# 3 UDDI Registry Data Structures

## 3.1 Data structure overview

This chapter describes the semantics of the data structures that are specified by the UDDI API Schema. Refinements that are specific to individual APIs are described in Chapter 5 *UDDI Programmers APIs*.

As described in Section 1.6 *Representing Information within UDDI*, the information that makes up a UDDI registry consists of instances of four core data structure types, the `businessEntity`, the `businessService`, the `bindingTemplate` and the `tModel`, together with instances of additional data structure types defined in the UDDI API Schema.

The four core types and their relationships are shown in a simplified diagram in Figure 1 and are explained in detail in this chapter.

![Figure 1 - UDDI core data structures](http://www.uddi.org/pubs/uddi-v3.00-published-20020719.htm)

Figure 1 - UDDI core data structures

The schema also defines a number of request and response structures, each of which contain the core structures, references to the core structures, or summary versions of them; see Chapter 5 *UDDI Programmers API's* for details.
3.2 Design Principles

Each of the core data structure types is used to express specific types of data, arranged in the relationship shown in Figure 1. A particular instance of an individual fact or set of related facts is expressed using XML according to the definition of these core types. For instance, two separate businesses may publish information in a UDDI registry about Web services they offer. Information describing each business and its Web services all exists as separate instances of the core data structures stored within the UDDI registry.

3.2.1 Keys as unique identifiers

Instances of many data structures in UDDI, including all of the core data structures are kept separately, and are accessed individually by way of unique identifiers called keys. An instance in the registry gets its keys at the time it is first published. Publishers may assign keys; if they don't, the UDDI node MUST assign them. See Section 4.4 About uddiKeys.

3.2.2 Containment and references

The core data structures are sensitive to the containment relationships found in the UDDI API schema and shown in Figure 1. The businessEntity structure contains one or more distinct businessService structures. Similarly, individual businessService structures contain specific instances of bindingTemplate structures.

It is important to note that no single instance of an entity is ever “contained” by more than one containing entity. This means, for example, that only one specific businessEntity structure (identified by its unique key value) will ever contain a specific instance of a businessService structure (also identified by its own unique key).

References, on the other hand, operate differently. We can see an example of this in Figure 1 where the bindingTemplate entities refer to instances of tModel entities. References to a given entity can occur multiple times, as needed.

Determining what is a reference and what is the key for a specific entity is straightforward. Each kind of keyed entity has an attribute whose type is a corresponding type of key. For example, businessEntity has a businessKey attribute and a businessService has a serviceKey attribute. The value of this attribute is the entity’s key. All other keys are references or containment relationships. Taking the bindingTemplate as an example, the tModelKey that occurs in its inner structure is a reference and the serviceKey that occurs in the bindingTemplate is a containment relationship.

3.2.3 Collections

Many elements in the UDDI API Schema may occur multiple times. Those elements that do not have a complex inner structure, for example, name and description, are provided in a list. Elements that do have a more complex inner structure are usually grouped in their own container element. For example, the contacts structure is a container where one or more contact structures reside.

3.2.4 Optional attributes

In the data structure elements of the UDDI API Schema, there are many optional attributes, for example, keyName and useType. Most optional attributes have defaults of empty string (“”). During schema assessment, this produces a single representation for an omitted or empty string in an optional attribute. Consider the following two keyedReferences:

```xml
<keyedReference
  tModelKey=“uddi:ubr.uddi.org: categorization: geo3166-2”
  keyName="”
  keyValue="US-CA" />
<keyedReference
  tModelKey=“uddi:ubr.uddi.org: categorization: geo3166-2”
  keyValue="US-CA" />
```

UDDI V3 Specification
http://www.uddi.org/pubs/uddi-v3.00-published-20020719.htm

35 of 430
12/6/2008 9:18 PM
Semantically speaking from the perspective of UDDI, omitted attributes are identical to empty attributes. However, with respect to signing, specifically, canonicalization, omitted attributes are different from empty attributes. Therefore, the digital signatures of the above two keyedReferences are different, even though clients would consider the two keyedReferences be identical.

The difference, from a perspective of canonicalization, puts additional burden on clients in publishing entities. As a result, when applicable, the data structure elements of UDDI API Schema define default values for optional attributes, so that omitted attributes are treated as attributes with default value with respect to signing.

The exceptions are xml:lang and keyValue in addressLine. Both prohibit empty string. Hence, the ambiguity discussed above is not applicable. In the case of xml:lang, empty string is not a valid language code. In the case of keyValue in addressLine, the definition of keyValue requires the string to have a minimal length of one.

