data – especially hierarchical data – requires browse capabilities. The browse pattern characteristically involves starting with some broad information, performing a search, finding general result sets and then selecting more specific information for drill-down.

The UDDI API specifications accommodate the browse pattern by way of the *find_xx* API calls. These calls form the search capabilities provided by the API and are matched with summary return messages that return overview information about the registered information that is associated with the inquiry message type and the search criteria specified in the inquiry.

A typical browse sequence might involve finding whether a particular business you know about has any information registered. This sequence would start with a call to *find_business*, perhaps passing the first

[9] In version (generic) 2.0, categorization has been extended to provide for limited forms of AND and OR treatment within a categoryBag, as well as providing a mechanism to support more than three conceptual dimensions. This additional behavior is afforded via new findQualifier values.

few characters of a business name that you already know. This returns a businessList result. This result is overview information (keys, names and descriptions) derived from the registered businessEntity information, matching on the name fragment that you provided.

If you spot the business you are looking for within this list, you can drill down into the corresponding businessService information, looking for particular service types (e.g. purchasing, shipping, etc) using the find_service API call. Similarly, if you know the technical *fingerprint* (tModel signature) of a particular software interface and want to see if the business you've chosen provides a web service that supports that interface, you can use the find_binding inquiry message.

4.1.1.2 The drill-down pattern

Once you have a key for one of the four main data types managed by a UDDI or compatible registry[10], you can use that key to access the full registered details for a specific data instance. The current UDDI data types are businessEntity, businessService, bindingTemplate and tModel. You can access the full registered information for any of these structures by passing a relevant key type to one of the *get_xx* API calls.

Continuing the example from the previous section on browsing, one of the data items returned by all of the *find_xx* return sets is key information. In the case of the business we were interested in, the businessKey value returned within the contents of a businessList structure can be passed as an argument to *get_businessDetail*. The successful return to this message is a *businessDetail* message containing the full registered information for the entity whose key value was passed. This will be a full *businessEntity* structure.

4.1.1.3 The invocation pattern

In order to prepare an application to take advantage of a remote web service that is registered within the UDDI registry by other businesses or entities, you need to prepare that application to use the information found in the registry for the specific service being invoked. This type of inter-business service call has traditionally been a task that is undertaken at development time. This will not necessarily change completely as a result of UDDI registry entries, but one significant problem can be managed if a particular invocation pattern is employed.

The *bindingTemplate* data obtained from the UDDI registry represents the specific details about an instance of a given interface type, including the location at which a program starts interacting with the service. The calling application or program should cache this information and use it to contact the service at the registered address whenever the calling application needs to communicate with the service instance. Tools have automated the tasks associated with caching (or hard coding) location information in previously popular remote procedure technologies. Problems arise however when a remote service is moved without any knowledge on the part of the callers. Moves occur for a variety of reasons, including server upgrades, disaster recovery, and service acquisition and business name changes.

When a call fails using cached information previously obtained from a UDDI registry, the proper behavior is to query the UDDI registry for fresh bindingTemplate information. The proper call is *get_bindingDetail* passing the original bindingKey value. If the data returned is different from the cached information, the service invocation should automatically retry the invocation using the fresh information. If the result of this retry is successful, the new information should replace the cached information.

By using this pattern with web services, a business using a UDDI Operator Site can automate the recovery of a large number of partners without undue communication and coordination costs. For example, if a business has activated a disaster recovery site, most of the calls from partners will fail when they try to invoke services at the failed site. By updating the UDDI information with the new

[10] Keys within a given UDDI compatible registry (i.e. a set of replicating UDDI nodes) such as the UDDI Business Registry are not synchronized with keys generated by a different UDDI compatible registry. There is no key portability mechanism presently defined for crossing from a replicated operator site to a compatible registry that is not part of the same replicated *Operator Cloud*. Private implementations that wish to interoperate with or store information that is also found in the public UDDI Business Registry should use the same keys that are used within that registry where appropriate.

address for the service, partners who use the invocation pattern will automatically locate the new service information and recover without further administrative action.

4.1.2 Effect of service projections on V1 find_business and find_service calls

In version 2 of UDDI the concept of "service projections"[11] which allows a businessEntity to advertise the businessService of another businessEntity as if it were its own. Because service projections are not available in UDDI Version 1, they never appear in the result set of a Version 1 find_business or find_service inquiry.

4.1.3 Elements whose length exceed the maximum lengths

The maximum length of the various UDDI data elements is documented in the UDDI V2.0 Data Structure Reference, Appendix D. These length maxima also apply to data passed in the inquiry APIs. If the length of an element passed in an inquiry API exceeds its documented maximum length, the registry will behave as if the element had been truncated at its maximum length. For example, the maximum length for a <name/> is 255 characters. If a name is passed to find_business that is longer than this, the registry will behave as if only the first 255 characters of the name had been passed.

[11] See section 4.4.13.3

4.2 Inquiry API functions

The messages in this section represent inquiries that anyone can make of any Operator Site at any time. These messages all behave synchronously and are required to be exposed via HTTP-POST only. Other synchronous or asynchronous mechanisms may be provided at the discretion of the individual UDDI Operator Site or UDDI compatible registry.

The publicly accessible queries are:

- **find_binding**: Used to locate specific bindings within a registered businessService. Returns a bindingDetail message.
- **find_business**: Used to locate information about one or more businesses. Returns a businessList message.
- **find_relatedBusinesses**: Used to locate information about businessEntity registrations that are related to a specific business entity whose key is passed in the inquiry. The Related Businesses feature is used to manage registration of business units and subsequently relate them based on organizational hierarchies or business partner relationships. Returns a relatedBusinessesList message.
- **find_service**: Used to locate specific services within a registered businessEntity. Returns a serviceList message.
- **find_tModel**: Used to locate one or more tModel information structures. Returns a tModelList structure.
- **get_bindingDetail**: Used to get full bindingTemplate information suitable for making one or more service requests. Returns a bindingDetail message.
- **get_businessDetail**: Used to get the full businessEntity information for one or more businesses or organizations. Returns a businessDetail message.
- **get_businessDetailExt**: Used to get extended businessEntity information. Returns a businessDetailExt message.
- **get_serviceDetail**: Used to get full details for a given set of registered businessService data. Returns a serviceDetail message.
- **get_tModelDetail**: Used to get full details for a given set of registered tModel data. Returns a tModelDetail message.

4.2.1 find_binding

The find_binding API call returns a bindingDetail message that contains zero or more bindingTemplate structures matching the criteria specified in the argument list.

4.2.1.1 Syntax:

```
<find_binding serviceKey="uuid_key" [maxRows="nn"] generic="2.0"
        xmlns="urn:uddi-org:api_v2" >
    [<findQualifiers/>]
    <tModelBag/>
</find_binding>
```

4.2.1.2 Arguments:

- ***serviceKey***: This *uuid_key* is used to specify a particular instance of a businessService element in the registered data. Only bindings in the specific businessService data identified by the serviceKey passed will be searched.
- ***maxRows***: This optional integer value allows the requesting program to limit the number of results returned.
- ***findQualifiers***: This optional collection of findQualifier elements can be used to alter the default behavior of search functionality. See the findQualifiers appendix for more information.
- ***tModelBag*:** This is a list of tModel *uuid_key* values that represents the technical *fingerprint* of a bindingTemplate structure contained within the businessService specified by the serviceKey value. Only bindingTemplates that contain all of the tModel keys specified will be returned (logical AND). The order of the keys in the tModel bag is not relevant.

4.2.1.3 Returns:

This API call returns a bindingDetail message upon success. In the event that no matches were located for the specified criteria, the bindingDetail structure returned will be empty (i.e., it contains no bindingTemplate data.) This signifies a zero match result. If no arguments are passed, a zero-match result set will be returned.

In the event of an overly large number of matches (as determined by each Operator Site), or if the number of matches exceeds the value of the *maxRows* attribute, the Operator site will truncate the result set.. If this occurs, the response message will contain the *truncated* attribute with the value "*true*".

4.2.1.4 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that the *uuid_key* value passed did not match with any known serviceKey or tModelKey values. The error structure will signify which condition occurred first, and the invalid key will be indicated clearly in text.
- **E_unsupported**: signifies that one of the findQualifier values passed was invalid. The invalid qualifier will be indicated clearly in text.

4.2.2 find_business

The find_business API call returns a businessList message that matches the conditions specified in the arguments.

4.2.2.1 Syntax:

```
<find_business [maxRows="nn"] generic="2.0" xmlns="urn:uddi-org:api_v2" >
    [<findQualifiers/>]
    [<name/> [<name/>]…]
    [<discoveryURLs/>]
    [<identifierBag/>]
    [<categoryBag/>]
    [<tModelBag/>]
</find_business>
```

4.2.2.2 Arguments:

- ***maxRows***: This optional integer value allows the requesting program to limit the number of results returned.
- ***findQualifiers***: This collection of findQualifier elements can be used to alter the default behavior of search functionality. See the findQualifiers appendix for more information.
- ***name*****:** This optional collection of string values represents one or more names potentially qualified with xml:lang attributes. Wildcard searching[12] can be accomplished using the % character. The businessList returned contains businessInfo structures for businesses whose name matches the value(s) passed.

  UDDI normally performs a search as though a trailing wildcard had been specified, resulting in matches on the initial portion of the search argument. (The initial portion is the left-most portion in left-to-right languages.) This behavior occurs whenever a find operation is performed that does not specify the exactNameMatch search qualifier. For that reason it is not necessary for the user to add a trailing wildcard for this type of search. The user may override this behavior by including wildcards in the find message. For example, (assuming a left-to-right language) <name>Super%docious</name> specified in a find_business would match any business name which begins with “Super” and ends with “docious”. If the user wishes this to be a left-most match, the name may be specified as <name>Super%docious%</name>. This matches any business name which begins with “Super” and contains the characters “docious” anywhere to the right of the characters “Super”. If multiple name values are passed, the match occurs on a logical OR basis.

  Each name may be marked with an xml:lang adornment. If provided, the adornment doesn't need to be unique within the message. If a language markup is specified, the search results will report a match only on those entries that match both the name value and language criteria. The match on language is a leftmost comparison of the characters supplied. This allows one to find all businesses whose name begins with an "A" and are expressed in any dialect of French, for example. No restrictions are placed on the values that can be passed in the language criteria adornment.
- ***identifierBag*****:** This is a list of business identifier references. The returned businessList contains businessInfo structures matching any of the identifiers passed (logical OR by default). When determining whether a keyedReference matches a passed keyedReference, a match occurs if and only if 1)the tModelKeys refer to the same tModel and 2) the keyValues are identical. The keyNames are not significant.

[12] Wildcard searching can be disabled by specifying the ExactNameMatch find qualifier value.

- ***categoryBag***: This is a list of category references. The returned businessList contains businessInfo elements matching all of the categories passed (logical AND by default). UDDI Version 2.0 defines special findQualifiers that affect categoryBag treatment. When determining whether a keyedReference matches a passed keyedReference, a match occurs if and only if:

  1) The tModelKeys refer to the same tModel. In deciding on this, an omitted tModelKey or an empty tModelKey (i.e., tModelKey="") is treated as though the tModelKey for uddi-org:general_keywords had been specified;
  2) The keyValues are identical; and
  3) If the tModelKey involved is that of uddi-org:general_keywords, the keyNames are identical. Otherwise keyNames are not significant. Omitted keyNames are treated as identical to empty (zero length) keyNames.

- ***tModelBag***: The registered businessEntity data contains a bindingTemplates element that in turn contains bindingTemplate elements that contain specific tModel references. The tModelBag argument lets you search for businesses that have bindings that expose a specific *fingerprint* within the tModelInstanceDetails collection. The returned businessList contains businessInfo structures that provide a summarized view of registered businessEntity data that contains bindingTemplate structures that match all of the tModel keys passed (logical AND by default)

- ***discoveryURLs***: This is a list of URLs to be matched against the discoveryURL data associated with any registered businessEntity information. To search for URL without regard to useType attribute values, pass the useType component of the discoveryURL elements as empty attributes. If useType values are included, then the match will be made only on registered information that matches both the useType and URL value. The returned businessList contains businessInfo structures matching any of the URL's passed (logical OR).

4.2.2.3 Returns:

This API call returns a businessList on success. This structure contains information about each matching business, and summaries of the businessServices, including service projections[13], exposed by the individual businesses. If a tModelBag was used in the search, the resulting serviceInfos structure will only reflect data for the businessServices that actually contained a matching bindingTemplate. In the event that no matches were located for the specified criteria, a businessList structure with zero businessInfo structures is returned. If no arguments are passed, a zero-match result set will be returned.

In the event of a large number of matches, (as determined by each Operator Site), or if the number of matches exceeds the value of the *maxRows* attribute, the Operator site will truncate the result set. If this occurs, the businessList will contain the *truncated* attribute with the value "*true*".

UDDI version 2.0 formalizes the ability for operators to support this inquiry with more than one named argument. The named arguments are all optional and, with the exception of name, may appear at most once. The name argument may appear at most five times. When more than one distinct named argument is passed, matching businesses are those which match on all of the criteria. All of the UDDI version 1.0 implementations behaved this way, but the UDDI 1.0 specification said that with the exception of findQualifiers, the remaining arguments were mutually exclusive. This resulted in an overly restrictive find_business capability.

4.2.2.4 Caveats:

If any error occurs in processing this API call, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that the *uuid_key* value passed did not match with any known tModelKey values. The error structure will signify which condition occurred first, and the invalid key will be indicated clearly in text.

[13] See section 4.4.13.3

- **E_unsupported**: signifies that one of the findQualifier values passed was invalid. The findQualifier value that was not recognized will be clearly indicated in the error text.
- **E_tooManyOptions**: signifies that the implementation defined limit on the number of name arguments was exceeded.

4.2.3 find_relatedBusinesses

The find_relatedBusinesses API call returns a relatedBusinessesList message containing results that match the conditions specified in the arguments.

4.2.3.1 Syntax:

```
<find_relatedBusinesses [maxRows="nn"] generic="2.0" xmlns="urn:uddi-org:api_v2" >
    [<findQualifiers/>]
    <businessKey/>
    [<keyedReference/>]
</find_relatedBusinesses>
```

4.2.3.2 Arguments:

- ***maxRows***: This optional integer value allows the requesting program to limit the number of results returned.
- ***findQualifiers***: This collection of findQualifier elements can be used to alter the default behavior of search functionality. See the findQualifiers appendix for more information.
- ***businessKey***: This *uuid_key* is used to specify a particular businessEntity instance to use as the focal point of the search. This argument is required and must be used to specify an existing businessEntity in the registry. The result set will report businesses that are related in some way to the businessEntity whose key is specified.
- ***keyedReference***: This is a single, optional keyedReference element that is used to specify that only businesses that are related to the focal point in a specific way should be included in the results. Specifying a keyedReference only effects whether a business is selected for inclusion in the results. If a business is selected, the results will include the full set of shared relationships between it and the focal point. See the uddi-org:relationships canonical tModel for more information on specifying relationships. The keyedReference passed matches one specified in a relationship if and only if 1) the tModelKey refers to the same tModel; 2) the keyNames are identical; and 3) the keyValues are identical. Omitted keyNames are treated as identical to empty (zero length) keyNames.

4.2.3.3 Returns:

This API call returns a relatedBusinessesList on success. In the event that no matches were located for the specified criteria, the relatedBusinessesList message returned will contain an empty relatedBusinessInfos element. This signifies zero matches. If no arguments are passed, a zero-match result set will be returned.

In the even of a large number of matches (as determined by each Operator Site), or if the number of matches exceeds the value of the *maxRows* attribute, the Operator site will truncate the result set. If this occurs, the relatedBusinessesList will contain the *truncated* attribute with the value of this attribute set to *true*.

4.2.3.4 Caveats:

If any error occurs in processing this API call, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that a *uuid_key* or tModel key value passed did not match with any known businessKey key or tModel key values. The error structure will signify which condition occurred and the key that caused the question will be clearly indicated in the error text.
- **E_unsupported**: signifies that one of the findQualifier values passed was invalid. The invalid qualifier will be clearly indicated in the error text.

