uddi.org
Universal Description, Discovery and Integration

UDDI Data Structure Reference V1.0
UDDI Published Specification, 28 June 2002

This version:
http://www.uddi.org/pubs/DataStructure-V1.00-Published-20020628.pdf

Latest version:
http://www.uddi.org/pubs/DataStructure-V1.00-Published-20020628.pdf

Authors (alphabetically):
Toufic Boubez, IBM
Maryann Hondo, IBM
Chris Kurt, Microsoft
Jared Rodriguez, Ariba
Daniel Rogers, Microsoft

Contents

Introduction

The programmatic interface provided for interacting with systems that follow the Universal Description, Discovery and Integration (UDDI) specifications make use of Extensible Markup Language (XML) and a related technology called Simple Object Access Protocol (SOAP), which is a specification for using XML in simple message-based exchanges.

The UDDI Programmer's API Specification defines approximately 30 SOAP messages that are used to perform inquiry and publishing functions against any UDDI-compliant Business Registry. This document outlines the details of each of the XML structures associated with these messages.

Note: This document is a co-requisite to the UDDI XML Schema document.

Service discovery

The purpose of UDDI-compliant registries is to provide a business discovery platform on the World Wide Web. Service discovery is related to being able to advertise and locate information about different technical interfaces exposed by different parties. Services are interesting when you can discover them, determine their purpose, and then have software that is equipped for using a particular type of Web service complete a connection and derive benefit from a service.

A UDDI-compliant registry provides an information framework for describing services exposed by an entity or business. Using this framework the description of a service that is managed by a UDDI registry is information about the service itself. In order to promote cross platform service description that is suitable to a "black-box[1]" Web environment, this description is rendered in cross-platform XML.

Four data structure types

The information that makes up a registration consists of four data structure types. This division by information type provides simple partitions to assist in the rapid location and understanding of the different information that makes up a registration.

The four core types are shown below:

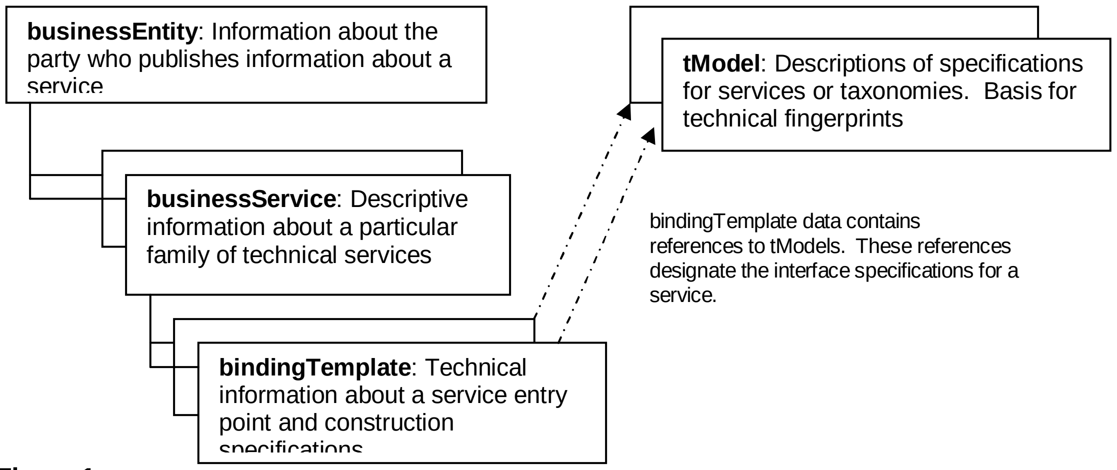
businessEntity: Information about the party who publishes information about a service
businessService: Descriptive information about a particular family of technical services
bindingTemplate: Technical information about a service entry point and construction specifications
tModel: Descriptions of specifications for services or taxonomies. Basis for technical fingerprints
bindingTemplate data contains references to tModels. These references designate the interface specifications for a service.

Figure 1

[1] The term "black box" in this context implies that the descriptive information found in a UDDI compliant registry is provided in a neutral format that allows any kind of service, without regard to a given services platform requirements or technology requirements. UDDI provides a framework for describing any kind of service, and allows storage of as much detail about a service and its implementation as desired.

These four structure types make up the complete amount of information provided within the UDDI service description framework. Each of these XML structures contains a number of data fields[2] that serve a business or technical descriptive purpose. Explaining each of these structures and the meaning and placement of each field is the primary purpose of this document.

These structures are described in the UDDI API Programmer's API Specification and XML Schema. The schema defines approximately 20 requests and 10 responses, each of which contain these structures, references to these structures, or summary versions of these structures. In this document we first explain the core structures, and then provide descriptions of the individual structures used for the request/response XML SOAP interface.

Core structure reference

This section outlines each of the four core structures and provides in-depth technical descriptions of each of the fields contained within each structure.

Overall principles

Each of the four 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 about the Web services they offer, whether these services are entry points for interfacing with accounting systems, or services that allow customers to query the status of a factory order. Each business, and the corresponding service descriptions (both logical and technical descriptions) exist as separate instances of data within a UDDI registry.

Unique identifiers

The individual facts about a business, its services, technical information, or information about specifications for services are kept separate, and are accessed individually by way of unique identifiers, or keys. A UDDI registry assigns these unique identifiers when information is first saved, and these identifiers can be used later as keys to access the specific data instances on demand.

Each unique identifier generated by a UDDI registry takes the form of a Universally Unique ID (UUID). Technically, a UUID is an octet of hexadecimal characters that has been generated according to a very exacting algorithm that is sufficiently precise as to prevent any two UUIDS from ever being generated in duplicate.

Containment

The individual instance data managed by a UDDI registry are sensitive to the parent/child relationships in the XML schema. This same containment relationship is seen in Figure 1 for the core structures. The businessEntity structure contains one or more unique businessService structures. Similarly, individual businessService structures contain specific instances of bindingTemplate data, which in turn contains information that includes pointers to specific instances of tModel structures.

It is important to note that no single instance of a core structure type is ever "contained" by more than one parent structure. This means that only one specific businessEntity structure (identified by its unique key value) will ever contain or be used to express information about a specific instance of a businessService structure (also identified by its own unique key value).

References, on the other hand, operate differently. We can see an example of this in Figure 1 where the bindingTemplate structures contain references to unique instances of tModel structures. References can be repeated within any number of the core typed data instances such that many references to a single unique instance are allowed.

[2] In XML vernacular, fields are called either elements or attributes.