### 3.3 businessEntity Structure

Each businessEntity entity contains descriptive information about a business or organization and, through its contained businessService entities, information about the services that it offers. From an XML standpoint, the businessEntity is the top-level data structure that holds descriptive information about the business or organization it describes. Each contained businessService describes a logical service offered by the business or organization. Similarly, each bindingTemplate contained within a given businessService provides the technical description of a Web service that belongs to the logical service that is described by the businessService.

#### 3.3.1 Structure diagram
### 3.3.2 Documentation

A given instance of the businessEntity structure is uniquely identified by its `businessKey`. When a businessEntity is published within a UDDI registry, the `businessKey` MUST be omitted if the publisher wants the registry to generate a key. When a businessEntity is retrieved from a UDDI registry, the `businessKey` MUST be present.

`discoveryURLs` is a list of Uniform Resource Locators (URL) that point to alternate, file based service discovery mechanisms.

Simple textual information about the businessEntity, potentially in multiple languages, is given by its `name`, short business `description` and `contacts`. The required, non-empty name and the optional description can occur multiple times. `contacts` is a simple list of single contact information.

`businessServices` is a list of business services provided by the businessEntity.

In addition to the `businessKey`, that uniquely identifies the businessEntity within the registry, the `identifierBag` contains a list of other identifiers, each valid in its own identifier system. Examples of identifiers are a tax identifier or D-U-N-S® number.

The `categoryBag` contains a list of business categories that each describes a specific business aspect of the businessEntity. Examples of categories are industry, product category or geographic region.

A businessEntity entity MAY be digitally signed using XML digital signatures. When a businessEntity is signed, each digital signature MUST be provided by its own `dsig:Signature` element. Appendix I *Support for XML Digital Signatures* covers the use of this element in accordance with the XML-Signature specification.

#### 3.3.2.1 discoveryURLs

The discoveryURLs structure is a simple container of one or more `discoveryURL` elements.

#### 3.3.2.2 discoveryURL

A discoveryURL is a URL that points to Web addressable (via HTTP GET) discovery documents. The expected return document is not defined. Rather, a framework for establishing conventions is provided, and a particular convention is defined within this specification.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>useType</code></td>
<td>optional</td>
</tr>
</tbody>
</table>
Each individual discoveryURL MAY be adorned with a useType attribute that designates the name of the convention that the referenced document follows. A reserved convention value is “businessEntity”. It is RECOMMENDED that discoveryURLs qualified with this value point to XML documents of the type businessEntity, as defined in the UDDI API Schema.

An example of a discoveryURL, generated by a UDDI node that is accessible at www.example.com and rendered by the publisher of the businessEntity that is identified by the businessKey “uddi:example.com:registry:sales:53”, is:

```xml
</discoveryURL>
```

Another reserved value for discoveryURL is “homepage”. Adorning a discoveryURL with this value signifies that a business’s homepage can be discovered at that URL.

### 3.3.2.3 name

A businessEntity MAY contain more than one name. Multiple names are useful, for example, in order to specify both the legal name and a known abbreviation of a businessEntity, or in order to support romanization (see Appendix D Internationalization).

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>xml:lang</td>
<td>optional</td>
</tr>
</tbody>
</table>

When a name is expressed in a specific language (such as the language into which a name has been romanized), it SHOULD carry the xml:lang attribute to signify this. When a name does not have an associated language (such as a neologism not associated with a particular language), the xml:lang attribute SHOULD be omitted.

As is defined in the XML specification, an occurrence of the xml:lang attribute indicates that the content to which it applies (namely the element on which it is found and to all its children, unless subsequently overridden) is to be interpreted as being in a certain natural language. Legal values for such attributes are specified in the IETF standard RFC 1766 and its successors (including, as of the time of the present writing, RFC 3066). As is indicated therein, language values begin with a primary language tag, and are optionally followed by a series of hyphen-delimited sub-tags for country or dialect identification; the tags are not case-sensitive. Examples include: “EN-us”, “FR-ca”.

### 3.3.2.4 description

A businessEntity can contain several descriptions, for example, in different languages.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>xml:lang</td>
<td>optional</td>
</tr>
</tbody>
</table>

In order to signify the language in which the descriptions are expressed, they MAY carry xml:lang values. There is no restriction on the number of descriptions or on what xml:lang value that they may have.
### 3.3.2.5 contacts

The contacts structure itself is a simple collection of one or more **contact** structures.

