UDDI Version 2.00 API Specification

Errata 4

05 July 2002

This version:

2.04.3

Editor:

David Ehnebuske, IBM

Contents

Introduction

This document contains the errata to the Open Draft of the UDDI V2.00 API Specification. The errors and requests for clarification covered by these Errata were brought to the editors' attention during the review of the Open Draft of that document. Once the review period is completed and the specification becomes final, the changes given below will be factored into the final document.

As additional errors and requests for clarification are dealt with during the review period, additional editions of this document may become necessary. Because of this, users of the Open Draft API Specification should verify that they have the latest version of the Errata.

Errata 1

When keyedReferences in find_xx APIs match ones in the registry

The spec is unclear what, precisely is meant by a "match" of the keyedReferences that are passed in the various find_xx APIs with those in registry entries.

For identifierBags

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.

For categoryBags

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.

For publisherAssertions

A match for publisherAssertions occurs 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.

Title of section 4.3.1

The title of section 4.3.1 is somewhat misleading. It has been changed to "Rationale for UDDI version 2.0 Publishing API Enhancements."

References to deleted bindingTemplates

It is not clear in the API specification what, if anything happens to references to a bindingTemplate that is deleted. The answer is that such references cease to point to anything meaningful but are not changed. Add the following to section 4.4.2.3 ("Returns" section of "delete_binding"):

"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."

Behavior of delete_tModel

The action of a delete_tModel API call is to render the target tModel “invisible” to find_tModel calls. It is still possible to use the get_tModelDetail to retrieve tModels which have been “hidden” in this way. The text of the specification makes it sound as if there is a “hidden” flag in tModels that controls this behavior. There is not. (While implementations may have such a thing, it’s not a part of the UDDI data model.) The text has been corrected to reflect this. It now reads:

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.

Behavior of save_business with respect to service projections

Section 4.4.13.3 discusses the behavior of the save_business API. Among other things it talks about what happens when a businessEntity is saved that has the side-effect of deleting a businessService that some other businessEntity is using as a service projection. The effect is that such service projections are automatically removed. The affected paragraph has been changed to read:

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 will be automatically deleted. 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.

Error code for invalid service projection in save_business

In section 4.4.13.3 the text of the specification inadvertently omits an error code. The error code is the one returned when a save_business is done using a service projection and the service projection is different than the service being projected upon. The correct error code is E_invalidProjection. The text has been corrected to read:

When saving a businessEntity containing a service reference, the content of the businessService provided in the save_business must either match the content of the referenced businessService or contain only the name of the businessService being referenced. If the businessService provides does not meet these requirements the save will fail with the error E_invalidProjection.

All elements in a publisherAssertion must be specified

The text of the specification is not clear that all elements of a publisher assertion must be specified. The text of section 4.4.16.2 has been modified to read:

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. 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.

Publisher must control at least one referred-to businessEntity

The text of section 4.4.16.4 (the Results section of set_publisherAssertions) is not clear that to succeed, the publisher must own at least one of the businessKeys specified in the API call. The text has been corrected to read:

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.

Duplicate error code number

The specification erroneously uses the numerical value 30100 for two different error codes, E_invalidCompletionStatus and E_messageTooLarge. The correct value for E_messageTooLarge is 30110.

Missing error code E_unvalidatable

The specification incorrectly omits the error code E_unvalidatable, numerical value 20220, from the list of valid error codes. This has been corrected.

Meaning of orAllKeys and andAllKeys search qualifiers

Section 9.1.1 of the text of the specification does not make the meaning of the search qualifiers orAllKeys and andAllKeys clear. The text has been corrected to read:

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.

And

andAllKeys: this changes the behavior for identifierBag to AND keys rather than OR them.

Wrong description of the uddi-org:taxonomy and uddi-org:general_keywords

Section 13.1.1.5, the description of the uddi-org:Taxonomy tModel is incorrect. This tModel defines the validate_values API. The correct description is:

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

Section 13.1.2.5 incorrectly describes the uddi-org:general_keywords taxonomy. The correct description is:

This tModel defines generic name / value pairs.