Determining what is a reference to an instance of a core data type and what is a key for a core data type within a specific instance is straightforward. Besides being able to reference this document, which will make the distinction clear in the field-by-field descriptions, you can also use your knowledge that there are four core data types, and that each of these types are key. Thus you know that the businessKey found within the businessEntity structure is a key, and not a reference. Similarly, the serviceKey and bindingKey values found respectively within the businessService and bindingTemplate structures are keys. The same holds true for the tModelKey value found within the tModel structure.

References on the other hand, occur in only two places today. When tModels are referenced, as seen within a bindingTemplate structure, these occur within a list structure designed for the purpose of holding references to tModels. This list, not being one of the four core data structure types, is not keyed as an individual instance. Rather, its own identity is derived from the parent structure that contains it – and it cannot be separated. Thus any key values directly contained in structures that are not themselves one of the four core structure types, are references. Examples include tModelKey values found in lists within bindingTemplate and categorization and identification schemes – in which context the tModel itself represents a uniquely identifiable namespace reference and qualifier.

The businessEntity structure

The businessEntity structure represents all known information about a business or entity that publishes descriptive information about the entity as well as the services that it offers. From an XML standpoint, the businessEntity is the top-level data structure that accommodates holding descriptive information about a business or entity. Service descriptions and technical information are expressed within a businessEntity by a containment relationship.

Structure specification

```
<element name = "businessEntity">
         <type content = "elementOnly">
                  <group order = "seq">
                           <element ref = "discoveryURLs" minOccurs = "0" maxOccurs = "1"/>
                           <element ref = "name"/>
                           <element ref = "description" minOccurs = "0" maxOccurs = "*"/>
                           <element ref = "contacts" minOccurs = "0" maxOccurs = "1"/>
                           <element ref = "businessServices" minOccurs = "0" maxOccurs = "1"/>
                           <element ref = "identifierBag" minOccurs = "0" maxOccurs = "1"/>
                           <element ref = "categoryBag" minOccurs = "0" maxOccurs = "1"/>
                  </group>
                  <attribute name = "businessKey" minOccurs = "1" type = "string"/>
                  <attribute name = "operator" type = "string"/>
                  <attribute name = "authorizedName" type = "string"/>
         </type>
</element>
```

Descriptive matrix

At the top level, a businessEntity contains the following data. Required data is designated by bold field names:

Field Name	Description	Data Type	Length
businessKey	Attribute. This is the unique identifier for a given instance of a businessEntity data set.	UUID	128 bits (hex)
authorizedName	Attribute. This is the recorded name of the individual that published the businessEntity data. This data is calculated by the controlling operator and should not be supplied within save_business operations.	string	64
operator	Attribute. This is the certified name of the UDDI registry site operator that manages the master copy of the businessEntity data. The controlling operator records this data at the time data is saved. This data is calculated and should not be supplied within save_business operations.	string	48
discoveryURLs	Optional element. This is a list of Uniform Resource Locators (URL) that point to alternate, file based service discovery mechanisms. Each recorded businessEntity structure is automatically assigned a URL that returns the individual businessEntity structure. URL search is provided via find_business call.	structure	
name	Required element. This is the name recorded for the businessEntity. Name search is provided via find_business call. Names may not be blank.	string	128
description	Optional repeating element. One or more short business	string	255

	descriptions. One description is allowed per national language code supplied.		
contacts	Optional element. This is an optional list of contact information.	structure	
businessServices	Optional repeating element. This is a list of one or more logical or business service descriptions. If no services are registered with a businessEntity structure for a period of X days, businessEntity data will be subject to clean-up operations.	structure	
identifierBag	Optional element. This is an optional list of name-value pairs that can be used to record identificaton numbers for a businessEntity. These can be used during search via find_business.	structure	
categoryBag	Optional element. This is an optional list of name-value pairs that are used to tag a businessEntity with specific taxonomy information (e.g. industry, product or geographic codes). These can be used during search via find_business.	structure	

Substructure breakdown

DiscoveryURLs

The discoveryURLs structure is used to hold pointers to URL-addressable discovery documents. The expected retrieval mechanism for URLs referenced in the data within this structure is HTTP-GET. The expected return document is not defined. Rather, a framework for establishing convention is provided, and two such conventions are defined within UDDI behaviors. It is the hope that other conventions come about and use this structure to accommodate alternate means of discovery.[3]

Field Name	Description	Data Type	Length
discoveryURL	These are attribute qualified repeating elements holding strings that represent web addressable (via HTTP-GET) discovery documents.	String w/attributes	255

DiscoveryURL

Each individual discovery URL consists of an attribute whose value designates the URL use-type convention, and a string, found within the body of the element. Each time a businessEntity structure is saved via a call to save_business, the UDDI registry node will generate one URL. The generated URL will point to an instance of either a businessEntity or businessEntityExt structure. The useType attribute of the discoveryURL will be set to either "businessEntity" or "businessEntityExt" according to the data type found while processing the save_business message. The discoveryURL collection will be augmented so that it includes this generated URL. This URL can then be used to retrieve a specific instance of a businessEntity. The XML returned will be formatted as a normal businessDetail message.

[3] An example of an alternate form of service discovery is seen in the ECO Framework as defined by the commerce.net initiative. A convention to provide pointers to ECO discovery entry points could take advantage of the structures provided in discoveryURLs by adopting the useType value “ECO”.

Field Name	Description	Data Type	Length
useType	Required attribute that designates the name of the convention that the referenced document follows. Two reserved convention values are “businessEntity” and “businessEntityExt”. URL’s qualified with these values should point to XML documents of the same type as the useType value.	String	20

Example: An example of the generated data for a given businessEntity might look similar to the following:

```
<discoveryURLs>
    <discoveryURL useType=”businessEntity”>
    http://www.someOperator?businessKey=BE3D2F08-CEB3-11D3-849F-0050DA1803C0
    </discoveryURL>
<discoveryURLs>
```

Contacts

The contacts structure provides a way for information to be registered with a businessEntity record so that someone finding the information can make human contact. Because the information held within the UDDI *Registry Sites* is freely available, some care should be taken when considering the amount of contact information to register. Electronic mail addresses in particular may be the greatest concern if you are sensitive to receiving unsolicited mail.

The contacts structure itself is a simple collection of *contact* structures. You’ll find that there are many collections in the UDDI XML schema. Like the *discoveryURLs* structure – which is a container for one or more discoveryURL structures, the *contacts* structure is a simple container where one or more contact structures reside.

Contact

The contact structure lets you record contact information for a person. This information can consist of one or more optional elements, along with a person’s name. Contact information exists by containment relationship alone, and no mechanisms for tracking individual contact instances is provided by UDDI specifications.