4.2.4 find_service

The find_service API call returns a serviceList message that matches the conditions specified in the arguments.

4.2.4.1 Syntax:

```
<find_service [businessKey="uuid_key"] " [maxRows="nn"] generic="2.0"
      xmlns="urn:uddi-org:api_v2" >
    [<findQualifiers/>]
    [<name/> [<name/>]…]
    [<categoryBag/>]
    [<tModelBag/>]
</find_service>
```

4.2.4.2 Arguments:

- ***businessKey***: This optional *uuid_key* is used to specify a particular businessEntity instance to search. This argument may be used to specify an existing businessEntity in the registry or it may be omitted or specified as businessKey="" to indicate that all businessEntities are to be searched.
- ***maxRows***: This optional integer value allows the requesting program to limit the number of results returned.
- ***findQualifiers***: This optional collection of findQualifier elements can be used to alter the default behavior of search functionality. See the findQualifiers appendix for more information.
- ***name*****:** This optional collection of string values represents one or more names potentially qualified with xml:lang attributes. Those businessService elements having names that match the name argument(s) are returned. The argument(s) may contain wildcards if desired. Matching and wildcard behavior is as described for find_business.

  Each name may be marked with an xml:lang adornment. If provided, the adornment doesn't need to be unique within the message. If a language markup is specified, the search results will report a match only on those entries that match both the name value and language criteria. The match on language is a leftmost comparison of the characters supplied. This allows one to find all businesses whose name begins with an "A" and are expressed in any dialect of French, for example. No restrictions are placed on the values that can be passed in the language criteria adornment.
- ***categoryBag***: This is a list of category references. The returned serviceList contains serviceInfo structures matching all of the categories passed (logical AND by default). UDDI Version 2.0 defines special findQualifiers that affect categoryBag treatment. When determining whether a keyedReference matches a passed keyedReference, a match occurs if and only if all of the following are true:

  1) The tModelKeys refer to the same tModel. In deciding on this, an omitted tModelKey or an empty tModelKey (i.e., tModelKey="") is treated as though the tModelKey for uddi-org:general_keywords had been specified;
  2) The keyValues are identical; and
  3) If the tModelKey involved is that of uddi-org:general_keywords, the keyNames are identical. Otherwise keyNames are not significant. Omitted keyNames are treated as identical to empty (zero length) keyNames.

- ***tModelBag*****:** This is a list of tModel *uuid_key* values that represent the technical *fingerprint* of a bindingTemplate structure to find. Version 2.0 defines a way to associate businessService structures with more than one businessEntity. All bindingTemplate structures within any businessService associated with the businessEntity specified by the businessKey argument will be searched. If more than one tModel key is specified in this structure, only

businessService structures that contain bindingTemplate structures with *fingerprint* information that matches all of the tModel keys specified will be returned (logical AND only).

4.2.4.3 Returns:

This API call returns a serviceList on success. In the event that no matches were located for the specified criteria, the serviceList message returned will contain an empty businessServices element. This signifies zero matches. If no search arguments (including businessKey) are passed, a zero-match result set will be returned. When a businessKey is supplied, the resulting serviceList will only contain services that are offered by the designated business. When the businessKey argument is absent or the key was specified "", all services that meet the other criteria are returned in the serviceList, without regard to the business offering them. The named arguments are all optional and, with the exception of name, may appear at most once. When more than one distinct named argument is passed, matching services are those which match on all of the criteria.

When the businessKey attribute is used to specify a particular businessEntity to be searched, the find_service API treats service projections[14] in the same way it treats other businessServices. If they match the specified search criteria, they are returned in the result set. If businessKey is omitted or specified as empty (i.e., businessKey="") find_service omits service projections from the result set.

In the event of a large number of matches (as determined by each Operator Site), or if the number of matches exceeds the value of the *maxRows* attribute, the Operator site will truncate the result set. If this occurs, the serviceList will contain the *truncated* attribute with the value of this attribute set to *true*.

4.2.4.4 Caveats:

If any error occurs in processing this API call, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that a *uuid_key* value passed did not match any known businessKey key or tModel key values. The error structure will signify which condition occurred first and the key that caused the question will be clearly indicated in the error text.
- **E_tooManyOptions**: signifies that the implementation defined limit on the number of name arguments was exceeded.
- **E_unsupported**: signifies that one of the findQualifier values passed was invalid. The findQualifier value that was not recognized will be clearly indicated in the error text.
    - o

[14] See section 4.4.13.3

4.2.5 find_tModel

This find_tModel API call is for locating a list of tModel entries that match a set of specific criteria. The response will be a list of abbreviated information about registered tModel data that matches the criteria specified. This result will be returned in a tModelList message.

4.2.5.1 Syntax:

```
<find_tModel [maxRows="nn"] generic="2.0" xmlns="urn:uddi-org:api_v2" >
    [<findQualifiers/>]
    [<name/>]
    [<identifierBag/>]
    [<categoryBag/>]
</find_tModel>
```

4.2.5.2 Arguments:

- ***maxRows***: This optional integer value allows the requesting program to limit the number of results returned.
- ***findQualifiers***: This collection of findQualifier elements can be used to alter the default behavior of search functionality. See the findQualifiers appendix for more information.
- ***name*****:** This string value represents a name. Since tModel data only has a single name, only a single name may be passed. The returned tModelList contains tModelInfo elements for tModels whose name matches the value passed. Matching and wildcard behavior is as described for find_business.
- ***IdentifierBag*****:** This is a list of business identifier references. The returned tModelList contains tModelInfo elements matching any of the identifiers passed (logical OR by default). findQualifiers can be used to alter this logical OR behavior. When determining whether a keyedReference matches a passed keyedReference, a match occurs if and only if 1) the tModelKeys refer to the same tModel and 2) the keyValues are identical. The keyNames are not significant.
- ***categoryBag***: This is a list of category references. The returned tModelList contains tModelInfo elements matching all of the categories passed (logical AND by default). findQualifiers can be used to alter this logical AND behavior. When determining whether a keyedReference matches a passed keyedReference, a match occurs if and only if all of the following are true:
    1) The tModelKeys refer to the same tModel. In deciding on this, an omitted tModelKey or an empty tModelKey (i.e., tModelKey="") is treated as though the tModelKey for uddi-org:general_keywords had been specified;
    2) the keyValues are identical; and
    3) If the tModelKey involved is that of uddi-org:general_keywords, the keyNames are identical. Otherwise keyNames are not significant. Omitted keyNames are treated as identical to empty (zero length) keyNames.

4.2.5.3 Returns:

This API call returns a tModelList message on success. In the event that no matches were located for the specified criteria, an empty tModelInfos element will be returned (e.g. containing zero tModelInfo elements). This signifies zero matches.

In the event of a large number of matches (as determined by each Operator Site), or if the number of matches exceeds the value of the *maxRows* attribute, the Operator site will truncate the result set. If this occurs, the tModelList will contain the *truncated* attribute with the value *true*.

4.2.5.4 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

- **E_unsupported**: signifies that one of the findQualifier values passed was invalid. The invalid qualifier will be clearly indicated in the error text.

4.2.6 get_bindingDetail

The get_bindingDetail API call is for requesting the run-time bindingTemplate information for the purpose of invoking a registered business API.

4.2.6.1 Syntax:

```
<get_bindingDetail generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <bindingKey/> [<bindingKey/> …]
</get_bindingDetail>
```

4.2.6.2 Arguments:

- ***bindingKey***: one or more *uuid_key* values that represent the UDDI assigned keys for specific instances of registered bindingTemplate data.

4.2.6.3 Behavior:

UDDI recommends that bindingTemplate information be cached locally by applications so that repeated calls to a service described by a bindingTemplate can be made without having to make repeated round trips to an UDDI registry. In the event that a call made with cached data fails, the get_bindingDetail message can be used to get fresh bindingTemplate data. This is useful in cases such as a service you are using relocating to another server or being restored in a disaster recovery site.

4.2.6.4 Returns:

This API call returns a bindingDetail message on successful match of one or more bindingKey values. If multiple bindingKey values were passed, the results will be returned in the same order as the keys passed.

If a large number of keys are passed, an Operator Site may truncate the result set. If this occurs, the bindingDetail result will contain the *truncated* attribute with the value *true*.

4.2.6.5 Caveats:

If any error occurs in processing this API call, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known bindingKey key values. No partial results will be returned – if any bindingKey values passed are not valid bindingKey values, this error will be returned. The key value that caused the error condition will be clearly indicated in the error text.

4.2.7 get_businessDetail

The get_businessDetail API call returns complete businessEntity information for one or more specified businessEntity registrations matching on the businessKey values specified.

4.2.7.1 Syntax:

```
<get_businessDetail generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <businessKey/> [<businessKey/> …]
</get_businessDetail>
```

4.2.7.2 Arguments:

- ***businessKey***: one or more *uuid_key* values that represent specific instances of known businessEntity data.

4.2.7.3 Returns:

This API call returns a businessDetail message on successful match of one or more businessKey values. If multiple businessKey values were passed, the results will be returned in the same order as the keys passed.

If a large number of keys are passed, an *Operator Site* may truncate the result set. If this occurs, the businessDetail response message will contain the *truncated* attribute with the value *true*.

businessEntity elements retrieved with get_businessDetail can contain service projections (see section 4.4.13.3). Such projected services appear in full in the businessEntity in which they occur. Projected services can be distinguished from the services that are naturally contained in the businessEntity in which they appear by their businessKey. Services naturally contained in the businessEntity have the businessKey of the businessEntity in which they appear. Projected services have the businessKey of the businessEntity of which they are a natural part.

4.2.7.4 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known businessKey values. No partial results will be returned – if any businessKey values passed are not valid, this error will be returned. The key value that caused the error will be clearly indicated in the error text.

4.2.8 get_businessDetailExt

The get_businessDetailExt API call returns extended businessEntity information for one or more specified businessEntity registrations. This message returns exactly the same information as the get_businessDetail message, but may contain additional attributes if the source is an external registry (e.g. not an Operator Site) that is compatible with this API specification.

4.2.8.1 Syntax:

```
<get_businessDetailExt generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <businessKey/> [<businessKey/> …]
</get_businessDetailExt>
```

4.2.8.2 Arguments:

- ***businessKey***: one or more *uuid_key* values that represent UDDI assigned businessKey values for specific instances of known businessEntity data.

4.2.8.3 Returns:

This API call returns a businessDetailExt message on successful match of one or more businessKey values. If multiple businessKey values were passed, the results will be returned in the same order as the keys passed.

If a large number of keys are passed, an Operator Site may truncate the result set. If this occurs, the businessDetailExt response message will contain the *truncated* attribute with the value *true*.

4.2.8.4 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known businessKey values. No partial results will be returned – if any businessKey values passed are not valid, this error will be returned. The key value that caused the error condition will be clearly indicated in the error text.
- **E_unsupported**: signifies that the UDDI implementation queried does not support the extended detail function. If this occurs, businessDetail information should be queried via the get_businessDetail API. Operator Sites will not return this code, but will instead return a businessDetailExt result with full businessDetail information embedded.

4.2.9 get_serviceDetail

The get_serviceDetail API call is used to request full information about a known businessService structure.

4.2.9.1 Syntax:

```
<get_serviceDetail generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <serviceKey/> [<serviceKey/> …]
</get_serviceDetail>
```

4.2.9.2 Arguments:

- ***serviceKey***: one or more *uuid_key* values that represent UDDI assigned serviceKey values of specific instances of known businessService data.

4.2.9.3 Returns:

This API call returns a serviceDetail message on successful match of one or more serviceKey values. If multiple serviceKey values were passed, the results will be returned in the same order as the keys passed.

If a large number of keys are passed, an Operator Site may truncate the result set. If this occurs, the response will contain a *truncated* attribute with the value *true*.

4.2.9.4 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known serviceKey values. No partial results will be returned – if any serviceKey values passed are not valid, this error will be returned. The key value that caused the error condition will be clearly indicated in the error text.

4.2.10 get_tModelDetail

The get_tModelDetail API call is used to request full information about /known tModel data by key.

4.2.10.1 Syntax:

```
<get_tModelDetail generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <tModelKey/>
    [<tModelKey/> …]
</get_tModelDetail>
```

4.2.10.2 Arguments:

- ***tModelKey***: one or more URN qualified *uuid_key* values that represent UDDI assigned tModelKey values of specific instances of known tModel data. All tModelKey values begin with a uuid URN qualifier (e.g. "uuid:" followed by a known tModel key.)

4.2.10.3 Returns:

This API call returns a tModelDetail message on successful match of one or more tModelKey values. If multiple tModelKey values were passed, the results will be returned in the same order as the keys passed.

If a large number of keys are passed, an Operator Site may truncate the result set. If this occurs, the response will contain a *truncated* attribute with the value of *true*.

4.2.10.4 Caveats:

If any error occurs in processing this API call, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the URN qualified *uuid_key* values passed did not match with any known tModelKey values. No partial results will be returned – if any tModelKey values passed are not valid, this error will be returned. Any tModelKey values passed without a “uuid:” URN qualifier will be considered invalid. The key value that caused the error condition will be clearly indicated in the error text.

4.3 About UDDI Publishing API functions

The messages in this section represent commands that require authenticated[15] access to an Operator Site, and are used to publish and update information contained in a UDDI compatible registry. Each business should initially select one Operator Site to host their information. Once chosen, information can only be updated at the site originally selected. UDDI provides no automated means to reconcile multiple or duplicate registrations.

The messages defined in this section all behave synchronously and are callable via HTTP-POST only. HTTPS is used exclusively for all of the calls defined in this publisher's API.

4.3.1 Rationale for UDDI version 2.0 Publishing API Enhancements

Many larger corporations and virtual businesses such as marketplaces and trade blocks had commented that UDDI Version 1 was not adequate for registration of complex business information. Additionally, feedback gathered from the Version 1 implementations made it clear that duplicating service and binding information for individual business units was effective, but not optimal. Others provided the view that it was essential to be able to distinguish a business registration made by the legitimate business from one merely named in a way so as to confuse the reader.

4.3.2 Features to help the registry become more useful

These requirements, as well as the need to fine-tune the UDDI design in this second release of the specifications have resulted in new publishing features. Five new publishers API messages are defined in UDDI version 2.0 for managing the ability to describe jointly managed business relationships between individual UDDI businessEntity registrations. Additionally, the existing save_business message has a new behavior that allows businessService references, called “service projections”, to be shared by more than one registered business, while maintaining the one-account content control for services affected in this way.

One new message is defined in the inquiry API for version 2.0 in order to make the relationship data that is confirmed visible to the public. Relationships that are not mutually confirmed – that is, not proven to be made by the parties controlling both businesses on either side of a relationship assertions – are not visible to the public, ever.

A jointly managed relationship occurs when the person associated with a publisher account that controls a specific set of business registrations expresses a relationship assertion that coincides with assertions made by the person who manages another business registration. UDDI provides some assurances to the general reader that the relationships represented in UDDI are mutually agreed upon relationships. This approach prevents parties from claiming a relationship that cannot be confirmed.

4.3.3 Publisher API summary

The publishing API calls defined that UDDI operators support are:

- **add_publisherAssertions**: Used to add relationship assertions to the existing set of assertions. See the appendix J describing relationship and publisher assertions.
- **delete_binding**: Used to remove an existing bindingTemplate from the bindingTemplates collection that is part of a specified businessService structure.
- **delete_business**: Used to delete registered businessEntity information from the registry.
- **delete_publisherAssertions**: Used to delete specific publisher assertions from the assertion collection controlled by a particular publisher account. Deleting assertions from the assertion collection will affect the visibility of business relationships. Deleting an assertion will cause any relationships based on that assertion to be invalidated.