- uddi:contacts
  - uddi:contact

### 3.3.2.6 contact

The contact structure records contact information for a person or a job role within the businessEntity so that someone who finds the information can make human contact for any purpose. This information consists of one or more optional elements, along with a person's name. Contact information exists by containment relationship alone; the contact structure does not provide keys for tracking individual contact instances.

- uddi:contact
  - uddi:personName
  - uddi:phone
  - uddi:email
  - uddi:address
    - uddi:addressLine

### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>useType</td>
<td>optional</td>
</tr>
</tbody>
</table>

The **useType** attribute is used to describe the type of contact in unstructured text. Suggested examples include “technical questions”, “technical contact”, “establish account”, “sales contact”, etc.
**description** is used to describe how the contact information should be used. Publishing several descriptions, e.g. in different languages, is supported. To signify the language in which the descriptions are expressed, they MAY carry `xml:lang` values.

**personName** is the name of the person or name of the job role supporting the contact. Publishing several names, e.g. for romanization purposes, is supported.

<table>
<thead>
<tr>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>xml:lang</td>
</tr>
</tbody>
</table>

In order to signify the contextual language (if any) in which a given name is expressed in (such as the language into which a name has been romanized), it SHOULD carry the `xml:lang` attribute. See Section 3.3.2.3 `name` for details on using `xml:lang` values in `name` elements. There is no restriction on the number of `personNames` or what `xml:lang` value each may have. An example for a role might be:

```xml
<contact useType="Technical support">
  <personName>Administrator</personName>
</contact>
```

**phone** is used to hold telephone numbers for the contact. This element MAY be adorned with an optional `useType` attribute for descriptive purposes.

**email** is the email address for the contact. This element MAY be adorned with an optional `useType` attribute for descriptive purposes.

**address** is the postal address for the contact.

### 3.3.2.7 address

**address** represents the contact’s postal address, in a form suitable for addressing an envelope. The address structure is a simple list of address lines.

<table>
<thead>
<tr>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>xml:lang</td>
</tr>
<tr>
<td>useType</td>
</tr>
<tr>
<td>sortCode</td>
</tr>
<tr>
<td>tModelKey</td>
</tr>
</tbody>
</table>

Address structures have four optional attributes.

The `xml:lang` value describes the language the address is expressed in. There is no restriction on the number of addresses or what `xml:lang` value they may have. Publication of addresses in several languages, e.g. for use in multilingual countries, is supported. See Appendix D *Internationalization* for an example.
The **useType** describes the address’ type in unstructured text. Suggested examples include “headquarters”, “sales office”, “billing department”, etc.

The **sortCode** attribute is deprecated because of the guarantee of preserving the document order (see Section 4.5.3 *Preservation of Document Order*). In order to achieve this behavior, the data has just to be published in the desired order.

The **tModelKey** is a tModel reference that specifies that keyName keyValuePair pairs given by subsequent addressLine elements, if addressLine elements are present at all, are to be interpreted by the address structure associated with the tModel that is referenced. For a description of how to use tModels in order to give the address line list structure and meaning, see Appendix D *Internationalization*.

### 3.3.2.8 addressLine

**addressLine** contains the actual address in text form.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyName</td>
<td>optional</td>
</tr>
<tr>
<td>keyValue</td>
<td>optional</td>
</tr>
</tbody>
</table>

Each addressLine element MAY be adorned with two optional descriptive attributes, **keyName** and **keyValue**. Both attributes MUST be present in each address line if a tModelKey is specified in the address structure. When no tModelKey is provided for the address structure, the keyName and keyValue attributes have no defined meaning.

### 3.3.2.9 businessServices

The **businessServices** structure is used to describe families of Web services. This simple container holds one or more businessService entities (see Section 3.4 *businessService structure*).

![Diagram of businessServices structure](http://www.uddi.org/pubs/uddi-v3.00-published-20020719.htm)

### 3.3.2.10 identifierBag

The optional **identifierBag** element allows businessEntity structures to be identified according to published identifier systems, for example, Dun & Bradstreet D-U-N-S® numbers or tax identifiers.

![Diagram of identifierBag structure](http://www.uddi.org/pubs/uddi-v3.00-published-20020719.htm)
An identifierBag is a list of one or more keyedReference structures, each representing a single identification. For a full description on how to establish an identity, see Appendix E Using Identifiers.

### 3.3.2.11 keyedReference (in identifierBags)

A keyedReference, when included in an identifierBag, represents an identifier of a specific identifier system.

#### Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>tModelKey</td>
<td>required</td>
</tr>
<tr>
<td>keyName</td>
<td>optional</td>
</tr>
<tr>
<td>keyValue</td>
<td>required</td>
</tr>
</tbody>
</table>

The keyedReference consists of the three attributes `tModelKey`, `keyName` and `keyValue`. The required `tModelKey` refers to the tModel that represents the identifier system, and the required `keyValue` contains the actual identifier within this system. The optional `keyName` MAY be used to provide a descriptive name for the identifier. Omitted `keyNames` are treated as empty `keyNames`.

For example, identifying SAP AG by its Dun & Bradstreet D-U-N-S® Number, using the corresponding `tModelKey` within the UDDI Business Registry, is done as follows:

```xml
<identifierBag>
  <keyedReference
    tModelKey="uddi:ubr.uddi.org:identifier:dnb.com:D-U-N-S"
    keyName="SAP AG"
    keyValue="31-626-8655" />
</identifierBag>
```

### 3.3.2.12 categoryBag

The optional categoryBag element allows businessEntity structures to be categorized according to published categorization systems. For example, a businessEntity might contain UNSPSC product and service categorizations that describe its product and service offering and ISO 3166 geographical regions that describe the geographical area where these products and services are offered.
Similar to the identifierBag, a categoryBag contains a simple list of `keyedReference` structures, each containing a single categorization. The categoryBag MAY also contain a simple list of `keyedReferenceGroup` structures. At least one keyedReference or one keyedReferenceGroup MUST be provided within the categoryBag.

For a full description of how to establish a categorization, see Appendix F *Using Categorization*.

### 3.3.2.13 `keyedReference` (in categoryBags)

As within an identifierBag (see Section 3.3.2.13 *keyedReference (in identifierBags)*), a keyedReference contains the three attributes `tModelKey`, `keyName` and `keyValue`. The required `tModelKey` refers to the tModel that represents the categorization system, and the required `keyValue` contains the actual categorization within this system. The optional `keyName` can be used to provide a descriptive name of the categorization. Omitted `keyNames` are treated as empty `keyNames`. A `keyName` MUST be provided in a `keyedReference` if its `tModelKey` refers to the general_keywords category system (see also Section 5.1.7 *Matching Rules for keyedReferences and keyedReferenceGroups*).

For example, in order to categorize a businessEntity as offering goods and services in California, USA, using the corresponding ISO 3166 `tModelKey` within the UDDI Business Registry, one would add the following `keyedReference` to the businessEntity's categoryBag:

```xml
<keyedReference
  tModelKey="uddi:ubr.uddi.org:categorization:geo3166-2"
  keyName="California, USA"
  keyValue="US-CA" />
```

### 3.3.2.14 `keyedReferenceGroup`

A `keyedReferenceGroup`, by itself, is a simple list of `keyedReference` structures that logically belong together.
tModelKey required

The keyedReferenceGroup MUST contain a tModelKey attribute that specifies the structure and meaning of the keyedReferences contained in the keyedReferenceGroup.

For example, to categorize a businessEntity as being located at the geodetic point that is specified by the latitude/longitude pair 49.6827/8.2952 using the corresponding World Geodetic System 1984 (WGS 84) tModelKey within the UDDI Business Registry, one would add the following keyedReferenceGroup to the businessEntity's categoryBag:

```
<keyedReferenceGroup tModelKey="uddi:ubr.uddi.org:categorizationGroup:wgs84" >
  <keyedReference
    tModelKey="uddi:ubr.uddi.org:categorization:wgs84:latitude"
    keyName="WGS 84 Latitude"
    keyValue="+49.682700" />
  <keyedReference
    tModelKey="uddi:ubr.uddi.org:categorization:wgs84:longitude"
    keyName="WGS 84 Longitude"
    keyValue="+008.295200" />
</keyedReferenceGroup>
```

### 3.4 businessService Structure

The businessService structure represents a logical service and contains descriptive information in business terms. A businessService is the logical child of a single businessEntity, the provider of this businessService. Technical information about the businessService is found in the contained bindingTemplate entities.

In some cases, businesses would like to share or reuse services, e.g. when a large enterprise publishes separate businessEntity structures. This can be done by using the businessService structure as a projection to a published businessService, as explained below.

### 3.4.1 Structure Diagram

```
attributes
```
3.4.2 Documentation

A given businessService entity is uniquely identified by its serviceKey. When a businessService is published within a UDDI registry, the serviceKey MUST be omitted if the publisher wants the registry to generate a key. When a businessService is retrieved from a UDDI registry, the serviceKey MUST be present.

The businessKey attribute uniquely identifies the businessEntity which is the provider of the businessService. Every businessService is “contained” in exactly one businessEntity.