Missing uddi-org:isReplacedBy tModel

The uddi-org:isReplacedBy tModel was inadvertently omitted from the specification. A new section – 13.1.2.11 – has been added to describe it. Its description is:

Name: uddi-org:isReplacedBy

Description: 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.

tModel UUID: uuid:E59AE320-77A5-11D5-B898-0004AC49CC1E

Categorization: identifier

Checked: Yes

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.

Typos and formatting glitches

A number of typographical errors and formatting problems have been reported and fixed during the open specification review. None change the meaning of the text.

Errata 2

Missing error code

As the API Specification is written it is not possible for a user to distinguish between a save_xx that fails because a required validate_values service fails to work correctly and one that fails because a value failed validation. To distinguish these cases, a new error code is introduced:

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.

In addition, save_business, save_service, and save_tModel are updated to indicate they can return this error code:

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.

Unclear what find_relatedBusiness returns when keyedReference specified

The API call find_relatedBusiness can take a keyedReference as an optional argument. It is unclear from the text of the specification what, exactly, is included in the result when this is specified. The text is changed to make it clear that the keyedReference is used to select which businesses are contained in the result and that when a business is selected, the result contains all of the shared relationships between the selected business and the focal point business. In particular the text describing the optional keyedReference is modified as follows:

- **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.

Wrong key for uddi-org:iso-ch:3166:1999 tModel

The specification gives the incorrect tModel key for uddi-org:iso-ch:3166:1999. The correct key is `uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88`. The text of section 13.1.2.4 is corrected to reflect this change.

Conformance with RFC 3023

The text of section 7.2 (XML Encoding Requirement) does not conform to RFC 3023 (XML Media Types). In particular, while the specification requires UDDI messages to be encoded in UTF-8, it allows them to omit the charset parameter in the Content-Type HTTP header. RFC 3023 says that when this parameter is omitted, the default is “us-ascii”. Since UDDI requires UTF-8, we should require an explicit confirmation of that in the Content-Type HTTP header. The text of section 7.2 of the UDDI API spec is altered to read:

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.

Effect of Version 1 save_xx in Version 2 UDDI Registries

The text of the specification does not spell out how Version 1 and Version 2 saves interact with one another in a Version 2 registry. To remedy this, a new section “Effect of Version 1 save_xxx in Version 2 UDDI Registries” has been added just before section 4.3.4 “Special considerations for validated namespaces”. This section says:

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.

Saving the same entity multiple times in one save_xx message

The save_xx messages, save_binding, save_business, save_service, and save_tModel, all allow multiple entities to be saved in a single call. When the same entity is saved more than once in a single message, it is not clear from the text what, exactly, is supposed to appear in the returned result. To clarify this, the following changes have been made:

For save_binding

Delete “The order in which these are processed is not defined.” From the description of the bindingTemplate argument (section 4.4.12.2). The order is defined in the next paragraph to be first to last.

Change the “Returns” section for save_binding (4.4.12.3) to read:

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.

For save_business

Change the “Returns” section for save_business (4.4.13.4) to read:

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.

For save_service

Change the "Returns" section for save_service (4.4.14.4) to read:

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.

For save_tModel

Change the "Returns" section for save_tModel (4.4.15.4) to read:

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.

uddi-org:misc-taxonomy is called uddi-org:general_keywords in V2

The V1 tModel uddi-org:misc-taxonomy was renamed for version 2. The text of the specification does not indicate that this is so. Add the following text to the end of the section defining uddi-org:general_keywords (section 13.1.2.5):

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.

Missing maxRows attribute on find_relatedBusinesses

The "Returns" section of the API description for find_relatedBusinesses (section 4.2.3.3) talks about the effect of maxRows, and the UDDI V2 schema correctly includes it, but the syntax and arguments sections (4.2.3.1 and 4.2.3.2) inadvertently omit it. To correct this, syntax section is modified to read:

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

And the arguments section is changed to include maxRows as the first item in the list:

- ***maxRows***: This optional integer value allows the requesting program to limit the number of results returned.