[15] Authentication is not regulated by this API specification. Individual Operator Sites will designate their own procedures for getting a userID and password credentials

- **delete_service**: Used to delete an existing businessService from the businessServices collection that is part of a specified businessEntity.
- **delete_tModel**: Used to hide registered information about a tModel. Any tModel hidden in this way is still usable for reference purposes and accessible via the get_tModelDetail message, but is simply hidden from find_tModel result sets. There is no way to actually cause a tModel to be deleted, except by administrative petition.
- **discard_authToken**: Used to inform an Operator Site that a previously provided authentication token is no longer valid and should be considered invalid if used after this message is received and until such time as an authToken value is recycled or reactivated at an operator's discretion. See get_authToken.
- **get_assertionStatusReport**: Used to get a status report containing publisher assertions and status information. This report is useful to help an administrator manage active and tentative publisher assertions. Publisher assertions are used in UDDI to manage publicly visible relationships between businessEntity structures. Relationships are a feature introduced in generic 2.0 that help manage complex business structures that require more than one businessEntity or more than one publisher account to manage parts of a businessEntity. Returns an assertionStatusReport that includes the status of all assertions made involving any businessEntity controlled by the requesting publisher account.
- **get_authToken**: Used to request an authentication token from an Operator Site. Authentication tokens are required when using all other API's defined in the publishers API. This function serves as the program's equivalent of a login request.
- **get_publisherAssertions**: Used to get a list of active publisher assertions that are controlled by an individual publisher account. Returns a publisherAssertions message containing all publisher assertions associated with a specific publisher account. Publisher assertions are used to control publicly visible business relationships.
- **get_registeredInfo**: Used to request an abbreviated synopsis of all information currently managed by a given individual.
- **save_binding**: Used to register new bindingTemplate information or update existing bindingTemplate information. Use this to control information about technical capabilities exposed by a registered business.
- **save_business**: Used to register new businessEntity information or update existing businessEntity information. Use this to control the overall information about the entire business. Of the save_xx API's this one has the broadest effect. In UDDI V2, a feature is introduced where save_business can be used to reference a businessService that is parented by another businessEntity.
- **save_service**: Used to register or update complete information about a businessService exposed by a specified businessEntity.
- **save_tModel**: Used to register or update complete information about a tModel.
- **set_publisherAssertions**: (UDDI V2 and later) – used to save the complete set of publisher assertions for an individual publisher account. Replaces any existing assertions, and causes any old assertions that are not reasserted to be removed from the registry. Publisher assertions are used to control publicly visible business relationships.

4.3.4 Effect of Version 1 save_xx in Version 2 UDDI Registries

UDDI Version 2 registries must accept UDDI Version 1 messages. (See Versioning, above.) UDDI Version 2 introduces a number of changes, for example multiple name elements on businessEntity, that are not expressible in UDDI Version 1. As is true with all save_xx operations, a Version 1 save_xx operation completely replaces the entity being saved. If the entity being replaced previously contained data not expressible in Version 1, it will no longer contain such data after a successful Version 1 save_xx operation.

4.3.5 Saving categorization and identification information

Several of the API's defined in this section allow you to save categorization and identification information that is used to support searches that use taxonomy references. These are currently the save_business, save_service and save_tModel APIs. Categorization is specified using the element named categoryBag, which contains keyedReference elements, each of which is a namespace-qualified reference to a taxonomy key and description. Identification is specified analogously using the IdentifierBag element to contain keyedReferences that refer to identifier systems.

4.3.5.1 Specifying keyedReferences

KeyedReference elements have three attributes, tModelKey, keyValue, and keyName.

The tModelKey is a reference to a taxonomy tModel (in the case of categorization) or to an identifier system tModel (in the case of identification). The reference specifies which taxonomy or identifier system the keyedReference uses. Normally, the tModelKey MUST contain a valid tModelKey for the keyedReference to be valid. However, in the case of a keyedReference that is used in a categoryBag, the tModelKey may be omitted or specified as a zero-length string to indicate that the taxonomy being used is uddi-org:general_keywords. When it is omitted in this manner, the UDDI registry will insert the proper key during the save_xx operation.

When the keyedReference uses a taxonomy tModel, the keyValue specifies which category in the taxonomy is asserted by the keyedReference. When the tModelKey uses an identifier tModel, the keyValue specifies which identifier in the identifier system is asserted by the keyedReference. The keyValue is always required.

Normally, the keyName is an optional description of the keyValue. This is useful as documentation because many taxonomies and identifier systems use numerical values for their categories and identifiers that are difficult to remember. There is a special case, though. When the keyedReference asserts a categorization in the uddi-org:general_keywords taxonomy, the keyName specifies the name part of a name-value pair and the keyValue specifies the value part. For this reason, when the keyedReference refers to the uddi-org:general_keywords tModel, the keyName MUST be specified.

4.3.5.2 Special considerations for validated namespaces

UDDI Version 2.0 introduces the notion of external checked namespaces. The facilities provided allow third party *providers* to extend UDDI operator behavior to allow new categorization and identification schemes to be incorporated into the data in a UDDI registry. This new functionality provides a way to do innovative things with the data in a UDDI registry by making it possible to constrain the classifications and identifiers that can be attributed to one or more businesses.

This Version 2.0 feature makes it possible for a third party, who perhaps specializes in identity verification or provides a specialized classification that is restricted in use to either specific value sets or as applicable to specific members of a group or affiliation with an organization. The result is that UDDI data becomes "trustable" at such time as these trusted third parities use these new features to enhance the data found within UDDI registrations.

For built-in taxonomies (e.g. NAICS, UN/SPSC, and geography), data contained in the keyValue attribute of each keyedReference is validated against the taxonomy referenced by the associated tModelKey. Only valid keyValue data will be stored as entered unless the taxonomy specified by the tModelKey reference represents an unchecked (e.g. non-validated) namespace. The keyName attribute that accompanies each reference is optional and solely used for descriptive purposes on the part of the party that saves the categorization information. A good practice is to store the description of the code set as defined by the categorization taxonomy.

For externally validated taxonomies introduced as a result of Version 2.0 changes, there is no set behavior for determining what specific information is validated relative to the use of a checked namespace. The reason for this is that the external validation service is free to check any set of conditions that are deemed appropriate relative to the use of a given taxonomy or identifier scheme. The only behavior that is predictable, is that the UDDI Operator will pass the information being registered (as a result of a save_xx message) to the service that performs validation on a checked namespace. If that service returns no error, the save will be permitted. Otherwise the error information

returned by the validation service will be returned to the caller unchanged and the save operation will fail.

Operator Sites must reject any save request that contains (in either the identifierBag or categoryBag) a reference to a “checked” namespace where the validation service is either unavailable (due to outage or other conditions) or returns an error indication during the validation step. Unchecked namespaces do not require any validation, although the namespaces must still be registered as tModels.

4.3.6 Special considerations for the xml:lang attribute

The name and description UDDI elements may be adorned with the xml:lang attribute to indicate the language in which their content is expressed. (See the UDDI V2.0 Data Structure Reference.) In a list of names or descriptions passed in a save_xx call, the xml:lang attribute MAY be omitted. When so omitted, the UDDI registry MAY insert an xml:lang attribute into the element before it is saved. If an xml:lang attribute is inserted, the value of the inserted attribute will be the code for the default language of the UDDI node operator.

When a list of name elements or description elements is passed in a save_xx call, the xml:lang attributes in the list, including any attribute inserted per the above, SHOULD be unique. UDDI operator nodes MAY fail save_xx calls that do not meet this requirement by returning an E_languageError error.

Name elements in UDDI core data structures are frequently the main targets for sorts during UDDI inquiries. When a UDDI data structure has multiple names, sorting occurs on the first name. Care should be taken to list the primary name first when the entity is saved to ensure the proper placement of the entity in a sorted result set.

4.3.7 Third party opportunities

Many opportunities exist for third parties who wish to provide value-added services as adjuncts to the core behaviors of UDDI Operator Sites. These opportunities include such options as becoming an external taxonomy or namespace validation authority, or providing richer search facilities that go beyond what the base UDDI API provides. In all cases, third parties are encouraged to work directly with individual UDDI Operators in order to set up the appropriate contractual business relationships.

4.4 Publishing API Function Reference

4.4.1 add_publisherAssertions

The add_publisherAssertions API call causes one or more publisherAssertions to be added to an individual publisher's *assertion collection*. See the appendix describing relationships and the message named get_publisherAssertions for more information on this collection.

4.4.1.1 Syntax:

```
<add_publisherAssertions generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
    <publisherAssertion>
        <fromKey/>
        <toKey/>
        <keyedReference/>
    </publisherAssertion>
    [<publisherAssertion/> …]
</add_publisherAssertions>
```

4.4.1.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- ***publisherAssertion***: one or more relationship assertions. Relationship assertions consist of a reference to two businessEntity key values as designated by the fromKey and toKey elements, as well as a required expression of directional relationship within the contained keyedReference element[16]. See the appendix on managing relationships. The fromKey, the toKey, and all three parts of the keyedReference – the tModelKey, the keyName, and the keyValue must be specified. Empty (zero length) keyNames and keyValues are permitted.

4.4.1.3 Returns:

Upon successful completion, a dispositionReport message is returned with a single success indicator.

4.4.1.4 Caveats:

If any error occurs in processing this API call, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known businessKey or tModelKey values. The key and element or attribute that caused the problem will be clearly indicated in the error text.
- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_userMismatch**: signifies that neither of the businessKey values passed in the embedded fromKey and toKey elements is controlled by the publisher account associated with the authentication token. The error text will clearly indicate which assertion caused the error.

[16] Note: The keyName, keyValue and tModelKey attributes associated with a keyedReference child of a publisherAssertion are all required to be present. An error will be thrown (E_fatalError) if any of the required components of the relationship expression are omitted.

4.4.2 delete_binding

The delete_binding API call causes one or more instances of bindingTemplate data to be deleted from the UDDI registry.

4.4.2.1 Syntax:

```
<delete_binding generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
    <bindingKey/> [<bindingKey/> …]
</delete_binding>
```

4.4.2.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- ***bindingKey***: one or more *uuid_key* values that represent specific instances of known bindingTemplate data.

4.4.2.3 Returns:

Upon successful completion, a dispositionReport is returned with a single success indicator. References to bindingTemplates that are deleted as a result of this call, such as those referenced by other bindingTemplates (in hostingRedirector elements) are not affected.

4.4.2.4 Caveats:

If any error occurs in processing this API call, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known bindingKey values. No partial results will be returned – if any bindingKey values passed are not valid or if the message contained multiple instances of a uuid_key value, this error will be returned. The key that caused the problem will be clearly indicated in the error text.
- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_userMismatch**: signifies that one or more of the bindingKey values passed refers to a bindingTemplate that is not controlled by the individual associated with the authentication token.

4.4.3 delete_business

The delete_business API call is used to remove one or more business registrations (e.g. registered businessEntity data) and all direct contents from a UDDI registry.

4.4.3.1 Syntax:

```
<delete_business generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
    <businessKey/>
    [<businessKey/> …]
</delete_business>
```

4.4.3.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- ***businessKey***: one or more *uuid_key* values that represent specific instances of known businessEntity data.

4.4.3.3 Returns:

Upon successful completion, a dispositionReport message is returned with a single success indicator.

4.4.3.4 Results:

All of the *natural contents* [17] of a businessEntity, including any currently nested businessService and bindingTemplate data will be permanently removed from the UDDI registry.

Any *projected references*[18] to businessServices deleted in this way are deactivated automatically. References to bindingTemplates that are deleted as a result of this call, such as those referenced by other bindingTemplates (in hostingRedirector elements) are not affected.

If the businessEntity deleted via this API call is involved in any publisher assertion, the assertions that referenced the business registration that is deleted will be automatically deleted.

If a businessEntity is deleted via this API call, and the businessKey of the business being deleted is part of any relationship assertions, the affected assertions will be deleted automatically, and the deleted business will no longer be visible or referenced via the find_relatedBusinesses message.

4.4.3.5 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known businessKey values. No partial results will be returned – if any businessKey values passed are not valid or if the message contained multiple instances of a uuid_key value, this error will be returned. The key that caused the error will be clearly indicated in the error text.

---

[17] When a business registration is first saved, all of the contained data found in the registered businessEntity element is referred to as the natural contents. UDDI defines several types of referencing mechanisms – and referenced elements are not considered to be natural contents of registered businessEntity data. Natural contents can be recognized by matching businessKey values between businessService and businessEntity data, or matching serviceKey values between businessService elements and bindingTemplate elements.

[18] UDDI version 2.0 allows save_business messages to be processed that contain businessService data references that are the natural children of a different business registration. Doing this creates a businessService reference that results in the data associated with the referenced businessService to be projected as though it were contained in the referencing businessEntity. These are called *projected references*.

- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_userMismatch**: signifies that one or more of the businessKey values passed refers to data that is not controlled by the individual who is represented by the authentication token.

4.4.4 delete_publisherAssertions

The delete_publisherAssertions API call causes one or more publisherAssertion elements to be removed from a publisher's *assertion collection*. See the appendix describing relationships and the message named get_publisherAssertions for more information on this collection.

4.4.4.1 Syntax:

```
<delete_publisherAssertions generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
    <publisherAssertion>
        <fromKey/>
        <toKey/>
        <keyedReference/>
    </publisherAssertion>
    [<publisherAssertion/> …]
</delete_publisherAssertions>
```

4.4.4.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- ***publisherAssertion***: one or more publisher assertion structures exactly matching an existing assertion that can be found in the publisher's assertion collection.

4.4.4.3 Returns:

Upon successful completion, a dispositionReport message is returned with a single success indicator.

4.4.4.4 Results:

The UDDI registry scans the assertion collection associated with the publisher account, and removes any assertions that exactly match all parts of each publisherAssertion passed. Any assertions described but that cannot be located will cause an error. Assertions removed in this way will also affect the visibility of relationships that are visible via the find_relatedBusinesses message.

4.4.4.5 Caveats:

If any error occurs in processing this API call, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_assertionNotFound**: signifies that one of the assertion structures passed does not have any corresponding match in the publisher's assertion collection. This will also occur if a publisher assertion appears multiple times in the message. The assertion that caused the problem will be clearly indicated in the error text.
- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
-

4.4.5 delete_service

The delete_service API call is used to remove one or more previously businessService elements from the UDDI registry and from its containing businessEntity parent.

4.4.5.1 Syntax:

```
<delete_service generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
    <serviceKey/>
    [<serviceKey/> …]
</delete_service>
```

4.4.5.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- ***serviceKey***: one or more *uuid_key* values that represent specific instances of known businessService data.

4.4.5.3 Returns:

Upon successful completion, a dispositionReport is returned with a single success indicator.

If a business service being deleted is the target of a business service projection associated with another businessEntity, that reference relationship will be automatically eliminated as a result of this call. All contained bindingTemplate data will also be removed from the registry as a result of this call. Any references to bindingTemplates so removed (such as within other hostingRedirector elements) will not be affected.

4.4.5.4 Caveats:

If any error occurs in processing this API call, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known serviceKey values. No partial results will be returned – if any serviceKey values passed are not valid or if the message contained multiple instances of a uuid_key value, this error will be returned. The key causing the error will be clearly indicated in the error text.
- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_userMismatch**: signifies that one or more of the serviceKey values passed refers to data that is not controlled by the individual who is represented by the authentication token.

4.4.6 delete_tModel

The delete_tModel API call is used to logically delete one or more tModel structures. Logical deletion hides the deleted tModels from find_tModel result sets but does not physically delete it. Deleting an already deleted tModel has no effect.

4.4.6.1 Syntax:

```
<delete_tModel generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
    <tModelKey/> [<tModelKey/> …]
</delete_tModel>
```

4.4.6.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- ***tModelKey*** : one or more URN qualified *uuid_key* values that represent specific instances of known tModel data. All tModelKey values begin with a uuid URN qualifier (e.g. "uuid:" followed by a known tModel UUID value.)

4.4.6.3 Returns:

Upon successful completion, a dispositionReport message is returned with a single success indicator.

If a tModel is hidden in this way it will not be physically deleted as a result of this call. Any tModels hidden in this way are still accessible, via the get_registeredInfo and get_tModelDetail messages, but will be omitted from any results returned by calls to find_tModel. The purpose of the delete_tModel behavior is to ensure that the details associated with a hidden tModel are still available to anyone currently using the tModel. A hidden tModel can be restored and made visible to search results by invoking the save_tModel API at a later time, passing the original data and the tModelKey value of the hidden tModel.