Field Name	Description	Data Type	Length
useType	Optional attribute that is used to describe the type of contact in freeform text. Suggested examples include “technical questions”, “technical contact”, “establish account”, “sales contact”, etc.	String	20
description	Optional element. Zero or more language-qualified [4]descriptions of the reason the contact should be used.	string	255

[4] All fields named *description* behave the same way and are subject to the same language identifier rules as described in the XML usage appendix found in the UDDI programmers API specification. Embedded HTML is prohibited in description fields.

personName	Required element. Contacts should list the name of the person or name of the job role that will be available behind the contact. Examples of roles include “administrator” or “webmaster”.	string	32
phone	Optional repeating element. Used to hold telephone numbers for the contact. This string value can be adorned with an optional useType attribute for descriptive purposes. If more than one phone element is saved, useType attributes are required on each.	string	50
email	Optional repeating element. This string value can be adorned with an optional useType attribute for descriptive purposes. If more than one email element is saved, useType attributes are required on each.	string	128
address	Optional repeating element. This structure represents the printable lines suitable for addressing an envelope.	structure	

Address

The address structure is a simple list of AddressLine elements within the address container. Each addressLine element is a simple string. UDDI compliant registries are responsible for preserving the order of any addressLine data provided. Address structures also have two optional attributes for recording the useType (freeform text) and sortCode data. The sortCode values are not significant within a UDDI registry, but may be used by user interfaces that present contact information in some ordered fashion using the values provided in the sortCode attribute.

AddressLine

AddressLine elements contain string data with a suggested line length limit of 40 character positions. Each addressLine element can be adorned with two optional descriptive attributes. Address line order is significant and will always be returned by the UDDI compliant registry in the order originally provided during a call to save_business.

Field Name	Description	Data Type	Length
useType	Optional attribute that is used to describe the type of address in freeform text. Suggested examples include “headquarters”, “sales office”, “billing department”, etc.	String	20
sortCode	Optional attribute that can be used to drive the behavior of external display mechanisms that sort addresses. The suggested values for sortCode include numeric ordering values (e.g. 1, 2, 3), alphabetic character ordering values (e.g. a, b, c) or the first n positions of relevant data within the address.	String	10

businessServices

The businessServices structure provides a way for describing information about families of services. This simple collection accessor contains zero or more businessService structures and has no other associated structures.

identifierBag

The identifierBag element allows business, entity, or tModel data to include information about common forms of identification such as Dun & Bradstreet D-U-N-S® numbers, tax identifiers, etc. This data can be used to signify the identity of the busienssEntity, or can be used to signify the identity of the publishing party. Including data of this sort is optional, but when used greatly enhances the search behaviors exposed via the *find_xx* messages defined in the UDDI Programmers API specification. For a full description of the structures involved in establishing an identity, see the appendix in this document called "Using Identifiers".

categoryBag

The categoryBag element allows businessEntity, businessService and tModel data to be categorized according to any of several available taxonomy based classification schemes. *Operator Registry Sites* automatically provide validated categorization support for three taxonomies that cover industry codes (via NAICS), product and service classifications (via UNSPSC) and geography. Including data of this sort is optional, but when used greatly enhances the search behaviors exposed by the *find_xx* messages defined in the UDDI Programmers API specification. For a full description of structures involved in establishing categorization information, see the appendex in this document called "Using Categorization".

Information included in categoryBag may be used as hints for locating and referencing your published UDDI information. Expect that third party classification services will crawl and index the information found in the UDDI *Registry* Site registry stores, and use the categorization information as a first pass in establishing business level searching facilities. For this reason, it is not necessary to include all possible variants of category information for a given purpose. The sheer volume of data within the UDDI replicated *Registry Sites* makes it impractical as a first source of business level searching using category information.

The businessService structure

The businessService structures each represent a logical service classification. The name of the element includes the term “business” in an attempt to describe the purpose of this level in the service description hierarchy. Each businessService structure is the logical child of a single businessEntity structure. The identity of the containing (parent) businessEntity is determined by examining the embedded businessKey value. If no businessKey value is present, the businessKey must be obtainable by searching for a businessKey value in any parent structure containing the businessService. Each businessService element contains descriptive information in business terms outlining the type of technical services found within each businessService element.

Structure specification

```
<element name = "businessService">
         <type content = "elementOnly">
                  <group order = "seq">
                           <element ref = "name"/>
                           <element ref = "description" minOccurs = "0" maxOccurs = "*"/>
                           <element ref = "bindingTemplates"/>
                           <element ref = "categoryBag" minOccurs = "0" maxOccurs = "1"/>
                  </group>
                  <attribute name = "serviceKey" minOccurs = "1" type = "string"/>
                  <attribute name = "businessKey" type = "string"/>
         </type>
</element>
```

Substructure breakdown

Field Name	Description	Data Type	Length
businessKey	This attribute is optional when the businessService data is contained within a fully expressed parent that already contains a businessKey value. If the businessService data is rendered into XML and has no containing parent that has within its data a businessKey, the value of the businessKey that is the parent of the businessService is required to be provided. This behavior supports the ability to browse through the parent-child relationships given any of the core elements as a starting point.	UUID	128 bits (hex)
serviceKey	This is the unique key for a given businessService. When saving a new businessService structure, pass an empty serviceKey value. This signifies that a UUID value is to be generated. To update an existing businessService structure, pass the UUID value that corresponds to the existing service. If this data is received via an inquiry operation, the serviceKey values may not be blank.	UUID	128 bits (hex)
name	This is a human readable name for this service family. Name values may not be blank.	string	40
description	Optional element. Zero or more language-qualified text descriptions of the logical service family.	string	255
bindingTemplates	This structure holds the technical service description information related to a given business service family.	structure	
categoryBag	Optional element. This is an optional list of name-value pairs that are used to tag a businessService with specific taxonomy information (e.g. industry, product or	structure	

	geographic codes). These can be used during search via find_service. See categoryBag under businessEntity for a full description.		

bindingTemplates

The bindingTemplates structure is a container for one or more bindingTemplate structures. This simple collection accessor has no other associated structure.

The bindingTemplate structure

Technical descriptions of Web services are accommodated via individual contained instances of bindingTemplate structures. These structures provide support for determining a technical entry point or optionally support remotely hosted services, as well as a lightweight facility for describing unique technical characteristics of a given implementation. Support for technology and application specific parameters and settings files are also supported.

Since UDDI's main purpose is to enable description and discovery of Web Service information, it is the bindingTemplate that provides the most interesting technical data.

Each bindingTemplate structure has a single logical businessService parent, which in turn has a single logical businessEntity parent.