Special considerations for the xml:lang attribute

The text of the specification is not clear about how the save_xx APIs deal with omitted xml:lang attributes on elements that allow them (i.e., name and description). To clarify this, a new section is added following section 4.3.5 “Special considerations for validated namespaces” This section reads:

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 on at most one element in the list. When so omitted, the UDDI registry will insert an xml:lang attribute into the element before it is saved. The value of the inserted attribute will be the code for the default language of the registering party.

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, must be unique. save_xx calls that do not meet this requirement will fail with an E_languageError error.

Typos and formatting glitches

A number of additional typographical errors and formatting problems have been reported and fixed since Errata 1 was published. None change the meaning of the text.

Errata 3

Number of <name> elements allowed in find_business and find_service

The API specification states that the maximum number of name elements is five. The number supported ought to be a registry policy, not part of the UDDI specification. To fix this, the text of the specification is altered to leave the maximum open, and to set the limit for UBR in the UBR policies document.

References to the maximum of 5 have been removed from the descriptions of find_business and find_service. Removed the footnotes on the subject of name behavior and placed the relevant remainder in the main text describing name. The name argument in section 4.2.2.2 (arguments to find_business) is altered to read:

- ***name*****:** This optional collection of string values represents one or more partial names potentially qualified with xml:lang attributes. Wildcard searching[1] can be accomplished using the % character. The businessList returned contains businessInfo structures for businesses whose name matches the value(s) passed (lexical-order match – i.e., leftmost in left-to-right languages – if no wild cards are present). 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.

The description of E_tooManyOptions in section 4.2.2.4 (caveats for find_business) is altered to read:

- **E_tooManyOptions**: signifies that the implementation defined limit on the number of name arguments was exceeded.

The name argument in section 4.2.4.2 (arguments for find_service) is altered to read:

- ***name*****:** This optional collection of string values represents one or more partial names potentially qualified with xml:lang attributes. Any businessService data contained in the specified businessEntity with a matching partial name value gets returned. A wildcard character % may be used to signify any number of any characters. If multiple name values are passed, the match occurs on a logical OR basis within any names supplied. 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.

The sentence in section 4.2.4.3 (the “Returns” section of find_service) referring to the limit of five names has been deleted.

A description of E_tooManyOptions was added to section 4.2.4.4 (caveats for find_service). It reads:

- **E_tooManyOptions**: signifies that the implementation defined limit on the number of name arguments was exceeded.

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

Cross-business find_service

Many scenarios have been presented in which it would be useful to be able to search UDDI for businessServices without regard for the businessEntity in which they appear. To enable this, the find_service API is enhanced to allow the attribute businessKey to be omitted or specified as “”. The text of section 4.2.4.1 (syntax of find_service) has been altered to read:

Syntax:

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

The description of businessKey in section 4.2.4.2 has been altered to read:

- ***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.

The text of section 4.2.4.3 (the “Returns” section of find_service) has been altered to read:

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 as “”, 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.

The definition of find_service in the UDDI V2 schema has been altered to read:

```
<element name = "find_service" type = "uddi:find_service"/>
<complexType name = "find_service">
  <sequence>
    <element ref = "uddi:findQualifiers" minOccurs = "0"/>
    <element ref = "uddi:name" minOccurs = "0" maxOccurs = "unbounded"/>
    <element ref = "uddi:categoryBag" minOccurs = "0"/>
    <element ref = "uddi:tModelBag" minOccurs = "0"/>
  </sequence>
  <attribute name = "generic" use = "required" type = "string"/>
  <attribute name = "maxRows" use = "optional" type = "int"/>
  <attribute name = "businessKey" use = "optional"
             type = "uddi:businessKey"/>
</complexType>
```

Deleting an already deleted tModel

It is clear in the text of the specification that deleting a tModel does not physically delete it. Deleted tModels are only “hidden” in the sense that they are invisible to the find_tModel operation. But the text does not make clear that deleting an already deleted tModel has no effect whatever. The tModel remains hidden. The text of section 4.4.6 (the description of delete_tModel) has been altered to read:

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.

Behavior of Service Projections

Establishing service projections

The text of the specification in section 4.4.13.3 paragraph 5 is not clear about precisely when a service projection is established or about what is returned by get_businessDetail. The text has been altered to read:

> 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 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.