4.4.6.4 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the URN qualified *uuid_key* values passed did not match with any known tModelKey values. No partial results will be returned – if any tModelKey values passed are not valid, this error will be returned. Any tModelKey values passed without a UUID: URN qualifier will be considered invalid. The invalid key references will be clearly indicated in the error text.
- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_userMismatch**: signifies that one or more of the tModelKey values passed refers to data that is not controlled by the individual who is represented by the authentication token.

4.4.7 discard_authToken

The discard_authToken API call is used to inform an Operator Site that the authentication token is to be discarded, effectively ending the session. Subsequent calls that use the same authToken will be rejected. This message is optional for Operator Sites that do not manage session state or that do not support the get_authToken message.

4.4.7.1 Syntax:

```
<discard_authToken generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
</discard_authToken>
```

4.4.7.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.

4.4.7.3 Returns:

Upon successful completion, a dispositionReport is returned with a single success indicator. Discarding an expired authToken will be processed and reported as a success condition.

4.4.7.4 Caveats:

If any error occurs in processing this API call, a dispositionReport structure will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.

4.4.8 get_assertionStatusReport

The get_assertionStatusReport API call provides administrative support for determining the status of current and outstanding publisher assertions that involve any of the business registrations managed by the individual publisher account. Using this message, a publisher can see the status of assertions that they have made, as well as see assertions that others have made that involve businessEntity structures controlled by the calling publisher account. See Appendix J on relationships and publisher assertions for more information.

4.4.8.1 Syntax:

```
<get_assertionStatusReport generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
    [<completionStatus/>]
</get_assertionStatusReport>
```

4.4.8.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- ***completionStatus***: one of the following values
    - **status:complete**: passing this value will cause only the publisher assertions that are complete to be returned. Each businessEntity listed in assertions that are complete has a visible relationship that directly reflects the data in a complete assertion (as per the find_relatedBusinesses message).
    - **status:toKey_incomplete**: passing this value will cause only those publisher assertions where the party who controls the businessEntity referenced by the toKey value in an assertion has not made a matching assertion to be listed.
    - **status:fromKey_incomplete**: passing this value will cause only those publisher assertions where the party who controls the businessEntity referenced by the fromKey value in an assertion has not made a matching assertion to be listed.

  This optional argument lets the publisher restrict the result set to only those relationships that have the status value specified. Assertion status is a calculated result based on the sum total of assertions made by the individuals that control specific business registrations.

4.4.8.3 Returns:

Upon successful completion, an assertionStatusReport message is returned containing assertion status information.

4.4.8.4 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

- **E_invalidCompletionStatus**: signifies that the completionStatus value passed is unrecognized. The completion status that caused the problem will be clearly indicated in the error text.
- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid

4.4.9 get_authToken

The get_authToken API call is used to obtain an authentication token. Authentication tokens are opaque values that are required for all other publisher API calls. This message is not required for Operator Sites that have an external mechanism defined for users to get an authentication token. This API is provided for implementations that do not have some other method of obtaining an authentication token or certificate, or that choose to use user ID and password based authentication.

4.4.9.1 Syntax:

```
<get_authToken generic="2.0" xmlns="urn:uddi-org:api_v2"
    userID="someLoginName"
    cred="someCredential" />
```

4.4.9.2 Arguments:

- ***userID***: this required attribute argument is the user identifier that an individual authorized user was assigned by an Operator Site. Operator Sites will each provide a way for individuals to obtain a userID and password credentials that will be valid only at the given Operator Site.
- ***cred***: this required attribute argument is the password or credential that is associated with the user.

4.4.9.3 Returns:

This function returns an authToken message that contains a valid authInfo element that can be used in subsequent calls to publisher API calls that require an authInfo value.

4.4.9.4 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

- **E_unknownUser**: signifies that the Operator Site that received the request does not recognize the userID and/or cred argument values passed as valid credentials.

4.4.10 get_publisherAssertions

The get_publisherAssertions API call is used to obtain the full set of publisher assertions that is associated with an individual publisher account.

4.4.10.1 Syntax:

```
<get_publisherAssertions generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
</get_publisherAssertions>
```

4.4.10.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.

4.4.10.3 Returns:

This API call returns a publisherAssertions message that contains a publisherAssertion element for each publisher assertion registered by the publisher account associated with the authentication information. Only assertions made by the individual publisher are returned. See get_assertionStatusReport and the appendix explaining publisher assertions for more details.

4.4.10.4 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.

4.4.11 get_registeredInfo

The get_registeredInfo API call is used to get an abbreviated list of all businessEntity and tModel data that are controlled by the individual associated with the credentials passed.

4.4.11.1 Syntax:

```
<get_registeredInfo generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
</get_registeredInfo>
```

4.4.11.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.

4.4.11.3 Returns:

Upon successful completion, a registeredInfo message will be returned, listing abbreviated business information in one or more businessInfo elements, and tModel information in one or more tModelInfo elements. This API is useful for determining the full extent of registered business and tModel information controlled by a single user in a single call. This message complements the get_publisherAssertions message, which returns information about assertions managed by an individual publisher account.

4.4.11.4 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.

4.4.12 save_binding

The save_binding API call is used to save or update a complete bindingTemplate element. This message can be used to add or update one or more bindingTemplate elements as well as the container/contained relationship that each bindingTemplate has with one or more existing businessService elements.

4.4.12.1 Syntax:

```
<save_binding generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
    <bindingTemplate/> [<bindingTemplate/>…]
</save_binding>
```

4.4.12.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- ***bindingTemplate***: one or more complete bindingTemplate elements. To save a new bindingTemplate, pass a bindingTemplate element with an empty bindingKey attribute value. Any bindingTemplate data saved in this way must provide a serviceKey value that references a businessService that is controlled by the same publisher.

Each bindingTemplate element passed must contain a serviceKey value that corresponds to a registered businessService controlled by the same publisher account. The net effect of this call is to determine the containing parent businessService for each bindingTemplate affected by this call. If the same bindingTemplate (determined by matching bindingKey value) is listed more than once, any relationship to the containing businessService will be determined by processing order, which is determined by the position of the bindingTemplate data in first to last order.

Using this message it is possible to move an existing bindingTemplate element from one businessService element to another by simply specifying a different parent businessService relationship along with the complete bindingTemplate. Changing a parent relationship in this way will cause two businessService elements to be affected. The net result of such a move is that the bindingTemplate will still reside within one, and only one businessService element based on the value of the serviceKey passed.

If a bindingTemplate being saved contains a hostingRedirector element, and that element references a bindingTemplate that itself contains a hostingRedirector element, an error condition (E_invalidKeyPassed) will be generated.

4.4.12.3 Returns:

This API returns a bindingDetail message containing the final results of the call that reflects the newly registered information for the effected bindingTemplate elements. If more than one bindingTemplate is saved in a single save_binding message, the resulting bindingDetail message will return results in the same order that they appeared in the save_binding message. If the same bindingTemplate (determined by matching bindingKey) is listed more than once in the save_binding message, it may be listed once in the result for each appearance in the save_binding message. If so, the last appearance in the results represents the final saved state.

4.4.12.4 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo

argument is either missing or is not valid.

- **E_invalidKeyPassed**: signifies that the request cannot be satisfied because one or more *uuid_*key values specified is not a valid key value, or that a hostingRedirector value references a bindingTemplate that itself contains a hostingRedirector value.
- **E_userMismatch**: signifies that one or more of the *uuid_key* values passed refers to data that is not controlled by the individual who is represented by the authentication token.
- **E_accountLimitExceeded**: signifies that user account limits have been exceeded. Account limits are established based on the relationship between an individual publisher and an individual operator. No operators may place other restrictions on publishing limits established by custodial operators.

4.4.13 save_business

The save_business API call is used to save or update information about a complete businessEntity element. This API has the broadest scope of all of the save_xx API calls in the publisher API, and can be used to make sweeping changes to the published information for one or more businessEntity elements controlled by an individual.

UDDI version 2 introduces the ability to use this API message to establish a reference relationship to businessService elements that are managed as the contents of another businessEntity. In this way, a businessService that is a natural part of one businessEntity can appear as a *projected part* of any other businessEntity. Any businessService data projected in this way (by way of a reference established by this API) are not managed as a part of the referencing entity.

4.4.13.1 Syntax:

```
<save_business generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
    <businessEntity/> [<businessEntity/>…]
</save_business>
```

4.4.13.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- ***businessEntity***: one or more complete businessEntity elements can be passed. These elements can be obtained in advance by using the get_businessDetail API call or by any other means.

4.4.13.3 Behavior:

If any of the *uuid_key* values within in a businessEntity element (e.g. any data with a key value regulated by a businessKey, serviceKey, bindingKey, or tModelKey) is passed with a blank value, this is a signal that the data that is so keyed is brand new, and being inserted for the first time.[19] If this case occurs, a new key value will be automatically generated for the data passed without an associated key.

To make this API call perform an update to existing registered data, the keyed entities (businessEntity, businessService, bindingTemplate or tModel) should have *uuid_key* values that correspond to the registered data.

Data can be deleted with this API call when registered information is different than the new information provided. One or more businessService and bindingTemplate elements that are found in the controlling Operator Site but are missing from the businessEntity information provided in or referenced by this call will be deleted from the registry after processing this call.

Data that is contained within one or more businessEntity can be rearranged with this API call when data passed to this function redefines parent container relationships for other registered information. For instance, if a new businessEntity is saved with information about a businessService that is registered already as part of a separate businessEntity, this will result in the businessService being moved from its current container to the new businessEntity. This condition occurs when the businessKey attribute contained in the businessService element being saved matches the businessKey attribute value associated with the businessEntity being saved. It only applies if the same party controls the data referenced. Any attempt to delete or move a businessService in this manner by a party who is not the publisher of the businessService will be rejected as an error.

If the businessEntity being saved contains a businessService element that has a businessKey that refers to some businessEntity other than the businessEntity being saved, the UDDI registry will note a reference, called a “service projection”, to the existing businessService. Subsequent calls to the

[19] This does not apply to structures that reference other keyed data, such as tModelKey references within bindingTemplate or keyedReference structures, since these are references.

get_businessDetail API, passing either the businessKey of the businessEntity that contains the referenced businessService or the businessKey of the businessEntity that is associated with the referenced businessService will result in an identical businessService element being included as part of the result set.

No changes to the referenced businessService will be effected by the act of establishing a service projection. Existing service projections associated with the businessEntity being saved that are not contained in the call to save_business are deleted automatically. This automatic reference deletion will not cause any changes to the referenced businessService. If the referenced businessService is deleted by any means, all references to it associated with other businessEntities are left untouched. Such "broken" service projections appear in their businessEntity as businessService elements containing the businessKey and serviceKey attributes as their only content. For this reason, it is a best practice to coordinate references to businessService data published under another businessEntity with the party who manages that data.

When saving a businessEntity containing a service reference, the content of the businessService provided in the save_business (except for the businessKey and the serviceKey attributes) is ignored.

For each businessEntity saved with this API, the Operator Site will create a URL that is specific to the Operator Site that can be used to get (via HTTP-GET) the businessEntity element being registered. This information will be added (if not present already) to the discoveryURLs collection associated with each businessEntity. The discoveryURL values generated in this way will have a useType value of "businessEntity".

4.4.13.4 Returns:

This API returns a businessDetail message containing the final results of the call that reflects the new registered information for the businessEntity information provided. These results will include any businessServices that are contained by reference. If the same entity (businessEntity, businessService, or bindingTemplate), determined by matching key, is listed more than once in the save_business message, it may be listed once in the result for each appearance in the save_business message. If so, the last appearance in the results represents the final saved state.

4.4.13.5 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_invalidKeyPassed**: signifies that the request cannot be satisfied because one or more *uuid_*key values specified is not a valid key value. This includes any tModelKey references, as well as references to serviceKey or bindingKey values that either do not exist. The key causing the error will be clearly indicated in the error text.
- **E_invalidProjection**: signifies that an attempt was made to save a businessEntity containing a service projection that does not match the businessService being projected. The serviceKey of at least one such businessService will be included in the dispositionReport.
- **E_userMismatch**: signifies that one or more of the *uuid_key* values passed refers to data that is not controlled by the individual who is represented by the authentication token. The key causing the error will be clearly indicated in the error text.
- **E_invalidValue**: A value that was passed in a keyValue attribute did not pass validation. This applies to checked categorizations, identifiers and other validated code lists. The error text will clearly indicate the key and value combination that failed validation.
- **E_requestTimeout**: Signifies that the request could not be carried out because a needed validate_values service did not respond in a reasonable amount of time. Details identifying the

failing service will be included in the dispositionReport element.

- **E_valueNotAllowed**: Restrictions have been placed by the taxonomy provider on the types of information that should be included at that location within a specific taxonomy. A validate_values service chosen by the Operator Site has rejected this businessEntity for at least one specified category.
- **E_accountLimitExceeded**: signifies that user account limits have been exceeded.

4.4.14 save_service

The save_service API call adds or updates one or more businessService elements.

4.4.14.1 Syntax:

```
<save_service generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
    <businessService/> [<businessService/>…]
</save_service>
```

4.4.14.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- ***businessService***: one or more complete businessService elements can be passed. For the purpose of performing round trip updates, this data can be obtained in advance by using the get_serviceDetail API call or by any other means.

4.4.14.3 Behavior:

Each businessService element passed must contain a businessKey value that corresponds to a registered businessEntity controlled by the same publisher who is making the save_service request.

If the same businessService, or within these, bindingTemplate (determined by matching businessService or bindingKey value) is contained in more than one businessService argument, any relationship to the containing businessEntity will be determined by processing order – which is determined by first to last order of the information passed in the request.

Using this API call it is possible to move an existing bindingTemplate element from one businessService element to another, or move an existing businessService element from one businessEntity to another by simply specifying a different parent businessEntity relationship. Changing a parent relationship in this way will cause two businessEntity elements to be affected.

4.4.14.4 Returns:

This API call returns a serviceDetail message containing the final results of the call that reflects the newly registered information for the effected businessService elements. In cases where multiple businessService elements are passed in the request, the result will contain the final results for each businessService passed and these will occur in the same order as found in the request. If the same entity (businessService, or bindingTemplate), determined by matching key, is listed more than once in the save_service message, it may be listed once in the result for each appearance in the save_service message. If so, the last appearance in the results represents the final saved state.

4.4.14.5 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_invalidKeyPassed**: signifies that the request cannot be satisfied because one or more *uuid_*key values specified is not a valid key value. This includes any tModelKey references to non-existent tModel data. The key causing the error will be clearly indicated in the error text.
- **E_userMismatch**: signifies that one or more of the *uuid_key* values passed refers to data that is not controlled by the individual who is represented by the authentication token.

- **E_invalidValue**: A value that was passed in a keyValue attribute did not pass validation. This applies to checked categorizations, identifiers and other validated code lists. The error text will clearly indicate the key and value combination that failed validation.
- **E_requestTimeout**: Signifies that the request could not be carried out because a needed validate_values service did not respond in a reasonable amount of time. Details identifying the failing service will be included in the dispositionReport element.
- **E_valueNotAllowed**: The taxonomy validation routine chosen by the Operator Site has rejected the businessService data provided.
- **E_accountLimitExceeded**: signifies that user account limits have been exceeded.

4.4.15 save_tModel

The save_tModel API call adds or updates one or more registered tModel elements.

4.4.15.1 Syntax:

```
<save_tModel generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
    <tModel/> [<tModel/>…]
</save_tModel>
```

4.4.15.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- ***tModel***: one or more complete tModel elements can be passed. If adding a new tModel, the tModelKey value should be passed as an empty element.

4.4.15.3 Behavior:

If any of the *uuid_key* values within in a tModel element (e.g. tModelKey) is passed with a blank value, this is a signal that a new tModel is being inserted and that the UDDI registry should assign a new tModelKey identifier to this data. The new key value will be returned in the tModelDetail response.

To make this API call perform an update to existing registered data, the tModelKey values should have *uuid_key* values that correspond to the registered data. All tModelKey values that are non-blank are formatted as urn values (e.g. the characters "uuid:" precede all UUID values for tModelKey values)

If a tModelKey value is passed that corresponds to a tModel that was previously hidden via the delete_tModel message, the result will be the restoration of the tModel to full visibility , making it available for return in find_tModel results.

