

# UDDI Version 2.0 Data Structure Reference

## UDDI Open Draft Specification 8 June 2001

**This version:**

<http://www.uddi.org/pubs/DataStructure-V2.00-Open-20010608.pdf>

**Latest version:**

<http://www.uddi.org/pubs/DataStructure-V2.00-Open-20010608.pdf>

**Editors (alphabetically):**

David Ehnebuske, IBM  
Dan Rogers, Microsoft  
Claus von Riegen, SAP

**Contributors (alphabetically):**

Tom Bellwood, IBM  
Andy Harris, i2 Technologies  
Denise Ho, Ariba  
Yin-Leng Husband, Compaq  
Alan Karp, HP  
Keisuke Kibakura, Fujitsu  
Jeff Lancelle, Verisign  
Sam Lee, Oracle  
Sean MacRoibeaird, Sun  
Barbara McKee, IBM  
Tammy Nordan, Compaq  
Dan Rogers, Microsoft  
Christine Tomlinson, Sun  
Cafer Tosun, SAP

Copyright © 2001 by Accenture, Ariba, Inc., Commerce One, Inc., Compaq Computer Corporation, Equifax, Inc., Fujitsu Limited, Hewlett-Packard Company, i2 Technologies, Inc., Intel Corporation, International Business Machines Corporation, Microsoft Corporation, Oracle Corporation, SAP AG, Sun Microsystems, Inc., and VeriSign, Inc. All Rights Reserved.

These documents are provided by the companies named above ("Licensors") under the following license. By using and/or copying this document, or the document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and distribute the contents of this document, or the document from which this statement is linked, in any medium for any purpose and without fee or royalty under copyrights is hereby granted, provided that you include the following on ALL copies of the document, or portions thereof, that you use:

1. A link to the original document.
2. An attribution statement: "Copyright © 2001 by Accenture, Ariba, Inc., Commerce One, Inc., Compaq Computer Corporation, Equifax, Inc., Fujitsu Limited, Hewlett-Packard Company, i2 Technologies, Inc., Intel Corporation, International Business Machines Corporation, Microsoft Corporation, Oracle Corporation, SAP AG, Sun Microsystems, Inc., and VeriSign, Inc. All Rights Reserved."

If the Licensors own any patents or patent applications which that may be required for implementing and using the specifications contained in the document in products that comply with the specifications, upon written request, a non-exclusive license under such patents shall be granted on reasonable and non-discriminatory terms.

THIS DOCUMENT IS PROVIDED "AS IS," AND LICENSORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

LICENSORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

# Contents

<b>1</b>	<b>TERMINOLOGY .....</b>	<b>5</b>
<b>2</b>	<b>INTRODUCTION.....</b>	<b>5</b>
2.1	SERVICE DISCOVERY .....	5
2.1.1	Five data types.....	5
<b>3</b>	<b>OVERALL DESIGN PRINCIPLES .....</b>	<b>6</b>
3.1	UNIQUE IDENTIFIERS.....	6
3.2	CONTAINMENT.....	7
<b>4</b>	<b>DATA STRUCTURE NOTATION.....</b>	<b>7</b>
<b>5</b>	<b>THE BUSINESSENTITY STRUCTURE .....</b>	<b>8</b>
5.1	STRUCTURE SPECIFICATION.....	8
5.2	SUBSTRUCTURE BREAKDOWN .....	8
5.2.1	discoveryURLs.....	9
5.2.1.1	discoveryURL.....	9
5.2.2	name.....	10
5.2.3	contacts .....	10
5.2.3.1	contact.....	10
5.2.3.2	address.....	11
5.2.3.3	addressLine .....	12
5.2.4	businessServices .....	12
5.2.5	identifierBag .....	12
5.2.6	categoryBag.....	13
<b>6</b>	<b>THE BUSINESSSERVICE STRUCTURE.....</b>	<b>14</b>
6.1	STRUCTURE SPECIFICATION .....	14
6.2	SUBSTRUCTURE BREAKDOWN .....	14
6.2.1	bindingTemplates .....	15
<b>7</b>	<b>THE BINDINGTEMPLATE STRUCTURE .....</b>	<b>16</b>
7.1	STRUCTURE SPECIFICATION.....	16
7.2	SUBSTRUCTURE BREAKDOWN .....	16
7.2.1	accessPoint .....	17
7.2.2	hostingRedirector.....	18
7.2.3	tModelInstanceDetails .....	18
7.2.3.1	tModelInstanceInfo.....	18
7.2.3.2	instanceDetails .....	19
7.2.3.3	overviewDoc .....	19
<b>8</b>	<b>THE TMODEL STRUCTURE.....</b>	<b>21</b>
8.1	TWO MAIN USES .....	21
8.1.1	Defining the technical fingerprint.....	21
8.1.2	Defining an abstract namespace reference.....	21
8.2	STRUCTURE SPECIFICATION.....	22
8.3	SUBSTRUCTURE BREAKDOWN.....	22
<b>9</b>	<b>THE PUBLISHERASSERTION STRUCTURE.....</b>	<b>24</b>
9.1	STRUCTURE SPECIFICATION .....	24
9.2	SUBSTRUCTURE BREAKDOWN .....	24
<b>10</b>	<b>APPENDIX A: USING IDENTIFIERS .....</b>	<b>25</b>
10.1	THE IDENTIFIER DILEMMA .....	25