Service projections and get_businessDetail

The text for get_businessDetail (section 4.2.7) is silent about the existence of service projections. To make it clear that projected services can occur in the results of a get_businessDetail, the following paragraph is added to the end of the section describing what get_businessDetail returns (section 4.2.7.3):

> 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.

Service projections and find_business

The text of the specification is not clear about whether projected services can appear in the result set for find_business. They can. The text of section 4.2.2.3 (the "Returns" section of find_business) is altered to read:

> This API call returns a businessList on success. This structure contains information about each matching business, and summaries of the businessServices, including service projections[2], 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.

Service projections and find_service

The text of the specification is not clear about whether projected services can appear in the result set for find_service. If a non-empty businessKey is supplied as an argument, they can, otherwise that cannot. A new paragraph has been inserted following the first paragraph of section 4.2.4.3 (the "Results" section of find_service) to explain this:

[2] See section 4.4.14.3

> When the businessKey attribute is used to specify a particular businessEntity to be searched, the find_service API treats service projections[3] 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.

Service projections with V1 find_business and V1 find_service

The text of the specification is silent on how service projections are handled when V2 registries are queried using the V1 find_business and find_service APIs. To correct this, Section 4.1 has been rearranged and retitled "About UDDI inquiry API functions" and the section "Three query patterns" (the former section 4.1) and all its content have been pushed down a level to make room for a new section entitled "4.1.2 Effect of service projections on V1 find_business and find_service calls". This section reads:

> In version 2 of UDDI the concept of "service projections"[4] 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.

Effect on existing service projections when projected service is deleted

When a service that's referred to in a service projection is deleted, it is not clear from the text what happens to the reference. The answer is that nothing special happens. After the service it refers to is deleted, a service projection does not refer to anything. A businessEntity with such a "broken" service projection has a businessService with no content. The attributes businessKey and serviceKey still contain what they did before the referred-to service was deleted. Attempting to do a get_serviceDetail using the serviceKey will, of course, fail with an E_invalidKeyPassed since the businessService no longer exists. Similarly, if the service was deleted as a consequence of its businessEntity having been deleted, attempting to do a get_businessDetail will fail the same way. To enable this behavior and to make it clear in the specification, paragraph 6 of section 4.4.13.3 (the "Behavior" section of save_business) is modified to read:

> 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.

In addition, a change was made to the definition of businessService in the UDDI V2 schema to permit empty businessService elements. (Before this change, the name and bindingTemplate elements were mandatory.)

```
<element name = "businessService" type = "uddi:businessService"/>
 <complexType name = "businessService">
   <sequence>
```

[3] See section 4.4.14.3

[4] See section 4.4.14.3

```
          <element ref = "uddi:name" minOccurs = "0" maxOccurs = "unbounded"/>
          <element ref = "uddi:description"
                   minOccurs = "0" maxOccurs = "unbounded"/>
          <element ref = "uddi:bindingTemplates" minOccurs = "0"/>
          <element ref = "uddi:categoryBag" minOccurs = "0"/>
       </sequence>
       <attribute name = "serviceKey" use = "required"
                  type = "uddi:serviceKey"/>
       <attribute name = "businessKey" use = "optional"
                  type = "uddi:businessKey"/>
    </complexType>
```

UNSPSC Upgrade

The UNSPSC taxonomy has undergone an incompatible change since the Open Draft of the UDDI V2 API Specification was published. In particular, the format of the category codes changed. As a result, following our Best Practice, the UNSPSC 3.1 taxonomy has been deprecated and a new tModel has been established. To reflect this, the following change to section 13.1.2.3 (the tModel definition for unspsc-org:unspsc:3-1) was made:

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

A new section was inserted directly following the section describing unspsc-org:unspsc:3-1 to describe the new tModel:

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.

Incorrect text associated with E_invalidValues

In a number of places, the text for E_invalidValues is inconsistent with its definition in Appendix A. These sections, 4.4.13.5 (Caveats for save_business), 4.4.14.5 (Caveats for save_service), and 4.4.15.5 (Caveats for save_tModel), have been corrected to read:

- **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.