Structure specification

```
<element name = "bindingTemplate">
         <type content = "elementOnly">
              <group order = "seq">
                  <element ref = "description" minOccurs = "0" maxOccurs = "*"/>
                  <group order = "choice">
                           <element ref = "accessPoint" minOccurs = "0" maxOccurs = "1"/>
                           <element ref = "hostingRedirector" minOccurs = "0" maxOccurs = "1"/>
                  </group>
                  <element ref = "tModelInstanceDetails"/>
              </group>
              <attribute name = "bindingKey" minOccurs = "1" type = "string"/>
              <attribute name = "serviceKey" type = "string"/>
         </type>
</element>
```

Substructure breakdown

Field Name	Description	Data Type	Length
bindingKey	This is the unique key for a given bindingTemplate. When saving a new bindingTemplate structure, pass an empty bindingKey value. This signifies that a UUID value is to be generated. To update an existing bindingTemplate structure, pass the UUID value that corresponds to the existing bindingTemplate instance. If this data is received via an inquiry operation, the bindingKey values may not be blank.	UUID	128 bits (hex)
serviceKey	This is the unique key for a given businessService. When saving a new businessService structure, pass an empty serviceKey value. This signifies that a UUID value is to be generated. To update an existing businessService structure, pass the UUID value that corresponds to the existing service. If this data is received via an inquiry operation, the serviceKey values may not be blank.	UUID	128 bits (hex)
description	Optional element. Zero or more language-qualified text descriptions of the technical service entry point.	string	255
accessPoint	Required attribute qualified data element[5]. This element is a text field that is used to convey the entry	string	255

[5] One of accessPoint or hostingRedirector is required.

	point address suitable for calling a particular Web service. This may be a URL, 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[6].		
hostingRedirector	Required element if *accessPoint* not provided. This field is redirected reference to a different bindingTemplate. If you query a bindingTemplate and find a hostingRedirector value, you should retrieve that bindingTemplate and use it in place of the one containing the hostingRedirector data.	structure	
tModelInstanceDetails	This structure is a list of tModelInfos. This data, taken in total, should form a distinct fingerprint that can be used to identify compatible services.	structure	

AccessPoint

The accessPoint element is an attribute-qualified pointer to a service entry point. The notion of service at the metadata level seen here is fairly abstract and many types of entry points are accommodated.

A single attribute is provided (named URLType). The purpose of the URLType attribute is to facilitate searching for entry points associated with a particular type of entry point. An example might be a purchase order service that provides three entry points, one for HTTP, one for SMTP, and one for FAX ordering. In this example, we'd find a businessService element that contains three bindingTemplate entries, each with identical data with the exception of the accessPoint value and URLType value.

The valid values for URLType are:

- **mailto**: designates that the accessPoint string is formatted as an electronic mail address reference. (Ex. mailto:purch@fabrikam.com)
- **http**: designates that the accessPoint string is formatted as an HTTP compatible Uniform Resource Locator (URL). (Ex. http://www.fabrikam.com/purchasing)
- **https**: designates that the accessPoint string is formatted as a secure HTTP compatible URL. (Ex. https://www.fabrikam.com/purchasing)
- **ftp**: designates that the accessPoint string is formatted as a writable FTP directory address. (Ex. ftp://ftp.fabrikam.com/public)
- **fax**: designates that the accessPoint string is formatted as a telephone number that will connect to a facsimile machine. (Ex. 1 425 555 5555)
- **phone**: designates that the accessPoint string is formatted as a telephone number that will connect to human or suitable voice or tone response based system. (Ex. 1 425 555 5555)
- **other**: designates that the accessPoint string is formatted as some other address format. When this value is used, one or more of the tModel signatures found in the tModelInstanceInfo collection must imply that a particular format or transport type is required.

[6] The content of the structure named tModelInstanceDetails that is found within a bindingTemplate structure serves as a technical fingerprint. This fingerprint is a series of references to uniquely keyed specifications and/or concepts. To build a new service that is compatible with a tModel, the specifications must be understood. To register a service compatible with a specification, reference a tModelKey within the tModelInstanceDetails data for a bindingTemplate instance.

hostingRedirector

The hostingRedirector element is used to designate that a bindingTemplate entry is a pointer 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 a service is remotely hosted (hence the name of this element), or when many service descriptions could benefit from a single service description.

The hostingRedirector element has a single attribute and no element content. The attribute is a bindingKey value (UUID Hex Octet) that is suitable within the same UDDI registry instance for querying and obtaining the bindingDetail data that is to be used.

There is more information on the hostingRedirector is in the appendices for the UDDI Programmer's API Specification.

tModelInstanceDetails

This structure is a simple accessor container for one or more tModelInstanceInfo structures. When taken as a group, the data that is presented in a tModelInstanceDetails structures forms a technically descriptive fingerprint by virtue of the unordered list of tModelKey references contained within this structure. What this means in English is that when someone registers a bindingTemplate (within a businessEntity data set), it will contain one or more references to specific and identifiable specifications that are implied by the tModelKey values provided with the registration. During an inquiry for a service, an interested party could use this information to look for a specific bindingTemplate that contains a specific tModel reference, or even a set of tModel references. By registering a specific fingerprint in this manner, a software developer can readily signify that they are compatible with the specifications implied in the tModelKeys exposed in this manner.

tModelInstanceInfo

A tModelInstanceInfo structure represents the bindingTemplate instance specific details for a single tModel by reference.

Field Name	Description	Data Type	Length
tModelKey	Required Attribute. This is the unique key reference that implies that the service being described has implementation details that are specified by the specifications associated with the tModel that is referenced	string	255 128 bits (Hex GUID)[7]
description	Optional repeating element. This is one or more language qualified text descriptions that designate what role a tModel reference plays in the overall service description.	string	255
instanceDetails	Optional element. This element can be used when tModel reference specific settings or other descriptive information are required to either describe a tModel specific component of a service description or support services that require additional technical data support (e.g. via settings or other handshake operations)	structure	

[7] The data type for tModelKey allows for using URN values in a later revision. In the beta release, the key is a generated GUID. Design work around managing duplicate urn claims will allow user supplied URN keys on tModels in the future.

InstanceDetails

This structure holds service instance specific information that is required to either understand the service implementation details relative to a specific tModelKey reference, or to provide further parameter and settings support. If present, this element should not be empty. Because no single contained element is required in the schema description, this rule is called out here for clarity.

Field Name	Description	Data Type	Length
description	Optional repeating element. This language-qualfied text element is intended for holding a description of the purpose and/or use of the particular instanceDetails entry.	string	255
overviewDoc	Optional element. Used to house references to remote descriptive or instructions related to proper use of a bindingTemplate technical sub-element.	Structure	
instanceParms	Optional data element. Used to contain settings parameters or a URL reference to a file that contains settings or parameters required to use a specific facet of a bindingTemplate description. If used to house the parameters themselves, the suggested content is a namespace qualified XML string – using a namespace outside of the UDDI schema. If used to house a URL pointer to a file, the suggested format is URL that is suitable for retrieving the settings or parameters via HTTP-GET.	string	

overviewDoc

This optional structure is provided as a placeholder for metadata that describes overview information about a particular tModel use within a bindingTemplate.