10.2 IDENTIFIER CHARACTERISTICS .....25

    10.2.1 Using identifiers.....26

    10.2.2 Structure Specification .....27

**11 APPENDIX B: USING CATEGORIZATION .....28**

    11.1 STRUCTURE SPECIFICATION .....28

**12 APPENDIX C: RESPONSE MESSAGE REFERENCE .....30**

    12.1 ASSERTIONSTATUSREPORT .....30

        12.1.1 Sample.....30

    12.2 AUTHTOKEN.....30

        12.2.1 Sample.....30

    12.3 BINDINGDETAIL .....31

        12.3.1 Sample.....31

    12.4 BUSINESSDETAIL.....31

        12.4.1 Sample.....31

    12.5 BUSINESSDETAILEXT .....31

        12.5.1 Sample.....31

    12.6 BUSINESSLIST.....32

        12.6.1 Sample.....32

    12.7 PUBLISHERASSERTIONS .....32

        12.7.1 Sample.....32

    12.8 REGISTEREDINFO .....33

        12.8.1 Sample.....33

    12.9 RELATEDBUSINESSESLIST .....33

        12.9.1 Sample.....33

    12.10 SERVICEDetail .....34

        12.10.1 Sample .....34

    12.11 SERVICEList .....34

        12.11.1 Sample .....34

    12.12 TMODELDETAIL .....34

        12.12.1 Sample .....35

    12.13 TMODELList .....35

        12.13.1 Sample .....35

**13 APPENDIX D: DATA FIELD LENGTHS.....36**

**14 APPENDIX E: STRUCTURED ADDRESS EXAMPLE .....37**



# 1 Terminology

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

# 2 Introduction

The programmatic interface provided for interacting with systems that follow the Universal Description Discovery & 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 Version 2.0 API Specification defines approximately 40 SOAP messages that are used to perform inquiry and publishing functions against any UDDI compliant service registry. This document outlines the details of each of the XML structures associated with these messages.

## 2.1 Service Discovery

The purpose of UDDI compliant registries is to provide a service 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 any entity or business. In order to promote cross platform service description that is suitable to a “black-box<sup>1</sup>” Web environment, this description is rendered in cross-platform XML.

### 2.1.1 Five data types

The information that makes up a registration consists of five 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 five core types are shown in figure 1.

These five 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<sup>2</sup> that serve either 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 defined in the UDDI Version 2.0 API schema. The schema defines approximately 25 requests and 15 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.

---

<sup>1</sup> 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.

<sup>2</sup> In XML vernacular, fields are called either elements or attributes.