4.4.15.4 Returns:

This API returns a tModelDetail message containing the final results of the call that reflects the new registered information for the effected tModel elements. If multiple tModel elements were passed in the save_tModel request, the order of the response will exactly match the order the elements appeared in the save. If the same tModel, determined by matching key, is listed more than once in the save_tModel message, it may be listed once in the result for each appearance in the save_tModel message. If so, the last appearance in the results represents the final saved state.

4.4.15.5 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_invalidKeyPassed**: signifies that the request cannot be satisfied because one or more *uuid_*key values specified is not a valid key value. This will occur if a *uuid_key* value is passed in a tModel that does not match with any known tModel key. The key value that causes an error will be indicated clearly in the error text.
- **E_userMismatch**: signifies that one or more of the *uuid_key* values passed refers to data that is not controlled by the individual who is represented by the authentication token.
- **E_invalidValue**: A value that was passed in a keyValue attribute did not pass validation. This applies to checked categorizations, identifiers and other validated code lists. The error text will

clearly indicate the key and value combination that failed validation.

- **E_requestTimeout**: Signifies that the request could not be carried out because a needed validate_values service did not respond in a reasonable amount of time. Details identifying the failing service will be included in the dispositionReport element.
- **E_valueNotAllowed**: Restrictions have been placed by the taxonomy provider on the types of information that should be included at that location within a specific taxonomy. The validation routine chosen by the Operator Site has rejected this tModel for at least one specified category.
- **E_accountLimitExceeded**: signifies that user account limits have been exceeded.

4.4.16 set_publisherAssertions

The set_publisherAssertions API call is used to manage all of the tracked relationship assertions associated with an individual publisher account. See the appendix on relationship assertions for more information.

4.4.16.1 Syntax:

```
<set_publisherAssertions generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo/>
    [<publisherAssertion>
        <fromKey/>
        <toKey/>
        <keyedReference/>
        </publisherAssertion>…]
</set_publisherAssertions>
```

4.4.16.2 Arguments:

- ***authInfo***: this required argument is an element that contains an authentication token. Authentication tokens are obtained using the get_authToken API call.
- ***publisherAssertion***: zero or more relationship assertions. Relationship assertions consist of a reference to two businessEntity key values as designated by the fromKey and toKey elements, as well as a required expression of directional relationship within the contained keyedReference element. See the appendix on managing relationships. The fromKey, the toKey, and all three parts of the keyedReference – the tModelKey, the keyName, and the keyValue – must be specified. Empty (zero length) keyNames and keyValues are permitted.

4.4.16.3 Returns:

Upon successful completion, a publisherAssertions message is returned containing all of the relationship assertions currently attributed to the publisher account that is associated with the authInfo data passed.

4.4.16.4 Results:

The full set of assertions associated with a publisher is effectively replaced whenever this message is used. When this message is processed, the publisher assertions that are active prior to this API call for a given publisher account are examined by the UDDI registry. Any new assertions not present prior to the call are added to the assertions attributed to the publisher. As a result, new relationships may be activated (e.g. determined to have a completed status), and existing relationships may be deactivated.

Any relationships attributed to assertions previously present but not present in the data provided in this call are deactivated. For the sake of determining a unique row within an assertion set, all values within the publisherAssertion element are used as part of the unique row determination. Any differences in any of the individual publisherAssertion element contents constitute a new unique assertion for purposes of detecting new assertions. The direction of the relationship, as indicated by the two businessKey values in the fromKey and toKey elements, is relevant in determining assertion uniqueness..

The publisher must control the fromKey, the toKey, or both. If both of the businessKey values passed within an assertion are controlled by the publisher, then the assertion is automatically complete and the relationship described in the assertion will be visible via the find_relatedBusinesses API. To form a relationship when the publisher only controls one of the two keys passed, the assertion must be matched exactly by an assertion made by the publisher who controls the other business referenced. Assertions exactly match if and only if they 1) refer to the same businessEntity in their fromKeys; 2) refer to the same businessEntity in their toKeys; 3) refer to the same tModel in their tModelKeys; 4) have identical keyNames; and 5) have identical keyValues.

4.4.16.5 Caveats:

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller in a SOAP Fault. The following error number information will be relevant:

- **E_invalidKeyPassed**: signifies that one of the *uuid_key* values passed did not match with any known businessKey or tModelKey values. The assertion element and the key that caused the problem will be clearly indicated in the error text.
- **E_authTokenExpired**: signifies that the authentication token value passed in the authInfo argument is no longer valid because the token has expired.
- **E_authTokenRequired**: signifies that the authentication token value passed in the authInfo argument is either missing or is not valid.
- **E_userMismatch**: signifies that neither of the businessKey values passed in the embedded fromKey and toKey elements is controlled by the publisher account associated with the authentication token. The error text will clearly indicate which assertion caused the error.

5 Appendix A: Error code reference

5.1 Error Codes

The following list of error codes can be returned in the error code and error number (errCode and errno attributes) within a dispositionReport response to the API calls defined in this programmer's reference[20]. The descriptions in this section are general and when used with the specific return information defined in the individual API call descriptions are useful for determining the reason for failures or other reasons. A list of the valid UDDI errCode (errno) values follows:

- **E_assertionNotFound**: (30000) Signifies that a particular publisher assertion (consisting of two businessKey values, and a keyed reference with three components) cannot be identified in a save or delete operation.
- **E_authTokenExpired**: (10110) Signifies that the authentication token information has timed out.
- **E_authTokenRequired**: (10120) Signifies that an invalid authentication token was passed to an API call that requires authentication.
- **E_accountLimitExceeded**: (10160) Signifies that a save request exceeded the quantity limits for a given data type. See "Structure Limits" in Appendix D for details.
- **E_busy**: (10400) Signifies that the request cannot be processed at the current time.
- **E_categorizationNotAllowed**: (20100) RETIRED. Used for UDDI Version 1.0 compatibility. Replaced by E_valueNotAllowed in 2 and higher. Restrictions have been placed on the types of information that can be categorized within a specific taxonomy. The data provided does not conform to the restrictions placed on the category used. Used with categorization only.
- **E_fatalError**: (10500) Signifies that a serious technical error has occurred while processing the request.
- **E_invalidKeyPassed**: (10210) Signifies that the uuid_key value passed did not match with any known key values. The details on the invalid key will be included in the dispositionReport element.
- **E_invalidProjection**: (20230) Signifies that an attempt was made to save a businessEntity containing a service projection that does not match the businessService being projected. The serviceKey of at least one such businessService will be included in the dispositionReport.
- **E_invalidCategory** (20000): RETIRED. Used for UDDI Version 1.0 compatibility only. Replaced by E_invalidValue in version 2 and higher. Signifies that the given keyValue did not correspond to a category within the taxonomy identified by the tModelKey. Used with categorization only.
- **E_invalidCompletionStatus**: (30100) signifies that one of the assertion status values passed is unrecognized. The completion status that caused the problem will be clearly indicated in the error text.
- **E_invalidURLPassed**: (10220) DO NOT USE. Signifies that an error occurred during processing of a save function involving accessing data from a remote URL. The details of the HTTP Get report will be included in the dispositionReport element. Not used in V1 or V2.

[20] Certain error codes specified in UDDI version 1 were determined to be problematic and were not used in either UDDI version 1 or version 2. Other error codes used in version 1 have been superceded in UDDI version 2. Error codes listed here that are marked "DO NOT USE" are not to be used by UDDI operators. Error codes listed here that are marked "RETIRED" are still used for version 1 compatibility, but in version 2 and higher, these retired codes are superceded.

- **E_invalidValue**: (20200) A value that was passed in a keyValue attribute did not pass validation. This applies to checked categorizations, identifiers and other validated code lists. The error text will clearly indicate the key and value combination that failed validation.
- **E_keyRetired**: (10310) DO NOT USE. Signifies that a *uuid_key* value passed has been removed from the registry. While the key was once valid as an accessor, and is still possibly valid, the publisher has removed the information referenced by the *uuid_key* passed. V1 errata – not used. Included here for historical code-set completion.
- **E_languageError**: (10060) Signifies that an error was detected while processing elements that were annotated with xml:lang qualifiers. Presently, only the description and name elements support xml:lang qualifications.
- **E_messageTooLarge**: (30110) Signifies that the message is too large. The upper limit will be clearly indicated in the error text.
- **E_nameTooLong**: (10020) RETIRED. Used for UDDI Version 1.0 compatibility only. Signifies that the partial name value passed exceeds the maximum name length designated by the policy of an implementation or Operator Site.
- **E_operatorMismatch**: (10130) DO NOT USE. Signifies that an attempt was made to use the publishing API to change data that is mastered at another Operator Site. This error is only relevant to the public Operator Sites and does not apply to other compatible registries. V1 defined this in error – caused precedence problems with E_unknownUser. Included here for historical code set completeness. Retired.
- **E_publisherCancelled**: (30220) The target publisher cancelled the custody transfer operation.
- **E_requestDenied**: (30210) A custody transfer request has been refused.
- **E_requestTimeout**: (20240) Signifies that the request could not be carried out because a needed web service, such as validate_values, did not respond in a reasonable amount of time. Details identifying the failing service will be included in the dispositionReport element.
- **E_secretUnknown**: (30230) The target publisher was unable to match the shared secret and the five (5) attempt limit was exhausted. The target operator automatically cancelled the transfer operation.
- **E_success**: (0) Signifies no failure occurred. This return code is used with the dispositionReport for reporting results from requests with no natural response document.
- **E_tooManyOptions**: (10030) Signifies that too many or incompatible arguments were passed. The error text will clearly indicate the nature of the problem.
- **E_transferAborted**: (30200) Signifies that a custody transfer request will not succeed.
- **E_unrecognizedVersion**: (10040) Signifies that the value of the *generic* attribute passed is unsupported by the *Operator Instance* being queried.
- **E_unknownUser**: (10150) Signifies that the user ID and password pair passed in a get_authToken message is not known to the Operator Site or is not valid.
- **E_unsupported**: (10050) Signifies that the implementer does not support a feature or API.
- **E_userMismatch**: (10140) Signifies that an attempt was made to use the publishing API to change data that is controlled by another party.
- **E_valueNotAllowed**: (20210) Signifies that a value did not pass validation because of contextual issues. The value may be valid in some contexts, but not in the context used. The error text may contain information about the contextual problem.
- **E_unvalidatable**: (20220) Signifies that an attempt was made to reference a taxonomy or identifier system in a keyedReference whose tModel is categorized with the unvalidatable categorization.

If a V2 registry encounters an error while processing a V1 message it may only return a V1 message.

Errors that, for V2 messages, would be reported with non-V1 error codes are reported with E_fatalError. These error codes are: E_invalidValue, E_valueNotAllowed, E_invalidProjection, E_assertionNotFound, E_invalidCompletionStatus, E_messageTooLarge, E_transferAborted, E_requestDenied, E_publisherCancelled, and E_secretUnknown.

Non-error conditions are not reported by way of SOAP Faults but are instead reported using the dispositionReport element.

5.1.1 Success reporting with the dispositionReport element:

The general form of a success report is:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoaporg.org/soap/envelope/">
<Body>
   <dispositionReport generic="2.0" operator="OperatorURI"
        xmlns="urn:uddi-org:api_v2" >
      <result errno="0" >
         <errInfo errCode="E_success" />
      </result>
   </dispositionReport>
</Body>
</Envelope>
```

In the case of success messages, the dispositionReport element is used as a normal SOAP message with the dispositionReport returned directly within the SOAP Body element.

5.1.2 Error reporting with the dispositionReport element:

All application errors are communicated via the use of the SOAP FAULT element. The general form of an error report is:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoaporg.org/soap/envelope/">
<Body>
     <Fault>
        <faultcode>Client</faultcode>
        <faultstring>Client Error</faultstring>
        <detail>
           <dispositionReport generic="2.0" operator="OperatorURI"
                xmlns="urn:uddi-org:api_v2" >
              <result errno="10050" >
                 <errInfo errCode="E_fatalError">
                    The findQualifier value passed is unrecognized: XYZ
                 </errInfo>
              </result>
           </dispositionReport>
        </detail>
     </Fault>
</Body>
</Envelope>
```

Multiple *result* elements may be present within the dispositionReport element, and can be used to provide very detailed error reports for multiple error conditions. The number of *result* elements returned within a disposition report is implementation specific. In general it is permissible to return an error response as soon as the first error in a request is detected. References within the API reference sections that describe error text content rules pertain to the content of the errInfo element.

When a registry receives a message that is indeterminate with respect to its UDDI version (for example because the generic attribute is “1.0” and the XML namespace is “urn:uddi-org:api_v2”) UDDI version 2 registries MUST return a version 2 fault as described above.

6 Appendix B: SOAP usage details

This appendix covers the SOAP specific conventions and requirements for Operator Sites.

6.1 Support for SOAPAction

SOAP 1.1 requires the presence of the HTTP header field named SOAPAction when an HTTP binding is specified. UDDI requires the presence of this HTTP Header field to be SOAP 1.1 compliant. Different SOAP toolkits treat this HTTP header field differently. UDDI version 1.0 required that this value be an empty string bounded by quotes. For maximum tool compatibility, UDDI version 2.0 allows any value to be specified for SOAPAction, including an empty string.. This behavior applies to both version 2.0 and version 1.0 messages.

6.1.1 Example

Both of the following message styles are permitted in UDDI version 2.0.

```
POST /someVerbHere HTTP/1.1
Host: www.someoperator.org
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
     <Body>
        <get_bindingDetail generic="2.0"
            xmlns="urn:uddi-org:api_v2">
  …
```

and

```
POST /someVerbHere HTTP/1.1
Host: www.someoperator.org
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "urn:uddi-org:api_v2#get_bindingDetail"

<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
     <Body>
        <get_bindingDetail generic="2.0"
            xmlns="urn:uddi-org:api_v2">
…
```

6.2 Support for SOAP Actor

In version 1 and version 2 of the UDDI specification, the SOAP Actor feature is not supported. Operator Sites will reject any request that arrives with a generic SOAP fault with no detail element and a “Client” faultcode. The faultstring will clearly indicate the problem.

6.3 Support for SOAP encoding

In UDDI version 2 there must be no claims made about the encoding style of any element within the

"urn:uddi-org:*" namespace. If such claims are made, the UDDI server must respond with a generic SOAP fault with no detail element and a "Client" faultcode. The faultstring will clearly indicate the problem.

6.4 Support for SOAP Fault

UDDI registries signal a generic SOAP Fault[21] when unknown API references are invoked, validation failures occur, etc. UDDI specific errors will be handled via a SOAP Fault element containing a UDDI dispositionReport element (see appendix A). The following SOAP fault codes are used:

- **VersionMismatch**: An invalid namespace reference for the SOAP envelope element was passed. The valid namespace value is "http://www.xmlsoap.org/soap/envelope/".
- **MustUnderstand**: A SOAP header element was passed to an Operator Site. Operator Sites do not support any SOAP headers, and will return this error whenever a SOAP request is received that contains any Headers element.
- **Client**: A message was incorrectly formed or did not contain enough information to perform more exhaustive error reporting.
- **Server**: The Server class of errors indicate that the message could not be processed for reasons not directly attributable to the contents of the message itself but rather to the processing of the message. For example, processing could include communicating with an upstream processor, which didn't respond. The message may succeed at a later point in time.

6.5 Support for SOAP Headers

In UDDI version 1.0 and 2.0, SOAP Headers are not supported. Operator Sites are permitted to ignore most extension headers received. SOAP headers that have the must_understand attribute set to true will be rejected with a SOAP fault - MustUnderstand.

6.6 XML prefix conventions – default namespace support

UDDI Operator Sites are required to support the use of the default namespaces (i.e. no XML prefix) in SOAP request and response documents as shown in the following HTTP example:

```
POST /someVerbHere HTTP/1.1
Host: www.someoperator.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: ""

<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
     <Body>
        <get_bindingDetail generic="2.0"
            xmlns="urn:uddi-org:api_v2">
  ...
```

6.7 Support for Unicode: SOAP listener behavior

The decision to use the UTF-8 encoding in all requests simplified the number of encoding variations that need to be handled within the XML interchanges used in this API specification. This section

[21] See section 4.4.1, "SOAP Fault Codes," in SOAP 1.1 specification for descriptive information.

describes the behavior of the SOAP listeners that run at Operator Sites in regards to the support of Unicode in XML.