When a businessService is published within a UDDI registry, the businessKey MAY be omitted if the businessService is a part of a fully expressed businessEntity element. When a businessService is retrieved from a UDDI registry, the businessKey MUST be present. This behavior provides the ability to browse through the containment relationships given any of the core elements as a starting point.

The businessKey may differ from the publishing businessEntity’s businessKey. This indicates a service projection. A service projection allows a business or organization to include in its businessEntity a businessService offered by some other business or organization. A projected businessService is made a part of a businessEntity by reference as opposed to by containment. Projections to the same service can be made in any number of business entities.

Simple textual information about the businessService, potentially in multiple languages, is given by its name and short service description. The non-empty name, required except when indicating a service projection, and the optional description can occur multiple times. More information about the structure of the name and description elements can be found in Section 3.3 businessEntity Structure.

bindingTemplates is a list of technical descriptions for the Web services provided.

The categoryBag contains a list of business categories that each describes a specific business aspect of the businessService (e.g. industry, product category or geographic region) and is valid in its own category system. More information about the categoryBag element can be found in Section 3.3 businessEntity Structure.

A businessService entity MAY be digitally signed using XML digital signatures. When a businessService is signed, each digital signature MUST be provided by its own dsig:Signature element. Appendix I Support for XML Digital Signature covers the use of this element in accordance with the XML-Signature specification.

3.4.2.1 bindingTemplates

The bindingTemplates structure holds, for a given businessService, the bindingTemplate entities that provide the technical descriptions of the Web services that constitute the businessService.

See Section 3.5 bindingTemplate structure for details on bindingTemplates.

3.5 bindingTemplate Structure
Technical descriptions of Web services are provided by bindingTemplate entities. Each bindingTemplate describes an instance of a Web service offered at a particular network address, typically given in the form of a URL. The bindingTemplate also describes the type of Web service being offered using references to tModels, application-specific parameters, and settings.

Each bindingTemplate is contained in a businessService.
3.5.1 Structure Diagram

```
bindingTemplate
|--> uddi:description
    | 0..∞
|--> uddi:accessPoint
|   |→ uddi:hostingRedirect

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>bindingKey</td>
<td>optional</td>
</tr>
<tr>
<td>serviceKey</td>
<td>optional</td>
</tr>
</tbody>
</table>
```

3.5.2 Documentation

A given bindingTemplate entity is uniquely identified by its **bindingKey**. When a bindingTemplate is published within a UDDI registry, the bindingKey MUST be omitted if the publisher wants the registry to generate a key. When a bindingTemplate is retrieved from a UDDI registry, the bindingKey MUST be present.

The **serviceKey** attribute uniquely identifies the businessService that contains the bindingTemplate. When a bindingTemplate is published within a UDDI registry, the serviceKey MAY be omitted if the bindingTemplate is a part of a fully expressed businessService element. When a bindingTemplate is retrieved from a UDDI registry, the serviceKey MUST be present.

Simple textual information about the bindingTemplate, potentially in multiple languages, is given by its short binding **description**. It is optional and can occur multiple times. More information about the structure of the description element can be found in Section 3.3 businessEntity structure.

The **accessPoint** is a string used to convey the network address suitable for invoking the Web service being described. This is typically a URL but may be an electronic mail address, or even a telephone number. No assumptions about the type of data in this field can be made without first understanding the technical...
requirements associated with the Web service.

The **hostingRedirector** is a deprecated element, since its functionality is now covered by the accessPoint. For backward-compatibility, it can still be used, but it is not recommended. See the set of UDDI Version 2 Specifications for a description on hostingRedirector.

Either an accessPoint or a hostingRedirector must be provided within a bindingTemplate.

The **tModelInstanceDetails** structure is a list of one or more tModelInstanceInfo elements. The collection of tModelKey attributes found in the tModelInstanceInfo elements together form the "technical fingerprint" of a Web service that can be used to identify compatible services.

The **categoryBag** contains a list of categorizations that each describes a specific aspect of the bindingTemplate and is valid in its own category system. A categoryBag in a bindingTemplate can be used, for example, to indicate that the Web service described by the bindingTemplate has the status "test" or "production". More information about the structure of the categoryBag element can be found in Section 3.3 businessEntity Structure.

A bindingTemplate entity **MAY** be digitally signed using XML digital signatures. When a bindingTemplate is signed, each digital signature **MUST** be provided by its own **dsig:Signature** element. Appendix I Support for XML Digital Signature covers the use of this element in accordance with the XML-Signature specification.