V2 error codes and with V1 messages

The text of the specification is not clear what error codes are returned from a V2 registry when it encounters certain error conditions while processing V1 messages. To correct his, the following text has been added following the list of error codes in section 5.1:

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.

Support for SOAPAction

Section 6.1 of the UDDI Version 2.0 API Specification extends the values accepted for the SOAPAction HTTP header, but it still conflicts with some SOAP toolkits. Also the text is unclear whether this relaxation applied to V2 messages only or also to V1 messages. It applied to both V1 and V2 messages. To relax the allowed values in SOAPAction and to make it clear that the relaxation applies to V1 messages, the text of section 6.1 has been altered to read:

> 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.

No V2 findQualifiers in V1 messages

UDDI version 2 introduces a number of new findQualifiers. The text is not clear that these cannot be used in V1 messages. New text to make this clear has been added directly following the list of new findQualifiers in section 9.0. This reads:

> 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.

Soundex findQualifier removed

The “soundex” search qualifier is not usefully defined for strings containing non-Latin characters, which are common in UDDI names. Rather than extending the definition of soundex to include these characters, “soundex” has been dropped as a UDDI findQualifier. Sections 9.0 and 9.1 have been modified accordingly.

No claims of SOAP encodingStyle allowed

UDDI does not support specifying the SOAP encodingStyle attribute, for example encodingStyle=” http://schemas.xmlsoap.org/soap/encoding/” for elements in the urn:uddi-org:* namespace and rejects messages that make such claims. When rejecting these, it must conform to section 4.4 of the SOAP 1.1 specification.[2] The text of section 6.3 (SOAP Encoding) has been altered to correctly state these requirements:

> 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.

Comply with SOAP 1.1 Specification when rejecting SOAP Actor

UDDI Version 2 registries reject messages with SOAP Actor in them. This is fine, but the way they report this needs to conform to the SOAP 1.1 Specification. The text of section 6.2 has been altered to conform to section 4.4 of the SOAP 1.1 Specification. It now reads:

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.

SOAP <faultcode/> “Server” allowed

Section 6.4 (Support for SOAP Fault) omits the SOAP <faultcode/> “Server”. The text has been altered to include this <faultcode/> as one that may be returned. The text describing “Server” is from the SOAP 1.1. Specification:

- **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.

Applicability of findQualifiers made explicit

The text of the specification is not explicit about which findQualifiers apply to which APIs or about the behavior of an API when a non-applicable findQualifier is passed. It also implies that findQualifiers may be used with get_xx API calls. They can’t. The text has been altered to be explicit on these points. The text of section 9.1.1 (Search Qualifiers enumerated) has been altered to read:

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. 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. 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. Sort qualifiers involving date are secondary in precedence to the sortByName qualifiers. This causes sortByName elements to be sorted within name by date, oldest to newest. 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 (i.e., categoryBag or identifierBag) contains multiple keyedReference elements, 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_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.

Precedence of default findQualifiers made explicit

The default sort order when no findQualifier is specified is to sort the results first by name in ascending order and then, within each name, by date in ascending order. The text of findQualifier Precedence section (9.1.2) is not clear about what precedence these defaults have relative to explicitly specified findQualifiers. This results in ambiguity about what order the results appear in. For example when sortByDateDesc is specified and sortByNameAsc is defaulted, should the explicit sortByDateDesc have higher or lower precedence than the defaulted sortByNameAsc? To make it clear that defaulted sort ordering always has lower precedence, section 9.1.2 is modified to read:

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.

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.

E_invalidKeyPassed missing from find_business

The error code E_invalidKeyPassed is as characteristic of find_business as of any other find_xx API and yet the specification doesn't list it in the Caveats section for find_business. It should. The text of section 4.2.2.4 has been modified to add E_invalidKeyPassed to the list of error codes find_business returns:

- **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.

Clarify orLikeKeys applicability

To be sure no one is misled into thinking that okLikeKeys applies to tModelBags, the text describing orLikeKeys in section 9.1.1 (findQualifiers enumerated) is modified to read:

- **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.