Unicode UTF-8 allows data to be transmitted with an optional three-byte signature, also known as Byte Order Mark (BOM), preceding the XML data. This signature does not contain information that is useful for decoding the contents; but, in the case of UTF-8, tells the receiving program that the rest of the text is in UTF-8. Its presence makes no difference to the endianness of the byte stream as UTF-8 always has the same byte order. The BOM is not part of the textual content, and it is safe for Operator Sites to remove the BOM prior to processing messages received.

Operator Sites must be prepared to accept messages that contain Byte Order Marks, but the BOM is not required to process SOAP messages successfully.

Operator Sites will not return a BOM with any of the response messages defined in this specification. All such responses will be encoded in UTF-8.

All UDDI Operator Sites must support all of the Unicode characters, including all compatibility characters. This is to support transmission and processing of legacy data in certain languages. Operator Sites are not required to understand the nature of compatibility characters for substitutability or equivalence purposes, but must preserve the data as sent by a publisher.

6.8 Maximum Message Size

The maximum message size is 2 megabytes.

7 Appendix C: XML Usage Details

This appendix explains the specifics of XML conventions employed across all UDDI Operator Sites. Implementations that desire to remain compliant with the behaviors of Operator Sites should follow these same conventions.

7.1 Support for multiple languages

Many of the messages defined in this specification contain elements named *name* and *description*. Multiple names and descriptions are allowed in order to accommodate descriptions in multiple languages. In order to signify the language that these elements are expressed in, they MAY carry xml:lang values.

See section 4.3.6, *Special considerations for the xml:lang attribute* for details.

7.1.1 Valid Language Codes

The XML specification defines the valid values for the xml:lang markup. At the time of this specification, XML references the IETF standard known as RFC 1766. The value of the xml:lang attribute can consist of 1 or more parts: a primary language tag and a (possibly empty) series of sub-tags for country or dialect identification. Further information can be found at:

http://www.ietf.org/rfc/rfc1766.txt

Only one description or name element is allowed for each language code used at any container level.

7.1.2 Default Language Codes

A default language code MAY be determined for a publisher at the time that a party establishes permissions to publish at a given Operator Site or implementation. This default language code MAY be applied to any description values that are provided with no language code. If no default language is established, the operator MAY adopt an appropriate default language code.

7.2 XML Encoding requirements

All messages sent to and received from the Operator Site shall be encoded as UTF-8, and shall specify the HTTP Content-Type header with a charset parameter of "utf-8". All such messages shall also have the 'encoding="UTF-8"' markup in the XML-DECL that appears on the initial line. Other encoding name variants, such as UTF8, UTF_8, etc. shall not be used. Therefore, to be explicit, the initial line shall be:

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

and the Content-Type header shall be:

```
Content-type: text/xml; charset="utf-8"
```

Operator sites MUST reject messages that do not conform to this requirement.

8 Appendix D: Security model in the publishers API

The Publishers API describes the messages that are used to control the content contained within an Operator Site, and can be used by compliant non-operator implementations that adhere to the behaviors described in this programmers reference specification.

8.1 Authentication of publisher API calls

All UDDI publisher API messages are secured via SSL 3.0. All calls made to Operator Sites that use the messages defined in the publishers API will be transported using SSL encryption. Operator Sites will each provide a service description that exposes a bindingTemplate that makes use of HTTPS and SSL to secure the transmission of data.

8.1.1 Authentication

Each of the calls in the publishers API that change information at a given Operator Site requires the use of an opaque authentication token. These tokens are generated by or provided by each Operator Site independently, and are passed from the caller to the Operator Site in the element named *authInfo*.

These tokens are meaningful only to the Operator Site that provided them and are to be used according to the published policies of a given Operator Site.

Each party who has been granted publication access to a given Operator Site will be provided a token by the site. Obtaining this token is Operator Site specific.

8.1.2 Establishing credentials

Before any party can publish data within an Operator Site, credentials and permission to publish must be established with the individual operator. Establishing publishing credentials involves providing some verifiable identification information, contact information and establishing security credentials with the individual Operator Site. The specifics of these establishing credentials is Operator Site dependant, and all valid Operator Sites will provide a Web-based user interface via which to establish an identity and secure permissions to publish data.

8.1.3 Authentication tokens are not portable

Every registry implementation that adheres to these specifications will establish their own mechanism for token generation and authentication. The only requirement placed on token generation for use with the publishers API is that the tokens themselves must be valid string text that can be placed within the authInfo XML element. Given that binary to string translations are well understood and in common use, this requirement will not introduce hardships.

Authentication tokens are not required to be valid except at the Operator Site or implementation from which they originated. These tokens need only have meaning at a single Operator Site or implementation, and will not be expected to work across sites.

8.1.4 Generating Authentication Tokens

Many implementations are expected to require a login step. The get_authToken message is provided to accommodate those implementations that desire a login step. Security schemes that are based on the convention of exchanging User ID and password credentials fall into this category. For implementations that desire this kind of security, the get_authToken API is provided as an optional means for generating a temporary authentication token.

Certificate based authentication and similar security mechanisms do not require this additional step of “logging in” and can directly pass compatible authentication token information (such as a certificate value) within the authInfo element provided on each of the publishers API messages. If certificate

based authentication or similar security is employed by the choice of a given Operator Site, the use of the get_authToken and discard_authToken messages is optional.

8.2 Per-account space limits

Operator Sites may impose limits on the amount of data that can be published by a given user. The initial limits for a new user are:

- businessEntity: 1 per user account
- businessService: 4 per businessEntity
- bindingTemplate: 2 per businessService
- tModel: 100 per user account
- publisherAssertion: 10 per user account

Individual user accounts can negotiate per-account limits with the Operator Site. UDDI version 2.0 addressed the need to treat tModel data differently. To establish publisher limits that are higher than the amounts shown here, publishers should work directly with a UDDI operator site using the contact information found on each operator site.

9 Appendix E: findQualifiers

The Inquiry API functions *find_binding, find_business, find_relatedBusinesses, find_service,* and *find_tModel* each will accept an optional element named findQualifiers. This element argument is provided as a means to allow the caller to override default search behaviors.

UDDI Version 2 introduces five new findQualifier values. These are:

- orLikeKeys
- orAllKeys
- combineCategoryBags
- serviceSubset
- andAllKeys.

These findQualifiers may only be used with UDDI Version 2 messages. Attempting to use them with Version 1 messages will result in a dispositionReport with an error code of E_unsupported.

9.1 General form of findQualifiers

findQualifiers are expressed by passing a findQualifiers element where appropriate. The general form of the findQualifiers element is:

```
<findQualifiers>
   <findQualifier>searchQualifierValue</findQualifier>
   [<findQualifier>searchQualifierValue</findQualifier> …]
</findQualifiers>
```

9.1.1 findQualifiers enumerated

The value passed in each findQualifier element represents the behavior change desired by the caller. Not every findQualifier is applicable to every find_xx. APIs ignore findQualifiers that are not applicable. These values must come from the following list of qualifiers:

- **exactNameMatch**: signifies that lexical-order – i.e., leftmost in left-to-right languages – name match behavior should be overridden. When this behavior is specified, only entries that exactly match the entry passed in the name argument will be returned. Applies to: find_business, find_service, and find_tModel.
- **caseSensitiveMatch**: signifies that the default case-insensitive behavior of a name match should be overridden. When this behavior is specified, case is relevant in the search results and only entries that match the case of the value passed in the name argument will be returned. Applies to: find_business, find_service, and find_tModel.
- **sortByNameAsc**: signifies that the result returned by a *find_xx* inquiry call should be sorted on the name field in ascending alphabetic sort order. When there is more than one name field, the sort uses the first of them. This sort is applied prior to any truncation of result sets. Only applicable on queries that return a *name* element in the topmost detail level of the result set. If no conflicting sort qualifier is specified, this is the default sort order for inquiries that return *name* values at this topmost detail level. Applies to: find_business, find_relatedBusinesses, find_service, and find_tModel.
- **sortByNameDesc**: signifies that the result returned by a *find_xx* inquiry call should be sorted on the name field in descending alphabetic sort order. When there is more than one name

field, the sort uses the first of them. This sort is applied prior to any truncation of result sets. Only applicable on queries that return a *name* element in the topmost detail level of the result set. This is the reverse of the default sort order for this kind of result. Applies to: find_business, find_relatedBusinesses, find_service, and find_tModel.

- **sortByDateAsc**: signifies that the result returned by a *find_xx* inquiry call should be sorted based on the date last updated in ascending chronological sort order (earliest returns first). If no conflicting sort qualifier is specified, this is the default sort order for all result sets. Applies to: find_binding, find_business, find_relatedBusinesses, find_service, and find_tModel.
- **sortByDateDesc**: signifies that the result returned by a *find_xx* inquiry call should be sorted based on the date last updated in descending chronological sort order (most recent change returns first). Sort qualifiers involving date are secondary in precedence to the sortByName qualifiers. This causes sortByName elements to be sorted within name by date, newest to oldest. Applies to: find_binding, find_business, find_relatedBusinesses, find_service, and find_tModel.
- **orLikeKeys**: when a bag container contains multiple keyedReference elements (i.e., categoryBag or identifierBag), any keyedReference filters that come from the same namespace (e.g. have the same tModelKey value) are OR'd together rather than AND'd. This allows one to say "*any of these four values from this namespace, and any of these two values from this namespace*". Applies to: find_business, find_service, and find_tModel.
- **orAllKeys**: this changes the behavior for tModelBag and categoryBag to OR keys rather than AND them. This qualifier negates any AND treatment as well as the effect of orLikeKeys. Applies to: find_binding, find_business, find_service, and find_tModel.
- **combineCategoryBags**: this is only used in the find_business message. This qualifier makes the categoryBag entries for the full businessEntity element behave as though all categoryBag elements found at the businessEntity level and in all contained or referenced businessService elements were combined. Searching for a category will yield a positive match on a registered business if any of the categoryBag elements contained within the full businessEntity element (including the categoryBag elements within contained or referenced businessService elements) contains the filter criteria. Applies to: find_business.
- **serviceSubset**: this is used only in the find_business message. This qualifier is used in only in conjunction with a passed categoryBag argument and causes the component of the search that involves categorization to use only the categoryBag elements from contained or referenced businessService elements within the registered data, and ignores any entries found in the categoryBag direct descendent element of registered businessEntity elements. The resulting businessList message will return those businesses that matched based on this modified behavior, in conjunction with any other search arguments provided. Additionally, the contained serviceInfos elements will only reflect summary data (in a serviceInfo element) for those services (contained or referenced) that matched on one of the supplied categoryBag arguments. Applies to: find_business.
- **andAllKeys**: this changes the behavior for identifierBag to AND keys rather than OR them. Applies to: find_business and find_tModel.

At this time, these are the only qualifiers defined. Operator Sites may define more search qualifier values than these – but all Operator Sites and fully compatible software must support these qualifiers and behaviors.

9.1.2 findQualifier Applicability and Precedence

Using a findQualifier with one of the find_xx APIs to which it does not apply results in that qualifier being ignored. Specifying mutually exclusive find qualifiers that do apply results in an E_unsupported error.

Precedence of findQualifiers, when combined is as follows:

1.The following findQualifiers occur at the same level of precedence. They are listed as two bullets for

clarity:

- **exactNameMatch**, **caseSensitiveMatch**: These can be combined or used separately.
- **orAllKeys, orLikeKeys, andAllKeys**: These are mutually exclusive.

2. Explicitly specified **sortByNameAsc**, **sortByNameDesc**: These are mutually exclusive.

3. Explicitly specified **sortByDateAsc**, **sortByDateDesc**: These are mutually exclusive.

4. Implicitly specified (defaulted) sortByNameAsc.

5. Implicitly specified (defaulted) sortByDateAsc.

6. serviceSubset, combineCategoryBags: these conditions are orthogonal to the existence of the other findQualifiers.

The precedence order is used to determine the proper ordering of results for a given find_*xx* message.

9.1.2.1 Examples

findQualifier(s) specified	**Resulting sort order**
None	By name ascending; within name, by date ascending.
sortByNameDesc	By name descending; within name, by date ascending.
sortByDateAsc	By date ascending; within date, by name ascending.
sortByDateDesc, sortByNameDesc	By name descending; within name, by date descending.

9.1.3 Sorting Details

Comparison and sorting is performed based on binary sort orders. This applies to sortByNameAsc, sortByNameDesc, exactNameMatch. By default, case insensitive sorting is performed unless caseSensitiveMatch is specified as a findQualifier value.

10 Appendix F: Response message reference

Here we list each of the response messages. These are technically defined in the UDDI API schema as well as the UDDI data structure reference.

- **assertionStatusReport**: This element is returned by the get_assertionStatusReport and is used by a publisher to determine the status of assertions made by either the publisher or by other parties. Assertions are used to manage the visibility of relationship information related to specific pairs of businessEntity data.
- **authToken**: This element is returned by the optional get_authToken message to return authentication information. The value returned is used in subsequent calls that require an authInfo value.
- **bindingDetail**: This element is the technical information required to make a method call to an advertised web service. It is returned in response to the get_bindingDetail message.
- **businessDetail**: This element contains full details for zero or more businessEntity elements. It is returned in response to a get_businessDetail message, and optionally in response to the save_business message.
- **businessDetailExt**: This element allows compatible registries to define and share extended information about a businessEntity. Operator Sites support this message but return no additional data. This element contains zero or more businessEntityExt elements. It is returned in response to a get_businessDetailExt message.
- **businessList**: This element contains abbreviated information about registered businessEntity information. This message contains zero or more businessInfo elements. It is returned in response to a find_business message.
- **dispositionReport**: This element is used to report the outcome of message processing and to report errors discovered during processing. This message contains one or more result elements. A special case – success – contains only one result element with the special errno attribute value of E_success (0).
- **publisherAssertions**: This element is returned in response to a get_publisherAssertions message. It contains all of the assertions (that are controlled by an individual publisher). This full set is called the publisher's *assertion collection*. Assertions are used to manage the visibility of relationship information related to specific pairs of businessEntity data
- **registeredInfo**: This element contains abbreviated information about all registered businessEntity and tModel information that are controlled by the party specified in the request. This message contains one or more businessInfo elements and zero or more tModelInfo elements. It is returned in response to a get_registeredInfo message.
- **relatedBusinessesList**: This element reports publicly visible business relationships. It is returned in response to a find_relatedBusinesses message. Business relationships are visible between two businessEntity registrations when there are complete publisher assertions that verify that the publishers who control each of the businessEntity elements involved in a relationship agree that both businesses are involved.
- **serviceDetail**: This element contains full details for zero or more businessService elements. It is returned in response to a get_serviceDetail message, and optionally in response to the save_binding and save_service messages.
- **serviceList**: This element contains abbreviated information about registered businessService information. This message contains zero or more serviceInfo elements. It is returned in response to a find_service message.
- **tModelDetail**: This element contains full details for zero or more tModel elements. It is returned in response to a get_tModelDetail message, and optionally in response to the

save_tModel message.

- **tModelList**: This element contains abbreviated information about registered tModel information. This message contains zero or more tModelInfo elements. It is returned in response to a find_tModel message.

11 Appendix G: redirection via hostingRedirector element

One of the main benefits of using a public Operator Site instance of a UDDI registry is to provide a single point of reference for determining the correct location to send a business service request to a remote web service. In general, the controller of a particular instance of bindingTemplate element can be assured that by keeping the registered copy pointing to the proper server or invocation address, special conditions such as disaster recovery to a secondary site can be handled with a minimum of service disruption for customers or partners. The same holds true for those who choose to use a registry that is compatible with the API, but to a lesser degree.