Field Name	Description	Data Type	Length
description	Optional repeating element. This language-qualified string is intended to hold a short descriptive overview of how a particular tModel is to be used.	string	255
overviewURL	Optional element. This string data element is to be used to hold a URL reference to a long form of an overview document that covers the way a particular tModel specific reference is used as a component of an overall web service description. The suggested format is a URL that is suitable for retrieving an HTML based description via a web browser or HTTP-GET operation.	string	255

The tModel structure

Being able to describe a Web service and then make the description meaningful enough to be useful during searches is an important UDDI goal. Another goal is to provide a facility to make these descriptions useful enough to learn about how to interact with a service that you don't know much about. In order to do this, there needs to be a way to mark a description with information that designates how it behaves, what conventions it follows, or what specifications or standards the service is compliant with. Providing the ability to describe compliance with a specification, concept, or even a shared design is one of the roles that the tModel structure fills.

The tModel structure takes the form of keyed metadata (data about data). In a general sense, the purpose of a tModel within the UDDI registry is to provide a reference system based on abstraction. Thus, the kind of data that a tModel represents is pretty nebulous. In other words, a tModel registration can define just about anything, but in the current revision, two conventions have been applied for using tModels as sources for determining compatibility and as keyed namespace references.

The information that makes up a tModel is quite simple. There's a key, a name, an optional description, and then a URL that points somewhere – presumably somewhere where the curious can go to find out more about the actual concept represented by the metadata in the tModel itself.

Two main uses

There are two places within a businessEntity registration that you'll find references to tModels. In this regard, tModels are special. Whereas the other data within the businessEntity (e.g. businessService and bindingTemplate data) exists uniquely with one uniquely keyed instance as a member of one unique parent businessEntity, tModels are used as references. This means that you'll find references to specific tModel instances in many businessEntity data sets.

Defining the technical fingerprint

The primary role that a tModel plays is to represent a technical specification. An example might be a specification that outlines wire protocols, interchange formats and interchange sequencing rules. One such specification can be seen in the RosettaNet RNIF 1.0 specification. Other examples can be found in standards efforts such as ebXML[8], ECO[9] and various Electronic Document Interchange (EDI) efforts.

Software that communicates with other software across some communication medium 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 registering information about the specification in a tModel.

Once registered in this way, other parties can express the availability of Web services that are compliant with a specification by simply including a reference to the tModel identitifier (called a tModelKey) in their technical service descriptions bindingTemplate data.

This approach facilitates searching for registered Web services that are compatible with a particular specification. Once you know the proper tModelKey value, you can find out whether a particular business or entity has registered a Web service that references that tModel key. In this way, the tModelKey becomes a technical fingerprint that is unique to a given specification.

Defining an abstract namespace reference

The other place where tModel references is used is within the identifierBag and categoryBag structures that are used to define organizational identity and various classifications. Used in this context, the

[8] OASIS – see xml.org

[9] Eco Framework – see commerce.net

tModel reference represents a relationship between the keyed name-value pairs to the super-name, or namespace within which the name-value pairs are meaningful.

An example of this can be seen in the way a business or entity can express the fact that their U.S. tax code identifier (which they are sure they are known by to their partners and customers) is a particular value. To do this, let's assume that we find a tModel that is named "US Tax Codes", with a description "United States business tax code numbers as defined by the United States Internal Revenue Service". In this regard, the tModel still represents a specific concept – but instead of being a technical specification, it represents a unique area within which tax code ID's have a particular meaning.

When the meaning is established, a business can use the tModelKey for the tax code tModel as a unique reference that qualifies the remainder of the data that makes up an entry in the identifierBag data. *Operator Registry Sites* have registered a number of useful tModels, including U.S. Tax Codes, NAICS (an industry code taxonomy), UNSPSC (a product and service code taxonomy), and a geographic taxonomy to be determined.

Structure specification

```
<element name = "tModel">
         <type content = "elementOnly">
                  <group order = "seq">
                           <element ref = "name"/>
                           <element ref = "description" minOccurs = "0" maxOccurs = "*"/>
                           <element ref = "overviewDoc" minOccurs = "0" maxOccurs = "1"/>
                           <element ref = "identifierBag" minOccurs = "0" maxOccurs = "1"/>
                           <element ref = "categoryBag" minOccurs = "0" maxOccurs = "1"/>
                  </group>
                  <attribute name = "tModelKey" minOccurs = "1" type = "string"/>
                  <attribute name = "operator" type = "string"/>
                  <attribute name = "authorizedName" type = "string"/>
         </type>
</element>
```

Substructure breakdown

Field Name	Description	Data Type	Length
tModelKey	Required Attribute. This is the unique key for a given tModel structure. When saving a new tModel structure, pass an empty tModelKey value. This signifies that a UUID value is to be generated[10]. To update an existing tModel structure, pass the tModelKey value that corresponds to an existing tModel instance.	string	255
authorizedName	Attribute. This is the recorded name of the individual that published the tModel data. This data is calculated by the controlling operator and should not be supplied within save_tModel operations.	string	64
operator	Attribute. This is the certified name of the UDDI registry site operator that manages the master copy of the tModel data. The controlling operator records this data at the time data is saved. This data is calculated and should not be supplied within save_tModel operations.	string	48
name	Required element. This is the name recorded for the tModel. Name search is provided via find_tModel call.	string	128

[10] In the beta release (September 2000), tModelKey values will be generated as UUID strings. Subsequent work will focus on defining the tModel keys to be more useful by using URN/URI values that are supplied by the data owners. Process specifics around URN values prevented this feature from being part of the beta. Facilities to convert references to URN/URI values will be provided at the appropriate time.

	Names may not be blank, and should be meaningful to someone that looks at the tModel		
description	Optional repeating element. One or more short language-qualified descriptions. One description is allowed per national language code supplied.	string	255
overviewDoc	Optional element. Used to house references to remote descriptive or instructions related to the tModel. See the substructure breakdown for Overview doc in the bindingTemplate section of this document.	Structure	
identifierBag	Optional element. This is an optional list of name-value pairs that can be used to record identificaton numbers for a tModel. These can be used during search via find_tModel. See the full description of of this element in the businessEntity section of this document and in the appendix “Using Identifiers”	structure	
categoryBag	Optional element. This is an optional list of name-value pairs that are used to tag a tModel with specific taxonomy information (e.g. industry, product or geographic codes). These can be used during search via find_tModel. See the full description of of this element in the businessEntity section of this document and in the appendix “Using Categorization”	structure	

Appendix A: Using Identifiers

The identifier dilemma

One of the design goals associated with the UDDI registration data is the ability to mark information with identifiers. The purpose of identifiers in the UDDI registration data is to allow others to find the published information using more formal identifiers such as Dun & Bradstreet D-U-N-S® Numbers, tax identifiers, or any other kind of organizational identifiers, regardless of whether these are private or shared.