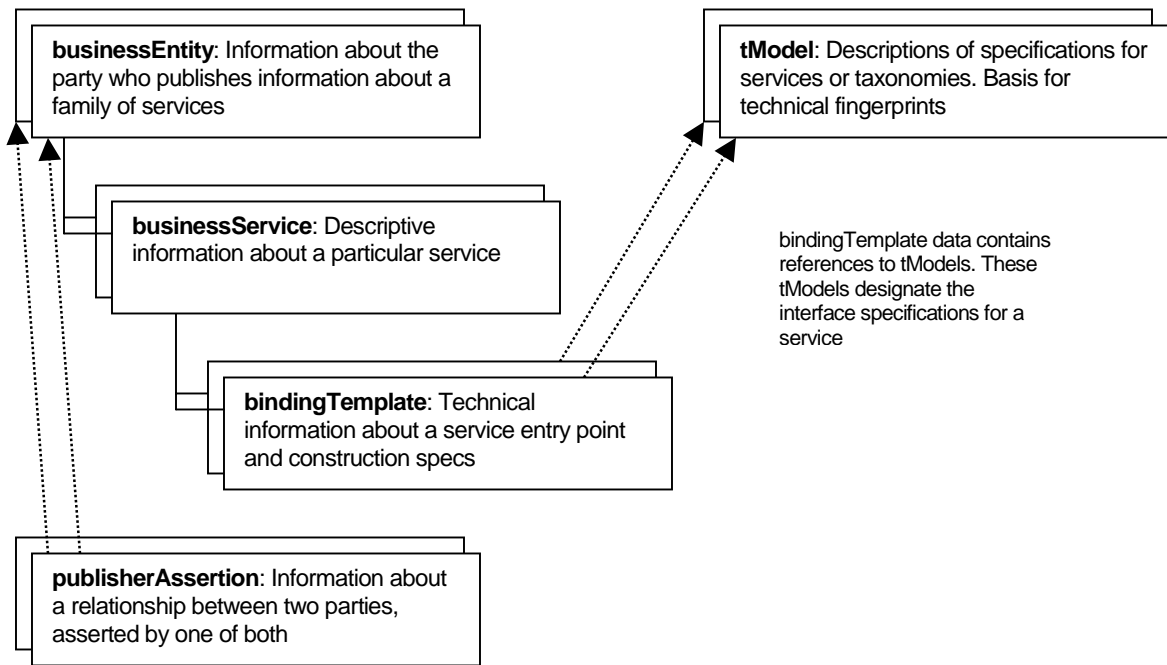


Figure 1

### 3 Overall Design Principles

Each of the five 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 even services that allow customers to query the status of a factory order. Each business, and the corresponding service descriptions (both logical and technical descriptions) all exist as separate instances of data within a UDDI registry.

#### 3.1 Unique identifiers

The individual facts about a business, its services, technical information, or even 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<sup>3</sup> (UUID). Technically, a UUID is a hexadecimal string 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<sup>4</sup>.

<sup>3</sup> The terms “Universally Unique Identifier” (UUID) and “Globally Unique Identifier” (GUID) are used synonymously in technical documentation. In the remainder of this document, the term UUID is used.

<sup>4</sup> The UUID structure and generation algorithm is described in the ISO/IEC 11578:1996 standard (see [www.iso.ch](http://www.iso.ch)).

## 3.2 Containment

The individual instance data managed by a UDDI registry are sensitive to the parent/child relationships found 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.

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. There are five core data types, and instances of each of these types are identified by unique keys. 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. The `publisherAssertion`'s key is logically the concatenation of all of its elements.

References on the other hand, occur in several places, especially for `tModels`. 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 five core data 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 five core structure types are references. Examples include `tModelKey` values found in lists within `bindingTemplate` and categorization and identification schemes – in which context the `tModel` represents a uniquely identifiable namespace reference and qualifier.

## 4 Data Structure Notation

Data structures are described by substructure breakdowns in tables of the following form.

Field Name	Description	Data Type	Length
Optional fields are written in normal font	Description of the field's meaning and whether it's <ul style="list-style-type: none"> <li>• An attribute or an element</li> <li>• Repeatable or not</li> </ul>	Possible Data Types include <ul style="list-style-type: none"> <li>• structure</li> <li>• string</li> <li>• UUID</li> </ul>	If the field's data type is string, the field's length is given here in Unicode characters
Required fields are written in <b>bold font</b>			

Most of the data structures are also given in their XML Schema representation (W3C XML Schema Candidate Recommendation, October 2000). Please use the UDDI XML Schema as the definitive technical reference, if needed.

## 5 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.

### 5.1 Structure specification

```
<element name = "businessEntity">
  <complexType>
    <sequence>
      <element ref = "discoveryURLs" minOccurs = "0"/>
      <element ref = "name" maxOccurs = "unbounded"/>
      <element ref = "description" minOccurs = "0"
        maxOccurs = "unbounded"/>
      <element ref = "contacts" minOccurs = "0"/>
      <element ref = "businessServices" minOccurs = "0"/>
      <element ref = "identifierBag" minOccurs = "0"/>
      <element ref = "categoryBag" minOccurs = "0"/>
    </sequence>
    <attribute ref = "businessKey" use = "required"/>
    <attribute ref = "operator"/>
    <attribute ref = "authorizedName">
  </complexType>
</element>
```

### 5.2 Substructure breakdown

Field Name	Description	Data Type	Length
<b>businessKey</b>	Attribute. This is the unique identifier for a given instance of a businessEntity structure.	UUID	41
authorizedName	Attribute. This is the recorded name of the individual that published the businessEntity data. This data is generated by the controlling operator and should not be supplied within save_business operations.	string	255
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 generated and should not be supplied within save_business operations.	string	255
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	
<b>name</b>	Required repeating element. These are the human readable names recorded for the businessEntity, adorned with a unique xml:lang value to signify the language that they are expressed in. Name search is provided via find_business call. Names may not be blank.	string	255



description	Optional repeating element. One or more short business descriptions. One description is allowed per national language code supplied.	string	255
contacts	Optional element. This is an optional list of contact information.	structure	
businessServices	Optional element. This is a list of one or more logical business service descriptions.	structure	
identifierBag	Optional element. This is an optional list of name-value pairs that can be used to record identifiers 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	