### 3.5.2.1 accessPoint

The accessPoint element is an attribute-qualified URI, typically a URL, representing the network address of the Web service being described. The notion of Web service seen here is fairly abstract and many types of network addresses are accommodated.

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>useType</td>
<td>optional</td>
</tr>
</tbody>
</table>

The purpose of the optional attribute **useType** is to facilitate the description of several types of accessPoints.

The following useType attributes values are pre-defined by UDDI:

- **endPoint**: designates that the accessPoint points to the actual service endpoint, i.e. the network address at which the Web service can be invoked,
- **bindingTemplate**: designates that the accessPoint contains a bindingKey that points to a different bindingTemplate entry. The value in providing this facility is seen when a business or entity wants to expose a service description (e.g. advertise that they have a service available that suits a specific purpose) that is actually a service that is described in a separate bindingTemplate record. This might occur when many service descriptions could benefit from a single service description,
- **hostingRedirector**: designates that the accessPoint can only be determined by querying another UDDI registry. This might occur when a service is remotely hosted.
- **wsdlDeployment**: designates that the accessPoint points to a remotely hosted WSDL document that already contains the necessary binding information, including the actual service endpoint.

The useType attribute **MAY** contain other values than the four listed above. See Appendix B Using and Extending the useType Attribute for more information.
3.5.2.2 tModelInstanceDetails

This structure is a container for one or more tModelInstanceInfo structures.

When a bindingTemplate is published it SHOULD contain, a tModelInstanceDetails element that in turn contains in its tModelInstanceInfo structures one or more tModel references. This arbitrarily ordered collection of references is called the “technical fingerprint” of the Web service. It indicates that the Web service being described complies with the specific and identifiable specifications implied by the tModelKey values provided. During an inquiry, interested parties can use this information to look for bindingTemplate entities that contain a specific fingerprint or partial fingerprint.

3.5.2.3 tModelInstanceInfo

Each tModelInstanceInfo structure represents bindingTemplate entity-specific details for each tModel referenced.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>tModelKey</td>
<td>required</td>
</tr>
</tbody>
</table>

The required tModelKey attribute references a tModel that represents a specification with which the Web service represented by the containing bindingTemplate complies.

The description is an optional repeating element. Each description, optionally qualified by an xml:lang attribute, describes what role the tModel plays in the overall service description.

The optional instanceDetails element can be used when tModel-specific settings or other descriptive information are required either to describe a tModel specific component of a service description or to support services that require additional technical data support (e.g. via settings or other handshake operations).
3.5.2.4 instanceDetails

This structure holds service instance-specific information that is required to either understand the service implementation details relative to a specific tModel, or to provide further parameter and settings support.

The `description` is an optional repeating element. Each description, optionally qualified by an xml:lang attribute, describes the purpose and/or use of the particular instanceDetails entry.

The `overviewDoc` is an optional repeating element, used to house references to remote descriptive information or instructions related to the use of a particular tModel and its instanceParms. Multiple overviewDoc elements are useful, for example, to handle alternative representations of the documentation.

The `instanceParms` is an optional element of type string, used to locally contain settings or parameters related to the proper use of a tModelInstanceInfo. The suggested format is a namespace-qualified XML document so that the settings or parameters can be found in the XML documents elements and attributes.

At least one overviewDoc or instanceParms has to be provided within the instanceDetails.

3.5.2.5 overviewDoc

This structure describes overview information about a particular tModel use within a bindingTemplate.

The `description` is an optional repeating element. Each description, optionally qualified by an xml:lang attribute, holds a short descriptive overview of how a
particular tModel is to be used. The optional overviewURL is to be used to hold a URL that refers to a long form of an overview document that covers the way a particular tModel is used as a component of an overall Web service description.

At least one description or an overviewURL must be provided within the overviewDoc.

3.5.2.6 overviewURL

The RECOMMENDED format for the overviewURL is a URI that is suitable for retrieving the actual overview document with an HTTP GET operation, for example, via a Web browser.

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>useType</td>
<td>optional</td>
</tr>
</tbody>
</table>

The optional useType attribute is used to provide information about the type of document at that URL. One common value used in the useType attribute is “text”. Using this value denotes that the overviewURL contains additional textual information. The content of the useType attribute may contain other values. See Appendix B Using and Extending the useType Attribute for more information.

3.6 tModel Structure

Making it easy to describe Web services in ways that are meaningful enough to be useful during searches is an important goal of UDDI. Another goal is to provide a facility to make these descriptions complete enough that people and programs can discover how to interact with Web services they do not know much about. To do this, there needs to be a way to mark a description with information that designates how it behaves, what conventions it follows, and what specifications or standards the service complies with.