When you look at an identifier, such as a D-U-N-S® Number, it is not always immediately apparent what the identifier represents. For instance, consider the following identifier:

`123-45-6789`

Standing alone, we could try and guess what this combination of digits and formatting characters implies. However, if we knew that this was a United States Social Security number, we would then have a better context and understand that the meaning, while still not clear, at least identifies one or more persons, perhaps even a living one. Expressed as a name / value pair, the identifier might then look like the following:

`United States Social Security Number, 123-45-6789`

Even with this new information, a search mechanism based on loosely qualified pairs (name of identifier type, identifier value), two different parties might spell or format either part of the information differently, and with the end result being a diminished value for searching.

The goal, of course, is to define a simple mechanism that disambiguates the conceptual meanings behind identifiers and exposes them in ways that are reliable and predictable enough to use, and yet are simple enough structurally to be easy to understand and extend.

Identifier characteristics

When we look at various types of simple identifiers, some common desirable characteristics become evident. In general terms, a system of identifiers that are used to facilitate searching need to be:

- **Resolvable**: Identifiers can be used in a way that allows the meaning of the identifier to be determined. For instance, a popular business identifier mechanism is provided by Dun & Bradstreet in the form of D-U-N-S® numbers. When you know an organization's D-U-N-S® Number, you can use this to reliably distinguish one organization from another.
- **Distinguishable**: Identifiers can be used in a way that you can tell what kind of identifier is being used, or you can specify what kind of identifier you are using to search for something. This means you can tell that two identifiers are the same kind of identifier or are different types (e.g. two D-U-N-S® Numbers, versus a tax identifier or an organizational membership number.)
- **Extensible**: The way that searchable identifiers are defines should be easy to extend so that anyone can register another type of identifier without having to create costly or difficult infrastructure. The search mechanisms that use Identifiers should be able to accommodate newly registered types without any changes to software, and anyone should be able to start using the new types immediately.

With this in mind, let’s look at the way that identifiers are used in the UDDI data structures.

Using identifiers

Instead of defining a simple property where you could attach a keyword or a simple identifier field, UDDI defines the notion of annotating or attaching identifiers to data. Two of the core data types specified by UDDI provide a structure to support attaching identifiers to data. These are the businessEntity and the tModel structures. By providing a placeholder for attaching identifiers to these two root data types, any number of identifiers can be used for a variety of purposes.

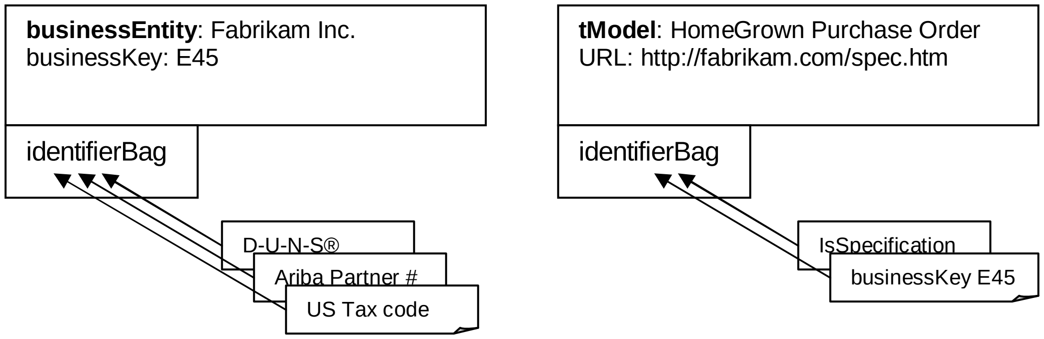

Figure 2

In Figure 2 we see that businessEntity and tModel structures both have a placeholder element named identifierBag[11]. This structure is a general-purpose placeholder for any number of distinct identifiers. In this example, we see five types of identifiers in use in a way that accommodates the kinds of searching that might be required to locate businesses or tModels.

For instance, it is likely that someone who wants to find the types of technical Web services that are exposed by a given business would search by a business identifier. Used in this way, identifiers can represent business identifier types. In the example shown in figure 2, we see that the individual who registered the businessEntity data specified a D-U-N-S® Number, an Ariba partner number, and a US Tax Code identifier[12].

On the other hand, since a tModel is a fairly abstract concept, I might care more that a tModel represents an identifier, and that it was registered by a particular businessEntity. In the example shown in figure 2, we have shown some more abstract identifier types and can tell that the tModel that describes the way that Fabrikam's purchasing Web service has been marked with information that identifies the data as being related to the businessEntity record with the theoretical businessKey value E45. A second identifier marks the tModel as a specification.

Structure specification

```
<element name = "identifierBag">
         <type content = "elementOnly">
                  <group order = "seq">
                           <element ref = "keyedReference" minOccurs = "0" maxOccurs = "*"/>
                  </group>
         </type>
</element>
```

From this structure definition we see that an identifier bag is an element that holds zero or more instances of something called a keyedReference. When we look at that structure, we see:

```
<element name = "keyedReference">
         <type content = "empty">
                  <attribute name = "tModelKey" type = "string"/>
                  <attribute name = "keyName" minOccurs = "1" type = "string"/>
                  <attribute name = "keyValue" minOccurs = "1" type = "string"/>
```

[11] The term "bag" is from the object design naming convention that places collections of like things within an outer container. From outside, it behaves like a bag – that is has a collection of things in it. To see what's in it, you have to look inside.

[12] In the diagram, the actual name/value properties were abbreviated for the sake of simplicity.

```
          </type>
</element>
```

Upon examining this, we see a general-purpose structure for a name-value pair, with one curious additional reference to a tModel structure. It is this extra (optional) attribute that makes the identifier scheme extensible by allowing tModels to be used as conceptual namespace qualifiers.

Understanding this, it then should be easy to see how the example in figure two functioned. Assuming that the identifiers were fully defined, each of the five types shown would each reference one of 5 other tModels. Using the information we've learned already from the discussion of the tModel structure in this document and related texts, we should then be able to see how the tModel structure is useful as a general purpose concept registry with specific UDDI emphasis on the concepts of software specifications, identification schemes, and as we see in the next appendix, as a way to define a general taxonomy namespace key.

The net result is that you can register a tModel to represent an idea, and then use a reference to that tModel as part of a general discovery mechanism that allows unknown facts to be discovered and explained.