### 5.2.1 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 hoped that other conventions come about and use this structure to accommodate alternate means of discovery.<sup>5</sup>

Field Name	Description	Data Type	Length
discoveryURL	Attribute qualified repeating element holding strings that represent web addressable (via HTTP-GET) discovery documents.	string w/attributes	255

#### 5.2.1.1 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 Operator Site will generate one URL. The generated URL will point to an instance of either a businessEntity or businessEntityExt structure, and 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 discoveryURLs 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, since the XML returned will be formatted as a normal businessDetail message.

Field Name	Description	Data Type	Length
------------	-------------	-----------	--------

<sup>5</sup> 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".

<b>useType</b>	Required attribute that designates the name of the convention that the referenced document follows. Two reserved convention values are "businessEntity" and "businessEntityExt". URLs qualified with these values should point to XML documents of the same type as the useType value.	string	255
----------------	--	--------	-----

**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>
```

### 5.2.2 name

The publishing of several names, e.g. for romanization purposes, is supported. In order to signify the language that the names are expressed in, they carry unique xml:lang values. Not more than one name element may omit specifying its language. Names passed in this way will be assigned the default language code of the registering party. This default language code is established at the time that publishing credentials are established with an individual Operator Site. If no default language is provisioned at the time a publisher signs up, the operator can adopt an appropriate default language code.

The same mechanism applies to the name element within the businessService structure.

### 5.2.3 contacts

The contacts structure provides a way for information to be registered with a businessEntity record so that someone that finds the information can make human contact for any purpose. Since the information held within the UDDI Operator 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 Version 2.0 API 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.

#### 5.2.3.1 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.

For transliteration purposes (e.g. romanization) the suggested approach is to file multiple contacts.

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	255
description	Optional element. Zero or more language-qualified <sup>6</sup> descriptions of the reason the contact should be used.	string	255
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	255
phone	Optional repeating element. Used to hold telephone numbers for the contact. This element 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 w/attributes	50
email	Optional repeating element. Used to hold email addresses for the contact. This element 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 w/attributes	255
address	Optional repeating element. This structure represents the printable lines suitable for addressing an envelope.	structure	

### 5.2.3.2 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 three optional attributes. The useType describes the address' type in freeform text. 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. The tModelKey references a tModel that specifies the meaning of keyName keyValue pairs given in subordinate addressLine elements. For a description of how to use tModels in order to give the simple addressLine list structure and meaning, see Appendix E: Structured Address Example.

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	255

<sup>6</sup> 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.

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
tModelKey	Optional attribute. This is the unique key reference that implies that the keyName keyValue pairs given by subsequent addressLine elements are to be interpreted by the taxonomy associated with the tModel that is referenced.	string	255 41 <sup>7</sup>
addressLine	Optional repeating element containing the actual address in freeform text. If the address element contains a tModelKey, these addressLine elements are to be adorned each with an optional keyName keyValue attribute pair. Together with the tModelKey, keyName and keyValue qualify the addressLine in order to describe its meaning.	string w/attributes	80

### 5.2.3.3 addressLine

AddressLine elements contain string data with a line length limit of 80 character positions. Each addressLine element can be adorned with two optional descriptive attributes, keyName and keyValue. Both attributes must be present in each address line if a tModelKey is assigned to the address structure. By doing this, the otherwise arbitrary use of address lines becomes structured. Together with the address' tModelKey, keyName and keyValue virtually build a keyedReference that represents an address line qualifier, given by the referenced tModel. See Appendix E for an example how structured addresses can be represented. When no tModelKey is provided for the address structure, the keyName and keyValue attributes can be used without restrictions, for example, to provide descriptive information for each addressLine by using the keyName attribute. Since both the keyName and the keyValue attributes are optional, 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.

### 5.2.4 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.


### 5.2.5 identifierBag

The identifierBag element allows businessEntity or tModel structures to include information about common forms of identification such as D-U-N-S<sup>®</sup> numbers, tax identifiers, etc. This data can be used to signify the identity of the businessEntity, 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 Version 2.0 API Specification. For a full description of the structures involved in establishing an identity, see Appendix A: Using Identifiers.

<sup>7</sup> The data type for tModelKey allows for using URN values in a later revision. In the current release, the key is a generated UUID. Design work around managing duplicate urn claims will allow user supplied URN keys on tModels in the future.

### 5.2.6 categoryBag

The categoryBag element allows businessEntity, businessService and tModel structures to be categorized according to any of several available taxonomy based classification schemes. Operator Sites automatically provide validated categorization support for three taxonomies that cover industry codes (via NAICS), product and service classifications (via UNSPC) and geography (via ISO 3166). 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 Version 2.0 API Specification. For a full description of structures involved in establishing categorization information, see Appendix B: Using categorization.



## 6 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.

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