Providing the ability to describe compliance with specifications, concepts, or even shared design is one of the roles that the tModel structure fills.

Each tModel instance is a keyed entity in UDDI. In a general sense, the purpose of tModel entities is to provide a reference system based on abstraction. There are two primary uses for tModel entities: as sources for determining compatibility of Web services and as keyed namespace references.

3.6.1 Common tModel uses

There are several places within a businessEntity that can refer to tModels. References to the same tModel instance can be found in many businessEntity structures. tModel references also occur in various API calls.

Section 3.6 tModel Structure gives an overview of the different types of tModels.

3.6.1.1 Defining the technical fingerprint

One common use for tModel entities is to represent technical specifications or concepts. For example, a tModel can be used to represent a specification that defines wire protocols, interchange formats and interchange sequencing rules. Examples can be seen in the RosettaNet Partner Interface Processes [5] specification, the Open Applications Group Integration Specification [6] and various Electronic Document Interchange (EDI) efforts.
Software that communicates with other software invariably adheres to some pre-agreed specifications. In situations where this is true, the designers of the specifications can establish a unique technical identity within a UDDI registry by publishing information about the specification in a tModel. While the main reason of registering a tModel with a specific UDDI registry is to define its identity, the actual specification or set of documents that describes the concept of a tModel is not a part of the registry and is remotely referenced using the overviewDoc structure. Publishers SHOULD choose well-known formats and description languages for the documents that describe the concept each tModel represents.

Once a tModel is published, other parties can express the availability of Web services that are compliant with a specification the tModel represents by simply including a reference to the tModel – i.e., its tModelKey – in their technical service descriptions bindingTemplate data.

This approach facilitates searching for registered Web services that are compatible with a particular specification. Once the proper tModelKey value is known, it is easy to discover that a particular businessEntity has registered a Web service that references the tModel. In this way, the tModelKey becomes a technical fingerprint that is unique to a given specification.

3.6.1.2 Defining value sets

The second general use for tModel entities is within the identifierBag, categoryBag, address and publisherAssertion structures that are used to specify organizational identity and various categories. Used in this context, a tModel represents the system of values used to identify or categorize UDDI entities.

For example, to represent the fact that a business described by a businessEntity has a particular US Tax identifier, a keyedReference is placed into the identifierBag of the businessEntity. The keyedReference has a keyValue that is the tax ID and refers to the tModel that means “the system of US Tax code identifiers”. Together, the keyValue and the tModel reference specify a particular value in a particular system of values.

3.6.1.3 Defining a find qualifier

The third general use for tModel entities is to represent find qualifiers. Find qualifiers are values that modify how the find_xx APIs work. For example, to cause find_business to sort its results in the order in which they were published, the uddi:uddi.org:findQualifier:sortByDateAsc may be specified. See Section 5.1.4 Find Qualifiers for details.
3.6.2 Structure diagram

3.6.3 Documentation

A given tModel entity is uniquely identified by its tModelKey. When a tModel is published within a UDDI registry, the tModelKey MUST be omitted if the publisher wants the registry to generate a key. When a tModel is retrieved from a UDDI registry, the tModelKey MUST be present.

In retrieved tModel data, the deleted attribute, an information-only field, indicates whether the tModel is logically deleted. The two allowed values for this attribute are “true” and “false”.

Simple textual information about the tModel, potentially in multiple languages, is given by its name and short description. While the tModel has exactly one non-empty name, it can have zero or more descriptions. The name SHOULD be formatted as a URI and, as a consequence, the xml:lang attribute of the name element SHOULD NOT be used. More information about the structure of the name and description elements can be found in Section 3.3 businessEntity structure.

The overviewDoc is an optional repeating element, used to house references to remote descriptive information or instructions related to the tModel. For more information about the structure of the overviewDoc v, see Section 3.5 bindingTemplate Structure.

The optional useType attribute contained in the overviewURL of the overviewDoc is used to provide information about the type of document at that URL.
common value used in the useType attribute is “text”. Using this value denotes that the overviewURL contains additional textual information. Another common value is “wsdlInterface”, which is used to designate that the overviewURL contains a WSDL interface document that can be re-used by many implementations. The content of the useType attribute may contain other values. See Appendix B Using and Extending the useType Attribute for more information.

In addition to the tModelKey that uniquely identifies the tModel within the registry, the identifierBag contains a list of logical identifiers, each valid in its own identifier system. For more information about identifierBags, see Section 3.3 businessEntity Structure.