Appendix B: Using categorization

Categorization and the ability to voluntarily assign category information to data in the UDDI distributed registry was a key design goal. Without categorization and the ability to specify that information be tangentially related to some well-known industry, product or geographic categorization code set, locating data within the UDDI registry would prove to be too difficult.

At the same time, it is impractical to assume that the UDDI registry will be useful for general-purpose business search. With a projected near-term population of several hundred thousand to million distinct entities, the notion of searching for businesses that satisfy a particular set of criteria will yield a manageably sized result set is unlikely. For example, suppose we searched for all of the business information for businesses that have classified themselves with a particular industry code – retail. Even if we searched within this specific industry classification, the breadth of the category makes it likely that we'll find tens of thousands of companies that are retailers or in some way think of themselves as belonging to a retail category.

Secondary considerations include the accuracy with which categories are applied and the exact value match nature of the UDDI categorization facility. When you register a specific category along with your UDDI registration data, only people searching for that exact category will find your results. For example, in the case where one business marks itself as "retail – pet-food", and another simply uses "retail", the specialization and generalization across categories of any particular categorization scheme or taxonomy is not known to the UDDI search facility.

More intelligent search facilities are required that have some apriori knowledge of the meanings of specific categories and that provide the ability to cross-reference across related categories. Such is the role of more traditional search engines. The design of UDDI allows simplified forms of searching and allows the parties that publish data about themselves, and their advertised Web services to voluntarily provide categorization data that can be used by richer search facilities that will be created above the UDDI technical layer.

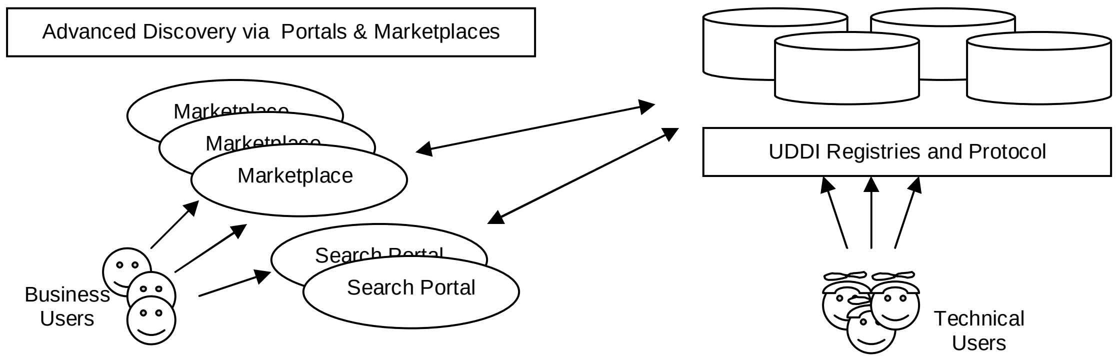

Figure 1B

In Figure 1B we see the tiered search concept illustrated. The role of search portals and marketplaces will support the business level search facilities for such activities as finding partners with products in a certain price range or availability, or finding high quality partners with good reputations. The data in UDDI is not sufficient to accommodate this because of the cross category issues associated with high volumes and voluntary classification.

Structure Specification

```
<element name = "categoryBag">
         <type content = "elementOnly">
                  <group order = "seq">
                           <element ref = "keyedReference" minOccurs = "0" maxOccurs = "*"/>
                  </group>
         </type>
</element>
```

From this structure definition we see that categoryBag is an element that holds zero or more instances of keyedReference elements. This was described in the section on identifiers (Appendix A) and the basic structure is used in the same way.

The key difference is that while the contents of the identifierBag entries are checked only for a valid tModelKey reference (if supplied), extra validation is applied to the references found within the category Bag element. The thinking that drives this is that there may be a wish on the part of the organizations that publish taxonomies to limit or prevent certain parties from using the taxonomy or from using certain categories. For this reason, the *Operator Registry Sites* that implement the UDDI specifications are required to check with a Web service that validates the categorization data associated with a registration.

Initially, three categorization taxonomies have been identified and made a core part of the UDDI *Operator* registries. These are the North American Industry Classification System (NAICS), Universal Standard Products and Services Classification (UNSPSC), and a geographic taxonomy to be determined. A fourth category is also defined – named "Other" – for general-purpose keyword type classification[13].

[13] *Operator Sites* are allowed to promote invalid category entries, or entries that are otherwise rejected by the category classification services, to this Other classification.

Appendix C: Response message reference

All of the messages defined in the Programmer's API Specification return response messages upon successful completion. These structures are defined here for reference purposes. All of the structures shown will appear within SOAP 1.1 compliant envelope structures according the specifications described in the UDDI Programmers API specification. Only the SOAP <body> element contents are shown in the examples in this section.

authtoken

This message returns the authentication information that should be used in subsequent calls to the publishers API messages.

Sample

```
<authToken generic="1.0" operator="uddi.someoperator" xmlns="urn:uddi-org:api" >
         <authInfo>some opaque token value</authInfo>
</authToken>
```

The authToken message contains a single authInfo element that contains an access token that is to be passed back in all of the publisher API messages that change data. This message is always returned using SSL encryption as a synchronous response to the get_authToken message.

bindingDetail

This message returns specific bindingTemplate information in response to a get_bindingDetail or find_binding inquiry message.

Sample

```
<bindingDetail generic="1.0" operator="uddi.someoperator" truncated="true"
                   xmlns="urn:uddi-org:api">
         <bindingTemplate bindingKey="F5E65…" serviceKey="E4D6…" >
                   …
         </bindingTemplate>
         [<bindingTemplate/>…]
</bindingDetail>
```

In this message, one or more bindingTemplate structures are returned according to the data requested in the request message. The serviceKey attributes are always returned when bindingTemplate data is packaged in this way. The truncated flag shown in the example indicates that not all of the requested data was returned due to an unspecified processing limit. Ordinarily, the truncated flag is not included unless the result set has been truncated.

businessDetail

This message returns one or more complete businessEntity data sets in response to a get_businessDetail inquiry message.

Sample

```
<businessDetail generic="1.0" operator="uddi.sourceOperator" truncated="true"
                   xmlns="urn:uddi-org:api">
         <businessEntity businessKey="F5E65…" authorizedName="J. Doe"
                   operator="uddi.publishingOperator" >
                   …
         </businessEntity>
         [<businessEntity/>…]
</businessDetail>
```

In this message, we see that the businessEntity contains the proper output information (e.g. authorizedName, and operator). The two *operator* attributes shown in the businessDetail element and the businessEntity element reflect the distinguished name of the *Operator Registry Site* providing the response message and the distinguished name of the operator where the data is controlled, respectively. Additionally, notice the name of the person who registered the data shown in the *authorizedName* attribute.

businessDetailExt

This message returns one or more complete businessEntity data sets in response to a get_businessDetailExt inquiry message. This is the same data returned by the businessDetail messages, but is provided for consistency with third party extensions to businessEntity information.