Any businessService projected in this way is not managed as a part of the referencing businessEntity, but centrally as a part of the referenced businessEntity. This means that changes of the businessService by the referenced businessEntity are automatically valid for the service projections done by referencing businessEntity structures.

In order to specify both referenced and referencing businessEntity structures correctly, service projections can only be published by a save\_business message with the referencing businessKey present in the businessEntity structure and both the referenced businessKey and the referenced businessService present in the businessService structure.

### 6.1 Structure Specification

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

### 6.2 Substructure Breakdown

Field Name	Description	Data Type	Length
businessKey	<p>This attribute is optional when the businessService data is contained within a fully expressed parent that already contains a businessKey value.</p> <p>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. The businessKey may differ from the publishing businessEntity’s businessKey to allow service projections.</p>	UUID	41

<b>serviceKey</b>	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.  When saving a new or updated service projection, pass the serviceKey of the referenced businessService structure.	UUID	41
<b>name</b>	Required repeating element. These are the human readable names recorded for the businessService, adorned with a unique xml:lang value to signify the language that they are expressed in. Name search is provided via find_service call. Names may not be blank.  When saving a new or updated service projection, pass the exact name of the referenced businessService, here.	string	255
description	Optional element. Zero or more language-qualified text descriptions of the logical service family.	string	255
<b>bindingTemplates</b>	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 geographic codes). These can be used during search via find_service. See categoryBag under businessEntity for a full description.	structure	

### 6.2.1 bindingTemplates

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



## 7 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.

### 7.1 Structure specification

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

### 7.2 Substructure breakdown

Field Name	Description	Data Type	Length
<b>bindingKey</b>	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	41
<b>serviceKey</b>	This attribute is optional when the bindingTemplate data is contained within a fully expressed parent that already contains a serviceKey value. If the bindingTemplate data is rendered into XML and has no containing parent that has within its data a serviceKey, the value of the serviceKey that is the ultimate containing parent of the bindingTemplate 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	41



description	Optional repeating element. Zero or more language-qualified text descriptions of the technical service entry point.	string	255
<b>accessPoint</b>	Required attribute qualified element <sup>8</sup> . This element is a text field that is used to convey the entry 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 <sup>9</sup> .	string w/attributes	255
<b>hostingRedirector</b>	Required element if <i>accessPoint</i> not provided. This element is adorned with a <i>bindingKey</i> attribute, giving the redirected reference to a different <i>bindingTemplate</i> . If you query a <i>bindingTemplate</i> and find a <i>hostingRedirector</i> value, you should retrieve that <i>bindingTemplate</i> and use it in place of the one containing the <i>hostingRedirector</i> data.	empty w/attributes	
<b>tModelInstanceDetails</b>	This structure is a list of zero or more <i>tModelInstanceInfo</i> elements. This data, taken in total, should form a distinct fingerprint that can be used to identify compatible services.	structure	

### 7.2.1 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, for example, `mailto:purch@fabrikam.com`.
- **http:** designates that the *accessPoint* string is formatted as an HTTP compatible Uniform Resource Locator (URL), for example, `http://www.fabrikam.com/purchasing`.
- **https:** designates that the *accessPoint* string is formatted as a secure HTTP compatible URL, for example `https://www.fabrikam.com/purchasing`.
- **ftp:** designates that the *accessPoint* string is formatted as a FTP directory address, for example `ftp://ftp.fabrikam.com/public`.
- **fax:** designates that the *accessPoint* string is formatted as a telephone number that will connect to a facsimile machine, for example `1 425 555 5555`.

<sup>8</sup> One of *accessPoint* or *hostingRedirector* is required.

<sup>9</sup> 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.

- **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, for example 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.

### 7.2.2 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 that is suitable within the same UDDI registry instance for querying and obtaining the bindingDetail data that is to be used.

More on the hostingRedirector can be found in the appendices for the UDDI Version 2.0 API Specification.

### 7.2.3 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 structure 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 structure), 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 tModelKey elements exposed in this manner.

#### 7.2.3.1 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
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	
-----------------	---	-----------	--

### 7.2.3.2 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-qualified 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 information or instructions related to proper use of a bindingTemplate technical sub-element.	structure	
instanceParms	Optional 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	255

### 7.2.3.3 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



## 8 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.

### 8.1 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 structures.

#### 8.1.1 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. Examples can be seen in the RosettaNet Partner Interface Processes<sup>10</sup> specification, the Open Applications Group Integration Specification<sup>11</sup> 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 identifier (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.

#### 8.1.2 Defining an abstract namespace reference

The other place where tModel references are used is within the identifierBag, categoryBag, address and publisherAssertion structures that are used to define organizational identity and various

---

<sup>10</sup> See [www.rosettanet.org](http://www.rosettanet.org)

<sup>11</sup> See [www.openapplications.org](http://www.openapplications.org)

classifications. Used in this context, the 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 US 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.

Once this 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.

To get things started, the UDDI Operator Sites have registered a number of useful tModels, including NAICS (an industry code taxonomy), UNSPC (a product and service category code taxonomy), and ISO 3166 (a geographical region code taxonomy).

## 8.2 Structure specification