In many cases, the API specified in the get_bindingDetail message is straightforward. Once a business or application knows of a service that needs to be invoked, the UDDI bindingTemplate information that represents this service can be cached until needed. In the event that the cached information fails at the time the partner web service is actually invoked (i.e., the accessPoint information in the cached bindingTemplate element is used to invoke a remote partner service), the application can use the bindingKey in the cached information to get a fresh copy of the bindingTemplate information. This cached approach serves to prevent needless round trips to the registry.

11.1 Special situations requiring the hostingRedirector

Two special needs arise that cannot be directly supported by the *accessPoint* information in a bindingTemplate. These are:

- **Third Party Hosting of Technical Web Services**: A business chooses to expose a service that is actually hosted at a remote or third party site. Application Service Providers (ASP) and Network Market Makers are common examples of this situation. In this situation, it is the actual third party that needs to control the actual value of the binding information.
- **Use-specific access control to binding location**: In other situations, such as situation specific redirection based on the identity of the caller, or even time-of-day routing, it is necessary to provide the actual contact point information for the remote service in a more dynamic way than the cached accessPoint data would support.

For these cases, the bindingTemplate elements contain an alternative data element called *hostingRedirector*. The presence of a hostingRedirector element specifies that the caller that wants to invoke a web service represented by a specific bindingTemplate must take an additional step to get the accessPoint information (e.g. the invocation address). There are two possible actions required to resolve a hostingRedirector reference.

11.2 Using the hostingRedirector data

When a bindingTemplate that represents a service entry (called the service binding) contains a hostingRedirector element, the programmer uses this information to locate the actual bindingTemplate that contains the accessPoint for the "hosted" service. The content of the hostingRedirector element is a bindingKey reference that refers to another bindingTemplate. This bindingKey is resolvable in the same UDDI registry where the hostingRedirector is found via the get_bindingDetail message.

After accessing this second binding (called the indirect binding), the caller looks at the *fingerprint*[22] of the indirect binding. If the fingerprint in the indirect binding is the same as the fingerprint found in the service binding, then the accessPoint on the indirect binding should be used to contact the web service. This use of hostingRedirector is called *simple indirection*. If however, the fingerprint of the indirect

[22] The term *fingerprint* refers to the set of tModelKey values found in the tModelInstanceInfos collection of the bindingTemplate. Fingerprints of two bindings are equivalent when both of the sets of tModelKey values are the same.

binding signifies that the indirect binding implements a *hostingRedirector Service*[23], then a second call to get_bindingDetail, sent to the accessPoint address found in the indirect binding will be required to get the true service binding. This use of hostingRedirector is called *redirection*.

11.2.1 Stepwise overview

In the simple case of simple indirection, the hosted bindingTemplate is simply referenced by the original service binding. For this case, there is no complex logic.

For the redirection case, the following scenario walkthrough describes the steps.

1. A business registers a bindingTemplate ***A*** for a remotely hosted or redirected business service ***S***. This bindingTemplate contains a bindingKey value ***Q*** that references a second bindingTemplate ***B***. The bindingTemplate ***B*** is typically controlled by the organization that hosts the redirection service. This bindingTemplate ***B*** contains an accessPoint element that points to the actual hostingRedirector service ***R***.

2. A program that wants to call the service ***S*** that is registered in step one gets the binding information for the advertised service. This bindingTemplate information contains a hostingRedirector element with the bindingKey ***K*** for the bindingTemplate ***B***.

3. The program takes the bindingKey ***K*** and issues a get_bindingDetail message against the registry that the original bindingTemplate ***A*** came from. This returns the data for bindingTemplate ***B***. The programmer now has the address of the service that implements the redirection. This information is in the accessPoint element found in bindingTemplate ***B***. This service, to be compliant, knows how to respond to a get_bindingDetail message.

4. Using the original binding key ***Q*** the program issues a get_bindingDetail request to the redirector service ***R***. This service is responsible for returning the actual binding information for the redirected business service ***S*** or returning an error. The program has the choice of caching this bindingTemplate if desired.

Using this algorithm, an organization that hosts services for other businesses to use can control the information that is used to actually access the hosted service. This not only provides this hosting organization with the ability to manage situations such as disaster recovery locations, but also lets them specify the actual URL that is used to make a call to the actual business service. This URL can be keyed specifically to the caller, or can be a general location for the hosted or redirected service.

In any case, the original caller is able to find the technical web service (bindingTemplate) advertised within the actual business partner's data without having to know that any redirection occurred.

[23] A hosting redirector service is an interface defined by UDDI. This is a service that implements only the UDDI get_bindingDetail message. When successfully invoked, using the original service binding key, the actual service binding containing the service accessPoint address will be returned.

12 Appendix H: Checking external value-sets

Whenever save_business, save_service or save_tModel are called, the contents of any included *categoryBag* or *identifierBag* element may be checked to see that it contains valid values. Checking is performed for taxonomies and identifier schemes that are classified as "checked". UDDI version 2 provides the ability for third parties to register new taxonomies and identifier schemes, and then control the validation process used by UDDI to perform such checks.

Third parties that want to provision such a capability must implement a web service in the same manner that UDDI does (e.g. using SOAP 1.1 over HTTP for message passing) that exposes a single method named validate_values. The interface for validate_values is described here.

12.1 validate_values

A UDDI operator sends the validate_values message to the appropriate external service, of which there is exactly one, whenever a publisher saves data that uses a categorization value or identifier whose use is regulated by the external party who controls that service. For purposes of discussion, these identifiers and classifications are called *checked value sets*. The normal use is to verify that specific categories or identifiers (checking the keyValue attribute values supplied) exist within the given taxonomy or identifier system. For certain categorizations and identifiers, the party providing the validation service may further restrict the use of a value to certain parties based on the identifiers passed in the message or any other type of contextual check that is possible using the passed data.

12.1.1.1 Syntax:

```
<validate_values generic="2.0" xmlns=" urn:uddi-org:api_v2">
   <businessEntity/>... | <businessService/>... | <tModel/>...
<validate_values>
```

12.1.1.2 Arguments:

The Operator that is calling validate_values will pass a businessEntity, a businessService, or a tModel element as the sole argument to this call-out. This is the same data that is being passed within a save_business, save_service, or save_tModel API call. Multiple elements of the same type may be passed together.

12.1.1.3 Behavior

The called service will perform any validation steps desired by the validator. This can involve merely checking that the *keyValue* values supplied are good for the given value set (as signified by the embedded keyedReference tModelKey values). Other types of validation as desired can be performed, including context sensitive checks that utilize the businessKey or other values passed in the call-out.

If no error is found, the proper response is a dispositionReport message as specified in Appendix A. The error code value returned should be "E_success" and the errno value in the result should be "0".

12.1.1.4 Caveats:

If any error is found, or the called service needs to signal that the information being saved is not valid based on the validation algorithm chosen by the external service provider, then the service should raise a SOAP Fault as specified in Appendix A.

When an error is signaled in this fashion, UDDI will reject the pending change and return to the original caller the same SOAP fault data returned by the validation Web Service. The error codes should indicate one of the following reasons, and the error text should clearly indicate the keyedReference

data that is being rejected and the reason it is being rejected.

- **E_invalidValue**: One or more of the keyValue values supplied failed validation. Only the first error need be reported.
- **E_valueNotAllowed**: The values may be valid, but are not allowed contextually.

13 Appendix I: Utility tModels and Conventions

In order to facilitate consistency in Service Description (tModel) registration, and provide a framework for their basic organization within the UDDI registry, a set of conventions was established in UDDI version 1.0. Version 2.0 continues this by defining further conventions. This section describes these *canonical conventions*, as well as a set of canonical tModels that facilitate registration of common information and the services provided by the UDDI registry itself.

13.1 Canonical tModel entities

13.1.1 UDDI Registry tModels

The UDDI registry defines a number of tModels to define its core services. Each of the core tModels are listed in this section.

13.1.1.1 uddi-org:inquiry[24]

tModel Description: UDDI Inquiry API - Core Specification
tModel UUID: `uuid:4CD7E4BC-648B-426D-9936-443EAAC8AE23`
Categorization: `specification, xmlSpec, soapSpec`

This tModel defines the inquiry API calls for interacting with the UDDI registry.

13.1.1.2 uddi-org:inquiry_v2

tModel Description: UDDI Inquiry API V 2.0- Core Specification
tModel UUID: `uuid:AC104DCC-D623-452F-88A7-F8ACD94D9B2B`
Categorization: `specification, xmlSpec, soapSpec`

This tModel defines the inquiry API calls for interacting with the UDDI registry.

13.1.1.3 uddi-org:publication

tModel Description: UDDI Publication API - Core Specification
tModel UUID: `uuid:64C756D1-3374-4E00-AE83-EE12E38FAE63`
Categorization: `specification, xmlSpec, soapSpec`

This tModel defines the publication API calls for interacting with the UDDI registry.

13.1.1.4 uddi-org:publication_v2

tModel Description: UDDI Publication API V2.0 - Core Specification
tModel UUID: `uuid:A2F36B65-2D66-4088-ABC7-914D0E05EB9E`
Categorization: `specification, xmlSpec, soapSpec`

This tModel defines the publication API calls for interacting with the UDDI registry.

13.1.1.5 uddi-org:taxonomy

tModel Description: UDDI Taxonomy API
tModel UUID: `uuid:3FB66FB7-5FC3-462F-A351-C140D9BD8304`
Categorization: `specification, xmlSpec, soapSpec`

This tModel defines the validate_values API call used for checked taxonomy and identifier system validation.

[24] The title of each tModel description is also the <name/> of the tModel.

13.1.1.6 uddi-org:taxonomy_v2

tModel Description: UDDI validate_values API
tModel UUID: `uuid:1E3E9CBC-F8CE-41AB-8F99-88326BAD324A`
Categorization: `specification, xmlSpec, soapSpec`

This tModel defines the validate_values API call made by UDDI registries to validate external checked taxonomies. See Appendix H.

13.1.2 UDDI Core tModels - built-in taxonomies, identifier systems, and relationships

An additional set of tModels has been established to assist in identification and categorization, for example within industry taxonomies. Their tModels are described below.

13.1.2.1 uddi-org:types

The UDDI specifications provide a great deal of flexibility in terms of the types of information that may be registered. To help in getting started, a taxonomy named uddi-org:types has been established to assist in general categorization of the tModels themselves.

The approach to categorization of tModels within the UDDI Type Taxonomy is consistent with that used for each of the other taxonomies. The categorization information for each tModel is added to the **`<categoryBag>`** elements in a **save_tModel** message. A **`<keyedReference>`** element is added to the category bag to indicate the type of tModel that is being registered.

The values used for keyed references are defined in the UDDI Type Taxonomy shown in the tModel description table below.

tModel Description: UDDI Type Taxonomy
tModel UUID: `uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4`
Categorization: `categorization`
Checked: Yes

13.1.2.1.1 Taxonomy Values

The following list of values are defined in the uddi-org:types taxonomy. These values are useful for classifying tModels and are helpful to others who want to find tModel data of a particular type.

tModel: The UDDI type taxonomy is structured to allow for categorization of registry entries other than tModels. This key is the root of the branch of the taxonomy that is intended for use in categorization of tModels within the UDDI registry. Categorization is not allowed with this key.

identifier: An identifier tModel represents a specific set of values used to uniquely identify information. For example, a Dun & Bradstreet D-U-N-S® Number uniquely identifies companies globally. The D-U-N-S® Number taxonomy is an identifier taxonomy.

namespace: A namespace tModel represents a scoping constraint or domain for a set of information. In contrast to an identifier, a namespace does not have a predefined set of values within the domain, but acts to avoid collisions. It is similar to the namespace functionality used for XML.

categorization: A categorization tModel is used for information taxonomies within the UDDI registry. NAICS and UNSPSC are examples of categorization tModels.

postalAddress: A postalAddress tModel is used to identify different forms of postal address within the UDDI registry. postalAddress tModels may be used with the address element to distinguish different forms of postal address.

relationship: A relationship tModel is used for relationship categorizations within the UDDI registry. relationship tModels are typically used in connection with publisher relationship assertions.

specification: A specification tModel is used for tModels that define interactions with a Web Service. These interactions typically include the definition of the set of requests and responses, or other types of interaction, that are prescribed by the service. tModels describing XML, COM, CORBA, or any other

services are specification tModels.

xmlSpec: An xmlSpec tModel is a refinement of the specification tModel type. It is used to indicate that the interaction with the service is via XML. The UDDI API tModels are xmlSpec tModels.

soapSpec: Further refining the xmlSpec tModel type, a soapSpec is used to indicate that the interaction with the service is via SOAP. The UDDI API tModels are soapSpec tModels, in addition to xmlSpec tModels.

wsdlSpec: A tModel for a Web Service described using WSDL is categorized as a wsdlSpec.

protocol: A tModel describing a protocol of any sort.

transport: A transport tModel is a specific type of protocol. HTTP, FTP, and SMTP are types of transport tModels.

signatureComponent: A signature component is used to for cases where a single tModel can not represent a complete specification for a Web Service. This is the case for specifications like RosettaNet, where implementation requires the composition of three tModels to be complete - a general tModel indicating RNIF, one for the specific PIP, and one for the error handling services. Each of these tModels would be of type signature component, in addition to any others as appropriate.

unvalidatable: Marking a tModel with this classification prevents references to that tModel in any keyedReference contained within an identifierBag or categoryBag element. Used as part of the process required to establish a new checked value set and to retire a checked value set.

checked: Marking a tModel with this classification asserts that it represents a categorization, identifier, or namespace tModel that has a properly registered validation service per the UDDI Version 2.0 Operators Specification Appendix A.

unchecked: Marking a tModel with this classification asserts that it represents a categorization, identifier, or namespace tModel that does not have a validation service.

13.1.2.2 ntis-gov:naics:1997

tModel Description:	Business Taxonomy: NAICS (1997 Release)
tModel UUID:	`uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2`
Categorization:	`categorization`
Checked:	Yes

This tModel defines the NAICS industry taxonomy.

13.1.2.3 unspsc-org:unspsc:3-1

tModel Description:	Product Taxonomy: UNSPSC (Version 3.1)
tModel UUID:	`uuid:DB77450D-9FA8-45D4-A7BC-04411D14E384`
Categorization:	`categorization`
Checked:	No

This tModel defines the UNSPSC product taxonomy. This taxonomy is deprecated. Users are advised to use the unspsc-org:unspsc taxonomy instead

13.1.2.4 unspsc-org:unspsc

tModel Description:	Product and Services Taxonomy: UNSPSC (Version 7)
tModel UUID:	`uuid: CD153257-086A-4237-B336-6BDCBDCC6634`
Categorization:	`categorization`
Checked:	Yes

This tModel defines the UNSPSC product and services taxonomy for Version 7 and beyond. It replaces the deprecated taxonomy unspsc-org:unspsc:3-1.

13.1.2.5 uddi-org:`iso-ch:3166:1999`

tModel Description:	UDDI Geographic Taxonomy

tModel UUID: `uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88`
Categorization: `categorization`
Checked: Yes

This tModel defines the ISO 3166 geographic classification taxonomy namespace.

13.1.2.6 uddi-org:general_keywords

tModel Description: Other Taxonomy
tModel UUID: `uuid:A035A07C-F362-44dd-8F95-E2B134BF43B4`
Categorization: `categorization`
Checked: Yes

This tModel defines generic name / value pairs. In a keyedReference involving this taxonomy, the keyName constitutes the name and the keyValue constitutes the value. In contrast to keyedReferences involving other taxonomies, those using uddi-org:general_keywords require that the keyName be provided in addition to the keyValue. Checking for this taxonomy consists of ensuring that keyName is not omitted or specified as the zero-length string; UDDI registries MUST fail save operations that contain keyedReferences that involve uddi-org:general_keywords and that do not specify a non-empty keyName.

The uddi-org:general_keywords tModel was renamed for V2; the V1 name was uddi-org:misc-taxonomy. The tModel UUID is still the same.

13.1.2.7 uddi-org:owningBusiness

tModel Description: A pointer to a businessEntity that owns the tagged data.
tModel UUID: `uuid:4064C064-6D14-4F35-8953-9652106476A9`
Categorization: `categorization`
Checked: Yes

The value set of this taxonomy is the set of businessKeys. It is used to specify that the businessEntity whose businessKey is the keyValue in a keyedReference “owns” the tagged tModel. The entity tagged must be a tModel, the referred-to businessEntity must exist, and it must have been published by the publisher that publishes the tagged information.