Use of E_unsupported in find_service inconsistent with other find_xx APIs

The specification states that E_unsupported will be returned by find_service if a "blank name value is passed." This is inconsistent with the other find_xx APIs. The text explaining what E_unsupported means in the context of find_service has been changed to be consistent with its use in other APIs:

- **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.

Clarify validation requirements for uddi-org:owning-Business

The section describing uddi-org:owning-Business (13.1.2.7) does not make it clear how its use is validated. The text of this section is modified to make it clear that the referred-to businessEntity must be one that the publisher has published:

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

This tModel identifies the businessEntity that published or owns the tagged information. Used with tModels to establish an “owned” relationship with a registered businessEntity The referred-to businessEntity must exist and must have been published by the publisher that publishes the tagged information.

Empty <categoryBag/> and <identifierBag/> elements not allowed

The UDDI V2 Schema allows empty <categoryBag/> and <identifierBag/> elements, each of which is optional in every place it is used. This leads to questions about whether an empty bag is equivalent to an omitted one and if not, what the difference is. Since there is no intended difference, having two ways of expressing the same thing is unnecessarily confusing. To eliminate the confusion, the UDDI V2 schema has been modified to require <categoryBag/> and <identifierBag/> elements to have at least one child:

```
<element name = "categoryBag" type = "uddi:categoryBag"/>
<complexType name = "categoryBag">
   <sequence>
      <element ref = "uddi:keyedReference" maxOccurs = "unbounded"/>
   </sequence>
</complexType>

<element name = "identifierBag" type = "uddi:identifierBag"/>
<complexType name = "identifierBag">
   <sequence>
      <element ref = "uddi:keyedReference" maxOccurs = "unbounded"/>
   </sequence>
</complexType>
```

Duplicate keys in delete_binding, delete_business, delete_publisherAssertion, and delete_service

The text of the specification is unclear what behavior will result if a delete_binding, delete_business, delete_service, or delete_publisherAssertion message containing duplicate uuid_keys is sent. Unless they fail for other reasons (e.g., E_userMismatch), such message fail with E_invalidKeyPassed or E_assertionNotFound. The text of the specification has been updated in the following sections.

Caveats for delete_binding (section 4.4.2.4)

- **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.

Caveats for delete_business (section 4.4.3.5)

- **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.

Caveats for delete_publisherAssertions (section 4.4.4.5)

- **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.

Caveats for delete_service (section 4.4.5.4)

- **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.

Empty <contacts/> elements not allowed

The UDDI is ambiguous with respect to how a businessEntity with no contacts should be represented. The schema allows two representations – a <contacts/> element with no content may be specified or <contacts/> may be omitted altogether. This ambiguity adds no value and adds unneeded complexity to UDDI. To eliminate this, the UDDI V2 schema is modified to exclude empty <contacts/>:

```
<element name = "contacts" type = "uddi:contacts"/>
<complexType name = "contacts">
    <sequence>
        <element ref = "uddi:contact" maxOccurs = "unbounded"/>
    </sequence>
</complexType>
```

Either <accessPoint/> or <hostingRedirector/> is required

The UDDI V2 schema allows both <accessPoint/> and <hostingRedirector/> to be omitted from <bindingTemplate/> this is an error – every <bindingTemplate/> must have either an <accessPoint/> or a <hostingRedirector/>, but not both. The UDDI V2 schema is modified to state this:

```
<element name = "bindingTemplate" type = "uddi:bindingTemplate"/>
<complexType name = "bindingTemplate">
  <sequence>
    <element ref = "uddi:description"
       minOccurs = "0" maxOccurs = "unbounded"/>
    <choice>
      <element ref = "uddi:accessPoint"/>
      <element ref = "uddi:hostingRedirector"/>
    </choice>
    <element ref = "uddi:tModelInstanceDetails"/>
  </sequence>
  <attribute name = "serviceKey" use = "optional"
    type = "uddi:serviceKey"/>
  <attribute name = "bindingKey" use = "required"
    type = "uddi:bindingKey"/>
</complexType>
```

Elements in inquiry messages are subject to truncation

Elements stored by UDDI implementations are subject to a maximum length, as documented in the UDDI V2.0 Data Structure Reference, Appendix D. In publication messages when an element exceeding its maximum length is encountered, it is truncated before being processed. In contrast, except for <name/> elements, the text of the specification is not clear how elements whose length exceeds these maxima are handled when they are passed in the inquiry API messages. For <name/> elements that are too long, the spec defines a special error code E_nameTooLong that is returned when a long name is encountered.