```
<element name = "tModel">
  <complexType>
    <sequence>
      <element ref = "name"/>
      <element ref = "description" minOccurs = "0"
        maxOccurs = "unbounded"/>
      <element ref = "overviewDoc" minOccurs = "0"/>
      <element ref = "identifierBag" minOccurs = "0"/>
      <element ref = "categoryBag" minOccurs = "0"/>
    </sequence>
    <attribute ref = "tModelKey" use = "required"/>
    <attribute ref = "operator"/>
    <attribute ref = "authorizedName"/>
  </complexType>
</element>
```

## 8.3 Substructure breakdown

Field Name	Description	Data Type	Length
<b>tModelKey</b>	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. 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	255
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	255

<b>name</b>	Required element. This is the name recorded for the tModel. Name search is provided via find_tModel call. Names may not be blank, and should be meaningful to someone who looks at the tModel	string	255
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 information or instructions related to the tModel. See the substructure breakdown for overviewDoc in section The bindingTemplate structure.	structure	
identifierBag	Optional element. This is an optional list of name-value pairs that can be used to record identification numbers for a tModel. These can be used during search via find_tModel. See the full description of this element in the businessEntity section of this document and in Appendix A: 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 this element in the businessEntity section of this document and in Appendix B: Using categorization	structure	



## 9 The publisherAssertion structure

Many businesses, like large enterprises or marketplaces, are not effectively represented by a single businessEntity, since their description and discovery are likely to be diverse. As a consequence, several businessEntity structures can be published, representing individual subsidiaries of a large enterprise or individual participants of a marketplace. Nevertheless, they still represent a more or less coupled community and would like to make some of their relationships visible in their UDDI registrations. Therefore, two related businesses use the `xx_publisherAssertion` messages, publishing assertions of business relationships.

In order to eliminate the possibility that one publisher claims a relationship between both businesses that is in fact not reciprocally recognized, both publishers have to agree that the relationship is valid by publishing their own publisherAssertion. Therefore, both publishers have to publish exactly the same information. When this happens, the relationship becomes visible. More detailed information is given in the appendices for the UDDI Version 2.0 API Specification.

In the case that a publisher is responsible for both businesses, the relationship automatically becomes visible after publishing just one of both assertions that make up the relationship.

The publisherAssertion structure consists of the three elements `fromKey` (the first businessKey), `toKey` (the second businessKey) and `keyedReference`. The `keyedReference` designates the asserted relationship type in terms of a `keyName` `keyValue` pair within a `tModel`, uniquely referenced by a `tModelKey`.

### 9.1 Structure Specification