The categoryBag contains a list of categories that describe specific aspects of the tModel (e.g. its technical type). Each category is valid in its own category system. For more information about categoryBags, see Section 3.3 businessEntity structure.

A tModel entity MAY be digitally signed using XML digital signatures. When a tModel is signed, each digital signature MUST be provided by its own dsig:Signature element. Appendix I Support for XML Digital Signatures covers the use of this element in accordance with the XML-Signature specification.

3.7 publisherAssertion Structure

Many businesses and organizations are not effectively represented by a single businessEntity, because their description and discovery are likely to be diverse. Examples include corporations with a variety of subsidiaries, private exchanges with sets of suppliers and their customers and industry consortiums with their members. An obvious solution is to publish several businessEntity structures. Such a set of businessEntity structures represents a more or less coupled community whose members often would like to make some of their relationships visible in their UDDI registrations. This may be accomplished by using the publisherAssertion structure. To eliminate the possibility that one publisher claims a relationship to another that is not reciprocated, both publishers must publish identical assertions for the relationship to become visible. More detailed information about relationships and publisher assertions is given in Appendix A Relationships and Publisher Assertions.

3.7.1 Structure Diagram

3.7.2 Documentation

The two businessEntity instances between which an assertion is made are uniquely identified by the required fromKey and toKey elements. The keyedReference describes the relationship between the businessEntity elements identified by fromKey and toKey. Similar to the general behavior of a keyedReference in a categoryBag (see full description in Section 3.3 businessEntity Structure), the included tModelKey uniquely identifies the relationship type system and the keyName keyValue pair designate a specific relationship type within this value set. Omitted keyNames are treated as empty keyNames.

A publisherAssertion entity MAY be digitally signed using XML digital signatures. When a publisherAssertion is signed, each digital signature MUST be provided by its own dsig:Signature element. Appendix I Support For XML Digital Signatures covers the use of this element in accordance with the XML-Signature specification.
3.8 operationalInfo Structure

Information about a publishing operation is captured whenever a UDDI core data structure is published. This data includes the date and time that the data structure was created and modified, the identifier of the UDDI node at which the publish operation took place, and the identity of the publisher. Operational information for a UDDI data structure is made accessible using the get_operationalInfo inquiry API. See Section 5.1.16 get_operationalInfo.

The operationalInfo structure is used to convey the operational information for the UDDI core data structures, that is, the businessEntity, businessService, bindingTemplate and tModel structures.

3.8.1 Structure diagram

![OperationalInfo Structure Diagram]

Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityKey</td>
<td>required</td>
</tr>
</tbody>
</table>

3.8.2 Documentation

The UDDI entity with which the operationalInfo is associated is uniquely identified by the required entityKey attribute.

The created element indicates the instant in time at which the entity with which the operationalInfo is associated first appeared in a registry.

The modified element indicates the instant in time at which the entity with which the operationalInfo is associated was last changed by a save operation that may have updated its content. This will initially be equivalent to the value of the created element, but will diverge as changes are made over time.

Some UDDI core data structures are containers of other UDDI core data structures. For instance, businessService elements are contained by businessEntity elements and bindingTemplate elements are contained by businessService elements. Independent changes made to contained entities of such entities (for example, changes to an existing businessService within a businessEntity by means of a save_service API call) do not affect the value of the modified element associated with the containing entity. Instead, the modifiedIncludingChildren element in the containing entity contains the maximum of its own modified element and the modifiedIncludingChildren elements of each of the entities it contains (if any). Changes in a service that is being projected do not affect the
modifiedIncludingChildren element of the businessEntity in which it is projected. The information provided by the modifiedIncludingChildren element is logically redundant, since it can be computed by inquirers. It is, however, commonly desired and is therefore provided directly in pre-computed form.

The degree to which the clocks of each UDDI node used to generate the created, modified, and modifiedIncludingChildren elements are synchronized is not architecturally specified, but rather is a matter of registry policy. Likewise, the frequency with which each clock is incremented (e.g.: 60Hz, 100Hz, etc.) is also a matter of registry policy.

The UDDI node (if any) that has custody of the entity to which an operationalInfo element is attached is identified by the nodeID element. The nodeID contains a unique key that is used to identify this node within a UDDI registry. As described in Section 7.5.2 Configuration of a UDDI Node – operator element for nodes that implement UDDI Replication, this element MUST match the value specified in the Replication Configuration element associated with the node.

A node may provide an indication of the owner of the data corresponding to the entityKey in the authorizedName element. The exact contents of this element and how the authorizedName element should be interpreted depends on the authentication, identification and privacy policies of the registry and node (see Chapter 9, Policy).