To specify and regularize how long elements are dealt with in UDDI, the handling of long elements in the inquiry APIs is changed to truncate them at their maximum lengths. The following changes were made:

Added "Elements whose length exceed the maximum lengths" section at the end of "About UDDI Inquiry APIs":

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.

Retired E_nameTooLong

Removed it from the Caveats sections of find_business (section 4.2.4.4), find_service (section 4.2.4.4), and find_tModel (section 4.2.5.4).

Changed description of E_nameTooLong in Error Codes (section 5.1) to indicate it is retired:

- **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.

Typos and formatting glitches

A number of additional typographical errors and formatting problems have been reported and fixed since Errata 2 was published. None change the meaning of the text.

Errata 4

Incomplete specification of uddi-org:operators

The tModel uddi-org:operators is incompletely specified. No specification of the valid values is given, no specification of how validity checking is to be done is given, and no indication of its intended usage is given. In addition, if it is really an identifier system, the implication is that each operator is identified by a different value, making it impossible to find all of the operators, a common scenario. To correct these shortcomings, the text defining the uddi-org:operators tModel is changed to read:

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.

Proper error for add_publisherAssertions

In a footnote to the add_publisherAssertions API, it incorrectly states that E_unsupported is returned when the situation covered by the footnote is encountered. This is incorrect. The correct error code is E_fatalError. The text of the footnote is modified to read:

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.

Wildcard behavior clarified

In find_business, find_service, and find_tModel the behavior of wildcards in the name argument is incompletely described. In particular, it is unclear whether UDDI behaves as though a trailing wildcard had been specified even when the argument actually contains user-specified wildcards. It doesn't. To make this clear, the text describing the name argument of find_business is modified to read:

***name*:** This optional collection of string values represents one or more names potentially qualified with xml:lang attributes. Wildcard searching[5] 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>

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

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.

The text describing the name argument of find_service is modified to read:

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.

And the text describing the name argument of find_tModel is modified to read:

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.

Specifying zero publisherAssertions in a set_publisherAssertions is permitted

The syntax of set_publisherAssertions and the description of the publisherAssertion argument incorrectly state that at least one assertion must be present. In fact, zero such arguments is permitted, allowing a publisher to remove all publisherAssertions with a single call. The text of the syntax for set_publisherAssertions is modified to read:

Syntax:

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

And the text of the publisherAssertion argument is modified to read:

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.

Special considerations concerning the uddi-org:general_keywords taxonomy

Several special considerations concerning the use of the uddi-org:general_keywords taxonomy are not clearly specified. First, it is not clear what the behavior is when a save_xx operation that contains a keyedReference that has no tModelKey specified. Second, it is not clear whether a keyName needs to be specified in keyedReferences that use uddi-org:general_keywords. To clear these up, a new section is added to the text in section 4.3 About UDDI Publishing API Functions:

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 be specified and 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.

And the text in the section is slightly rearranged to accommodate the new section. Finally, the text describing uddi-org:general_keywords is modified to read:

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.

Proper fault when responding to a message which is of indeterminate version

The specification is not clear that, when faced with messages that are of indeterminate version, UDDI version 2 registries respond with version 2 faults. To make this clear, text at the end of section 5.1.2 *Error reporting with the dispositionReport element* has been added:

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.

White space and conformance with the UDDI schema

The specification is inconsistent (or at least ambiguous) with respect to how leading and trailing white space that is not allowed by the UDDI schema is to be treated. To clarify this, the text in section 3.1.6 *White Space* is changed to read:

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.

Behavior when inapplicable or mutually exclusive find qualifiers are specified

The specification is not clear about the behavior of API calls in which inapplicable or mutually exclusive find qualifiers are specified. To make this clear, section 9.2.2 *findQualifier Precedence* is revised to read:

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.