```
<element name = "publisherAssertion">
  <complexType>
    <sequence>
      <element ref = "fromKey" />
      <element ref = "toKey" />
      <element ref = "keyedReference" />
    </sequence>
  </complexType>
</element>
```

### 9.2 Substructure Breakdown

Field Name	Description	Data Type	Length
<b>fromKey</b>	Required element. This is the unique key reference to the first businessEntity the assertion is made for.	UUID	41
<b>toKey</b>	Required element. This is the unique key reference to the second businessEntity the assertion is made for.	UUID	41
<b>keyedReference</b>	Required element. This designates the relationship type the assertion is made for, represented by the included <code>tModelKey</code> and described by the included <code>keyName</code> <code>keyValue</code> pair.	empty w/attributes	



## 10 Appendix A: Using Identifiers

### 10.1 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 D-U-N-S<sup>®</sup> numbers<sup>12</sup>, Global Location Numbers (GLN)<sup>13</sup>, 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<sup>®</sup> 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 this string, 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.

### 10.2 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<sup>®</sup> numbers. When you know an organization's D-U-N-S<sup>®</sup> 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<sup>®</sup> numbers, versus a tax identifier or an organizational membership number.)
- **Extensible:** The way that searchable identifiers are defined 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.

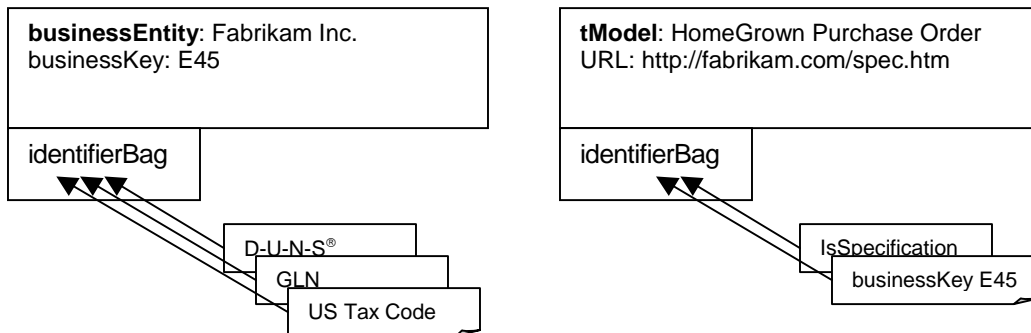
---

<sup>12</sup> D-U-N-S<sup>®</sup> Numbers are provided by Dun & Bradstreet. See <http://www.dnb.com>.

<sup>13</sup> The Global Location Number system is defined in the EAN UCC system (<http://www.ean-int.org/locations.html>).

### 10.2.1 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`<sup>14</sup>. 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, a Global Location Number, and a US Tax Code identifier<sup>15</sup>.

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 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.

Two identifier types have been identified and made a core part of the UDDI Operator registries, so far. These are the Dun & Bradstreet D-U-N-S® numbers and the Thomas Register supplier IDs<sup>16</sup>.

Identifier Name	tModel name
D-U-N-S	dnb-com:D-U-N-S
Thomas Register	thomasregister-com:supplierID

<sup>14</sup> 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.

<sup>15</sup> In the diagram, the actual name/value properties were abbreviated for the sake of simplicity.

<sup>16</sup> See <http://www.thomasregister.com>.

## 10.2.2 Structure Specification

```
<element name = "identifierBag">
  <complexType>
    <sequence>
      <element ref = "keyedReference" minOccurs = "0"
        maxOccurs = "unbounded" />
    </sequence>
  </complexType>
</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">
  <complexType>
    <attribute ref = "tModelKey" />
    <attribute ref = "keyName" />
    <attribute ref = "keyValue" use = "required" />
  </complexType>
</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 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 2 functions. Assuming that the identifiers were fully defined, the five types shown would each reference one of five different 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 publisherAssertion structure and 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.

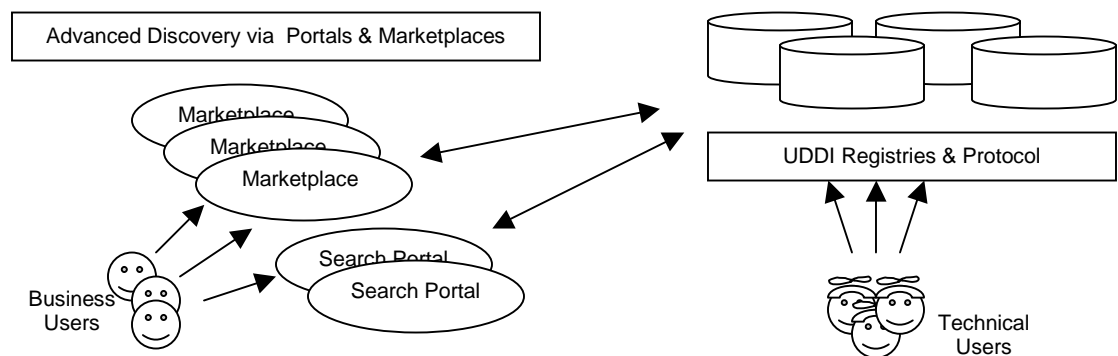
## 11 Appendix B: Using categorization

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, it is unlikely that searching for businesses that satisfy a particular set of criteria will yield a manageably sized result set. For example, suppose we searched for all 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 a priori 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 3**

In figure 3 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.