Sample

```
<businessDetailExt generic="1.0" operator="uddi.sourceOperator" truncated="true"
                   xmlns="urn:uddi-org:api">
         <businessEntityExt>
                   <businessEntity businessKey="F5E65…" authorizedName="J. Doe"
                            operator="uddi.publishingOperator" >
                            …
                   </businessEntity>
         <businessEntityExt>
         [<businessEntityExt/>…]
</businessDetail>
```

The message API design allows third party registries (e.g. non-operator sites) to implement the UDDI API Specifications while at the same time extending the details collected in a way that will not break tools that are written to UDDI specifications. *Operator Registry Sites* are required to support the *Ext* form of the businessDetail message for compatibility with tools, but are not allowed to manage extended data.

businessList

This message returns one or more businessInfo data sets in response to a find_business inquiry message. BusinessInfo structures are abbreviated versions of businessEntity data suitable for populating lists of search results in anticipation of further “drill-down” detail inquiries.

Sample

```
<businessList generic="1.0" operator="uddi.sourceOperator" truncated="true"
                   xmlns="urn:uddi-org:api">
         <businessInfos>
                   <businessInfo businessKey="F5E65…" >
                            <name>My Company</name>
                            <serviceInfos>
                                     <serviceInfo serviceKey="3D45…">
                                              <name>Purchase Orders</name>
                                     </serviceInfo>
                            </serviceInfos>
                   </businessInfo>
                   [<businessInfo/>…]
         <businessInfos>
</businessList>
```

This message returns overview data in the form of one or more businessInfo structures. Each businessInfo structure contains company name and optional description data, along with a collection element named serviceInfos that in turn can contain one or more serviceInfo structures[14]. Notice that

[14] Refer to the UDDI schema for structure details.

the businessKey attribute is not expressed in the serviceInfo structure due to the fact that this information is available from the containing businessInfo structure.

registeredInfo

This message returns overview information that is suitable for identifying all of the data published by the requester.   Provided as part of the publishers API message set, this information is only provided when requested via a get_registeredInfo message over an SSL connection.

Sample

```
<registeredInfo generic="1.0" operator="uddi.sourceOperator" [truncated="false"]
                   xmlns="urn:uddi-org:api">
         <businessInfos>
                   <businessInfo businessKey="F5E65…" >
                            <name>My Company</name>
                            <serviceInfos>
                                     <serviceInfo serviceKey="3D45…">
                                              <name>Purchase Orders</name>
                                     </serviceInfo>
                            </serviceInfos>
                   </businessInfo>
                   [<businessInfo/>…]
         <businessInfos>
         <tModelInfos>
                   <tModelInfo tModelKey="34D5…">
                            <name>Proprietary XML purchase order</name>
                   </tModelInfo>
                   [<tModelInfo/>…]
         </tModelInfos>
</registeredInfo>
```

This message contains overview data about all of the types of information published by a given publisher. This information is sufficient for driving tools that display lists of registered information and then provide drill-down features. This is the recommended structure for use after a network problem results in an unknown status of saved information.

serviceDetail

This message returns one or more complete businessService structures in response to a get_serviceDetail inquiry message.

Sample

```
<serviceDetail generic="1.0" operator="uddi.sourceOperator" [truncated="false"]
                   xmlns="urn:uddi-org:api">
         <businessService businessKey="F5E65…" serviceKey="3D21…">
                   …
         </businessService>
         [<businessService/>…]
</serviceDetail>
```

One can use serviceDetail messages to get complete descriptive and technical details about registered services by providing one or more serviceKey values in the get_serviceDetail message. Notice that the businessKey value is expressed in this message because the container does not provide a link to the parent businessEntity structure.

serviceList

This message returns one or more serviceInfo data sets in response to a find_service inquiry message.

Sample

```
<serviceList generic="1.0" operator="uddi.sourceOperator" [truncated="false"]
                   xmlns="urn:uddi-org:api">
         <serviceInfos>
                   <serviceInfo serviceKey="3D45…" businessKey="2E4C…">
                            <name>Purchase Orders</name>
                   </serviceInfo>
         </serviceInfos>
</serviceList>
```

This data is suitable for populating a list of services associated with a business and that match a pattern as specified in the inputs to the find_service message. Notice that the businessKey attribute is expressed in the serviceInfo elements found in this message. This is because this information is not available from a containing element.

tModelDetail

This message returns one or more complete tModel data sets in response to a get_tModelDetail inquiry message.

Sample

```
<tModelDetail generic="1.0" operator="uddi.sourceOperator" [truncated="false"]
                   xmlns="urn:uddi-org:api">
         <tModel tModelKey="F5E65…" authorizedName="J. Doe" operator="uddi.publishingOperator" >
                   …
         </tModel>
         [<tModel/>…]
</tModelDetail>
```

Because tModel structures are top-level data (e.g. able to stand alone with no parent containers) the authorizedName value is expressed. This is the name of the person whose account was used to register the data. The two expressions of the *operator* attribute each express the distinguished name of the *Operator Registry Site* that is providing the data and the operator where the data is managed.

tModelList

This message returns one or more abbreviated tModelInfo data sets in response to a find_tModel inquiry message.

Sample

```
<tModelList generic="1.0" operator="uddi.sourceOperator" [truncated="false"
                   xmlns="urn:uddi-org:api">
         <tModelInfos>
                   <tModelInfo tModelKey="34D5…">
                            <name>Proprietary XML purchase order</name>
                   </tModelInfo>
                   [<tModelInfo/>…]
         </tModelInfos>
</tModelList>
```

This data is suitable for finding candidate tModels, populating lists of results and then providing drill-down features that rely on the get_xxDetail messages.

Appendix D: Data Field Lengths

The following table summarizes all known stored element and attribute names based on the names of the fields defined in the XML schema. These are the storage length limits for information that is saved in the UDDI Registry. The *Operator Registry Sites* will truncate data that exceeds these lengths. Fields that are generated by the Operator Registry site (ignored on input) are not listed. Keys are listed even though they are generated. Since keys are referenced by other structures, they are shown here.

Field Name	
accessPoint	255
addressLine	40
authInfo	4096
authorizedName	64
bindingKey	41
businessKey	41
description	255
discoveryURL	255
email	128
hostingRedirector	41
instanceParms	255
keyName	128
keyType	16
keyValue	128
name	128
overviewUrl	255
personName	32
phone	50
serviceKey	41
sortCode	10
tModelKey	255
uploadRegister	255
URLType	16
useType	32