No matching requirement when saving service projections

The specification states, in the section discussing the behavior of save_business (4.4.13.3) that a service projection must either “match” the service being projected or contain only the name of the service being projected. There are several problems here. First, what it means to “match” is not well defined. Second, if a definition were put forward the checking involved would likely be a considerable burden on implementations. And third, the benefit to publishers of the behavior – notifying them when they erroneously attempt to update the content of a service while establishing a projection to it – is of marginal value. For these reasons, the matching requirement is dropped.

The alternative of providing only the name was largely an artifact of the UDDI schema which required the name element to be present. This requirement has since been dropped (Errata 3, *Behavior of Service Projections*). Given that name can now be omitted and given the low value to publishers of checking that a name, if specified, is the same as the name of the service being projected, the requirement to check for equality has been dropped.

The net effect of these changes is that, except for the businessKey and the serviceKey attributes, when a businessService is used to establish a service projection its content is ignored.

The paragraph in section 4.4.13.3 of the specification that reads:

> When saving a businessEntity containing a service reference, the content of the businessService provided in the save_business must either match the content of the referenced businessService or contain only the name of the businessService being referenced. If the businessService provides does not meet these requirements the save will fail with the error E_invalidProjection.

Is replaced by:

> 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.

xml:lang is truly optional

UDDI V3 is adding XML digital signatures. If a node is permitted or required to add or change the content of an entity during the publication, making digital signatures work correctly becomes tricky or impossible for publishers. Making them work is tricky if publishers can anticipate what the node will add or change; impossible if they cannot.

One situation in which this takes place is the required handling of the xml:lang attribute when the publisher omits it. Prior to this change V2, nodes were required to add xml:lang when it is permitted but unspecified by a publisher. In V3 nodes do not do this.

As a result, unlike V2, V3 entities retrieved from a registry do not necessarily have xml:lang attributes. This incompatibility is problematic for multi-version registries because, the V2 specs imply that returned data always has an xml:lang attribute wherever it is permitted because nodes are required to add them. When data stored with V3 save_xx APIs is retrieved using V2 APIs, this means that nodes will need to invent xml:lang attributes as needed to fill them in when they are absent in the data.

This relieves that burden without requiring V2 nodes to change their behavior by making it clear to V2 clients that it is possible that xml:lang attributes in data retrieved from a V2 registry MAY be omitted.

To effect this change, the text of section 4.3.6, *Special considerations for the xml:lang attribute* of the API specification be changed to read:

> 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.

Section 7.1, *Support for multiple languages* of the API specification is changed to read:

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.

And section 7.1.2, *Default language codes* of the API specification be changed to read:

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.

Sorting by name uses the first name when there is more than one name

Many find_xx operations allow sorting based on the content of name elements, but the specification is not clear which name is used when there is more than one. To clarify this, the following paragraph is appended to the end of section 4.3.6 *Special considerations for the xml:lang attribute:*

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.

And the descriptions sortByNameAsc and sortByNameDesc of in 9.1.1 *findQualifiers enumerated* are modified to read:

- **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.

Clarify validity check for uddi-org:owningBusiness

The specification is unclear about whether entities other than tModels can be tagged with the uddi-org:owningBusiness identifier system. They can't. Also, uddi-org:owningBusiness is a taxonomy, not an identifier system because multiple tModels may be categorized with the same value. Accordingly, the text describing uddi-org:owningBusiness is modified to read:

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.

The tModel uddi-org:homepage is deprecated

It has become a common practice for publishers to use the discoveryURL element with a useType of "homepage" to designate their homepages. This is an easy, sensible practice but it conflicts with the alternative practice of modeling a homepage as a bindingTemplate whose technical fingerprint is the tModelKey for uddi-org:homepage. The remove the conflict, the tModel uddi-org:homepage is deprecated and its description is changed to read:

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".

orAllKeys applies to find_binding

In section 9.1.1 find_binding was inadvertantly omitted from the list of find_xx operations to which orAllKeys applies. Accordingly, the paragraph of section 9.1.1 describing orAllKeys is modified to read:

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.

References

1) UDDI Version 2.0 API Specification: http://www.uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.doc.

2) SOAP 1.1 Specification: http://www.w3.org/TR/SOAP/.