13.1.2.8 uddi-org:relationships

tModel Description: UDDI businessEntity relationship descriptions
tModel UUID: `uuid:807A2C6A-EE22-470D-ADC7-E0424A337C03`
Categorization: `relationship`
Checked: No

This tModel is used to describe business relationships. Used in the publisher assertion messages.

13.1.2.8.1 Relationship values

parent-child Used to indicate that the businessEntity referred to by an assertion fromKey is the parent (e.g., holding company) of the businessEntity referred to by the toKey (e.g., subsidiary).

peer-peer Used to indicate that the businessEntity referred to by an assertion fromKey is a peer of the businessEntity referred to by the toKey.

identity Used to indicate that the businessEntity referred to by an assertion fromKey represents the same organization as the businessEntity referred to by the toKey.

13.1.2.9 uddi-org:operators

tModel Description: Taxonomy for categorizing the businessEntity of an operator of a registry
tModel UUID: `uuid:327A56F0-3299-4461-BC23-5CD513E95C55`
Categorization: `categorization`
Checked: Yes

This checked value set is used to categorize the businessEntity of a UDDI operator. Each operator of a registry typically publishes a businessEntity, called the "operator businessEntity" to represent the services the operator offers in connection with the registry. Each such businessEntity SHOULD be categorized with the uddi-org:operators taxonomy.

The only valid value in this taxonomy is "node". An operator site MUST NOT allow a businessEntity other than its operator businessEntity to be categorized using this taxonomy.

13.1.2.10 dnb-com:D-U-N-S

tModel Description: Dun & Bradstreet D-U-N-S® Number
tModel UUID: `uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823`
Categorization: `identifier`
Checked: No

This tModel is used for the Dun & Bradstreet D-U-N-S® Number identifier. Note that this tModel is initially registered as part of the UDDI core tModels. Once the registry is in production, management of this tModel is expected to be transferred to the Dun & Bradstreet publisher account. For more information, see http://www.dnb.com.

13.1.2.11 thomasregister-com:supplierID

tModel Description: Thomas Registry Suppliers
tModel UUID: `uuid:B1B1BAF5-2329-43E6-AE13-BA8E97195039`
Categorization: `identifier`
Checked: No

This tModel is used for the Thomas Register supplier identifier codes. Note that this tModel is initially registered as part of the UDDI core tModels. Once the registry is in production, custody of this tModel is expected to be transferred to the Thomas Register publisher account. For more information, see http://www.thomasregister.com.

13.1.2.12 uddi-org:isReplacedBy

tModel Description: Identifier system for indicating replacement entities
tModel UUID: `uuid:E59AE320-77A5-11D5-B898-0004AC49CC1E`
Categorization: `identifier`
Checked: yes

An identifier system used to point (using UDDI keys) to the tModel (or businessEntity) that is the logical replacement for the one in which isReplacedBy is used.

13.1.2.12.1 Taxonomy Values

The keyValues in keyedReferences that refer to this tModel must be tModelKeys or businessKeys. Such a keyValue specifies the entity that is the replacement for the entity in which the keyedReference appears.

13.1.3 UDDI Core tModels – Other

Additional tModels are defined to help register within leading industry encoding schemes and standard protocols. This list is expected to be expanded as appropriate as the UDDI business registry expands.

13.1.3.1 uddi-org:smtp

tModel Description: E-mail based web service
tModel UUID: `uuid:93335D49-3EFB-48A0-ACEA-EA102B60DDC6`

Categorization: `transport`

This tModel is used to describe a web service that is invoked through SMTP email transmissions. These transmissions may be either between people or applications.

13.1.3.2 uddi-org:fax

tModel Description: Fax based web service
tModel UUID: `uuid:1A2B00BE-6E2C-42F5-875B-56F32686E0E7`
Categorization: `protocol`

This tModel is used to describe a web service that is invoked through fax transmissions. These transmissions may be either between people or applications.

13.1.3.3 uddi-org:ftp

tModel Description: File transfer protocol (ftp) based web service
tModel UUID: `uuid:5FCF5CD0-629A-4C50-8B16-F94E9CF2A674`
Categorization: `transport`

This tModel is used to describe a web service that is invoked through file transfers via the ftp protocol.

13.1.3.4 uddi-org:telephone

tModel Description: Telephone based web service
tModel UUID: `uuid:38E12427-5536-4260-A6F9-B5B530E63A07`
Categorization: `specification`

This tModel is used to describe a web service that is invoked through a telephone call and interaction by voice and/or touch-tone.

13.1.3.5 uddi-org:http

tModel Description: An http or web browser based web service
tModel UUID: `uuid:68DE9E80-AD09-469D-8A37-088422BFBC36`
Categorization: `transport`

This tModel is used to describe a web service that is invoked through a web browser and/or the http protocol.

13.1.3.6 uddi-org:homepage

tModel Description: HTTP Web Home Page URL
tModel UUID: `uuid:4CEC1CEF-1F68-4B23-8CB7-8BAA763AEB89`
Categorization: `specification`

This tModel has been deprecated. It was used as the bindingTemplate fingerprint for a web home page reference. It is recommended that, instead of creating a bindingTemplate using this tModel as the technical fingerprint, publishers use the discoveryURL with a useType of “homepage”.

13.2 Registering tModels within the Type Taxonomy

When a new tModel is registered within UDDI, its type can be classified within the framework of the UDDI Type Taxonomy. This classification provides additional hints to applications for what type of tModel is being registered. For each appropriate classification, a keyed reference is added to the category bag element for the tModel.

As an example, the Dun & Bradstreet D-U-N-S® Number is a type of identifier for an organization. Within the UDDI type taxonomy, the `dnb-com:D-U-N-S` tModel is classified as type identifier.

The categoryBag element of the tModel registered would be as follows:

```
<categoryBag>
 <keyedReference
   tModelKey = "uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
```

```
   keyValue = "identifier"
   keyName = "tModel is a unique identifier" />
</categoryBag>
```

- **tModelKey:** This is the GUID for the UDDI Types taxonomy. It is required.
- **keyValue:** This is the identifier for the categorization within the UDDI Types taxonomy. It is required.
- **keyName:** This is the description of the identifier within the UDDI Types taxonomy. It is not required as a part of the registration, but simply provides additional information about the key selected.

14 Appendix J: Relationships and publisher assertions

UDDI version 2.0 introduces an assertion feature based on “publisher assertions”. Publisher assertions are the basis for a mechanism to allow more than one registered businessEntity elements to be linked in a manner that conveys a specific type of relationship. This is why the feature is sometimes called the relationship feature. Publisher assertions are used to establish visible relationships between registered data in a way that once completed, a set of assertions can be seen by the general inquiry message named “find_relatedBusinesses”.

The design goals around allowing businesses to describe relationship between different parts involved accommodating the needs of larger businesses to express their UDDI data in more than one part. After all, large companies are composed of many smaller parts, and within those individual business units there are various web services that need describing. Prior to UDDI version 2.0, businesses that wanted to model a very complex business were not able to do so.

In order to make it possible for either party in a relationship to have some control over the visibility of the relationship, another design goal was to make sure that visible relationships were only visible if both parties involved in a potential relationship expression actually agreed to the fact that such a relationship exists at a given point in time. The problematic scenario that was addressed was one where a party falsely claims that the businessEntity data that it registered were part of a large company, resulting in damages to the non-consenting company. Readers of the data could also be misled to believe that a business registration was part of another company.

The design adopted involves a requirement for the publishers that control the businesses that are related to assert in a symmetric fashion that the relationship exists. In the case where a different individual controls each businessEntity involved in such an expression, both parties would need to assert the same fact about a specific relationship before UDDI will surface any information about such a relationship. In cases where two parties are involved and both parties do not agree as to the details of a given assertion, there is no requirement for either party to complete an assertion. No relationship will be exposed via the Inquiry API in this case.

14.1.1 Example

The following picture shows the start of an assertion process.:

Joe and Xina each manage a businessEntity within UDDI. As we start our scenario, both Joe and Xina have registered a businessEntity at their favorite UDDI operator site. Joe and Xina wish to make it possible for users of UDDI to find out that the two businesses are in fact part of the same business, with business1 being the parent-business.

To make the relationship visible for anyone who calls find_relatedBusinesses passing either businessKey as a starting point, Joe and Xina must both make a publisher assertion. As the name of the operation suggests, a publisher assertion is an assertion made by a publisher who is expressing a particular fact about a business registration and its relationships to other business data within UDDI.

The process, Joe uses a program that helps him manage his UDDI data and makes a new publisher assertion about business1 and business2, expressing the fact that business 1 is the corporate holding company. This fact is expressed by Joe's software which sends a add_publisherAssertions message to UDDI. This message looks like:

```
<add_publisherAssertions generic="2.0" xmlns="urn:uddi-org:api_v2" >
    <authInfo>FFFFF</authInfo>
    <publisherAssertion>
        <fromKey>BK1</fromKey>
        <toKey>BK2</toKey>
        <keyedReference
            tModelKey=" uuid:807A2C6A-EE22-470D-ADC7-E0424A337C03" keyName="Holding
            Company" keyValue="parent-child" />
    </publisherAssertion>
</add_publisherAssertions>
```

In this example, we see that Joe asserts that the businessEntity with the businessKey value of BK1 is the parent holding company of the businessEntity with the businessKey value of BK2.

Because only Joe has asserted this fact, the information about the relationship is not yet visible via the inquiry API call, find_relatedBusinesses. Joe knows that for this assertion to become visible, the publisher of businessEntity BK2 must also express the same assertion. Joe calls Xina to let her know he wants her to make the assertion.

In order to see the data that she must express, Xina sends a get_assertionStatusReport to her UDDI operator site. From the resulting assertionStatusReport, Xina sees that there is indeed an unmatched assertion listed against her businessEntity B2. Since Joe has contacted her and she agrees that the relationship should be visible within UDDI, Xina sends the exact same assertion (with a different authInfo credential) to her UDDI operator site.

The UDDI operator sites now see the assertions made by the two publishers, each of whom control one of the two businesses involved. After checking that the requesting parties each control half of the relationship, UDDI matches the assertions together and the status of the relationship becomes complete.

After this is done, anyone who calls the Inquiry API call, find_relatedBusinesses, and passes either BK1 or BK2 as the businessKey value will see the relationship. Prior to both publishers asserting this same fact, the data about the relationship would not be visible via the Inquiry API.

14.1.2 Managing relationship visibility

The publishers API defines several messages to allow assertions to be managed by UDDI publishers. These messages fall into two general categories: administrative helpers and maintenance functions. The administrative helpers allow the publisher to see assertions that their businesses are involved in. In particular, the get_assertionStatusReport is the most useful for determining whether any assertions involving your business registrations are incomplete. This allows you to not only see work that may need to be done, but it also allows you to find out if others are trying to make assertions about your business that you may not either know about or agree with.

The maintenance functions let you either deal with all assertions as a single group (e.g. get_publisherAssertions / set_publisherAssertions) or individually (e.g. add_publisherAssertions / delete_publisherAssertions). The latter are handy if you want to add one at a time without having to keep track of all assertions you have made in the past.

One word of caution: set_publisherAssertions can be used to invalidate all assertions by removing all assertions in a single call.

15 References

This section contains URL pointers to various specifications and other documents that are pertinent in understanding this specification.

W3C specifications, notes and drafts

- XML 1.0
- XML Schema
- XML namespaces
- SOAP 1.1

UDDI specifications, white-papers and schemas

- UDDI Version 2.0 API schema
  (http://www.uddi.org/schema/uddi_v2.xsd)
- UDDI Version 2.0 Operators specification
  (http://www.uddi.org/pubs/Operators-V2.00-Open-20010608.doc)
- UDDI.org
  (http://www.uddi.org/)
- UDDI 2.0 Data Structure Reference
  (http://www.uddi.org/pubs/DataStructure-V2.00-Open-20010608.doc)

16 Change History

23 February, 2001: Dan Rogers initial pass at updating this for V2. Many of the appendices are not updated yet. For working group review. During this pass, corrected many of the overlooked V1 errata – as discussed and observable in the common operator implementations. Still have to fold in an appendix on internationalization, and update the schema to reflect the final face to face agreements around address, value validation, and ??? other stuff ??? I'll be working on this through the weekend as well.

26 February, 2001: Dan Rogers, Dave Ehnebuske. Review, incorporate comments, complete appendices (first cut) during the authors meeting held in Redmond in building 118.

20 March 2001: Dan Rogers: set document to change tracking for round 1 review cycle changes.

17 April 2001: Dan Rogers: apply feedback/edits from team. Published Rev 3 review draft as release candidate 1

31 May 2001: David Ehnebuske. Incorporate changes based on input from Advisors Group review cycle. As discussed on the http://groups.yahoo.com/group/uddi-v2api-review mailing list. Changes are: Clarification of which search arguments may be passed in the find_business and find_service messages, addition of "checked" and "unchecked" categories to uddi-org:types, miscellaneous wording changes for clarification and to correct oversights, and typo fix-ups.

16 July 2001: David Ehnebuske. Added changes for Errata 1:

- Change version to 2.01.
- Clarify when keyedReferences in find_xx APIs match ones in the registry.
- Change title of section 4.3.1. Clarify what happens to references to bindingTemplates that are deleted by invocation of delete_binding.
- Clarify behavior of delete_tModel.
- Clarify behavior of save_business with respect to service projections.
- Specify the error code returned when the content of a service projection in a save_business does not match the service stored in the registry.
- Specify that all of the elements in a publisherAssertion must be specified.
- Clarify that a publisher must control at least one of the businessEntities referred to in a set_publisherAssertions message.
- Fix duplicate error code number by assigning code 30110 instead of 30100 to E_messageTooLarge. Add error code E_unvalidatable (20220) to list of error codes.
- Clarify the meaning of the orAllKeys and andAllKeys search qualifiers.
- Correct the description of the uddi-org:taxonomy and uddi-org:general_keywords tModels. Added uddi-org:isReplacedBy identifier system tModel.
- Fix various typos and formatting glitches

18 September 2001: David Ehnebuske. Added changes for Errata 2:

- Change version to 2.02.
- Add missing error code for E_requestTimeout: (20240)
- Clarify what find_relatedBusiness returns when keyedReference specified
- Correct key for uddi-org:iso-ch:3166:1999 tModel
- Change XML encoding requirement to conform with RFC 3023
- Clarify effect of Version 1 save_xx in Version 2 UDDI Registries
- Clarify behavior when saving the same entity multiple times in one save_xx message
- Make it clear that uddi-org:misc-taxonomy is called uddi-org:general_keywords in V2
- Add missing maxRows attribute on find_relatedBusinesses
- Clarify behavior with respect to save_xx and the xml:lang attributes

2 February 2002: David Ehnebuske. Added changes for Errata 3:

- Number of <name> elements allowed in find_business and find_service
- Cross-business find_service
- Deleting an already deleted tModel
- Behavior of Service Projections
- UNSPSC Upgrade
- Incorrect text associated with E_invalidValues
- V2 error codes and with V1 messages
- Support for SOAPAction
- No V2 findQualifiers in V1 messages
- Soundex findQualifier removed
- No claims of SOAP encodingStyle allowed
- Comply with SOAP 1.1 Specification when rejecting SOAP Actor
- SOAP <faultcode/> “Server” allowed
- Applicability of findQualifiers made explicit
- Precedence of default findQualifiers made explicit
- E_invalidKeyPassed missing from find_business
- Clarify orLikeKeys applicability
- Use of E_unsupported in find_service inconsistent with other find_xx APIs
- Clarify validation requirements for uddi-org:owning-Business
- Empty <categoryBag/> and <identifierBag/> elements not allowed
- Duplicate keys in delete_binding, delete_business, delete_publisherAssertion, and delete_service
- Empty <contacts/> elements not allowed
- Either <accessPoint/> or <hostingRedirector/> is required
- Elements in inquiry messages are subject to truncation
- Typos and formatting glitches