### 11.1 Structure Specification

```
<element name = "categoryBag">
```

```

    <complexType>
      <sequence>
        <element ref = "keyedReference" minOccurs = "0"
          maxOccurs = "unbounded" />
      </sequence>
    </complexType>
  </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: Using Identifiers) and the basic structure is used in the same way.

Three categorization taxonomies have been identified and made a core part of the UDDI Operator registries, so far. These are the North American Industry Classification System (NAICS)<sup>17</sup>, Universal Standard Products and Services Classification (UNSPSC)<sup>18</sup>, and ISO 3166<sup>19</sup>, the international standard for geographical regions, including codes for countries and first-level administrative subdivisions of countries. A fourth category is also defined – named “Other Taxonomy” – for general-purpose keyword type classification<sup>20</sup>.

The `tModel` names for these taxonomies are

Taxonomy Name	tModel name
NAICS	ntis-gov:naics:1997
UNSPSC	unspsc-org:unspsc:3-1
ISO 3166	iso-ch:3166:1999
Other Taxonomy	uddi- org:general_keywords

<sup>17</sup> See <http://www.census.gov/epcd/www/naics.html>.

<sup>18</sup> See <http://www.unspsc.org>.

<sup>19</sup> See <http://www.din.de/gremien/nas/nabd/iso3166ma>.

<sup>20</sup> Operator Sites are allowed to promote invalid category entries, or entries that are otherwise rejected by the category classification services, to this miscellaneous taxonomy.

## 12 Appendix C: Response message reference

All of the messages defined in the UDDI Version 2.0 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 to the specifications described in the appendices for the UDDI Version 2.0 API Specification. Only the SOAP <body> element contents are shown in the examples in this section.

### 12.1 assertionStatusReport

This message returns zero or more assertionStatusItem structures in response to a get\_assertionStatusReport inquiry message.

#### 12.1.1 Sample

```
<assertionStatusReport generic="2.0" operator="uddi.someoperator" xmlns="urn:uddi-org:api_v2">
  <assertionStatusItem completionStatus="status:toKey_incomplete">
    <fromKey>F5E65...</fromKey>
    <toKey>A237B...</toKey>
    <keyedReference tModelKey="uuid:F5E65..." keyName="Subsidiary" keyValue="1"
    </keyedReference>
    <keysOwned>
      <fromKey>F5E65</fromKey>
    </keysOwned>
  </assertionStatusItem>
  [<assertionStatusItem/>...]
</assertionStatusReport>
```

This message reports all complete and incomplete assertions and serves an administrative use including the determination if there are any outstanding, incomplete assertions about relationships involving businesses the publisher account is associated with.

Since the publisher who was authenticated by the get\_assertionStatusReport message can manage several businesses, the assertionStatusReport message shows the assertions made for all businesses managed by the publisher.

While the elements fromKey, toKey and keyedReference together identify the assertion whose status is being reported on, the keysOwned element designates those businessKeys the publisher manages.

An assertion is complete only if the completionStatus attribute says so, that is, having a value "status:complete". If completionStatus has a value "status:toKey\_incomplete" or "status:fromKey\_incomplete", the party who controls the businessEntity referenced by the toKey or the fromKey has not made a matching assertion, yet. In the example we can see that the party who controls the businessEntity with the businessKey A237B... has not made a matching assertion to the one found in the assertionStatusItem, made by the party who controls the businessEntity with the businessKey F5E65... .

### 12.2 authToken

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

#### 12.2.1 Sample

```
<authToken generic="2.0" operator="uddi.someoperator" xmlns="urn:uddi-org:api_v2" >
  <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 Publisher's API messages that change data. This message is always returned using SSL encryption as a synchronous response to the `get_authToken` message.

## 12.3 bindingDetail

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

### 12.3.1 Sample

```
<bindingDetail generic="2.0" operator="uddi.someoperator" truncated="true"
  xmlns="urn:uddi-org:api_v2">
  <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.

## 12.4 businessDetail

This message returns one or more complete `businessEntity` structures in response to a `get_businessDetail` inquiry message.

### 12.4.1 Sample

```
<businessDetail generic="2.0" operator="uddi.sourceOperator" truncated="true"
  xmlns="urn:uddi-org:api_v2">
  <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 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.

## 12.5 businessDetailExt

This message returns one or more complete `businessEntityExt` structures 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.

### 12.5.1 Sample

```
<businessDetailExt generic="2.0" operator="uddi.sourceOperator" truncated="true"
  xmlns="urn:uddi-org:api_v2">
  <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 Version 2.0 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 Sites are required to support the *Ext* form of the businessDetail message for compatibility with tools, but are not allowed to manage extended data.

## 12.6 businessList

This message returns zero or more businessInfo structures 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.

### 12.6.1 Sample

```

<businessList generic="2.0" operator="uddi.sourceOperator" truncated="true"
  xmlns="urn:uddi-org:api_v2">
  <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 zero 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<sup>21</sup>. Notice that the businessKey attribute is not expressed in the serviceInfo structure due to the fact that this information is available from the containing businessInfo structure.

## 12.7 publisherAssertions

This message returns one or more publisherAssertion structures in response to a set\_publisherAssertions or a get\_publisherAssertions publishing message.

### 12.7.1 Sample

```

<publisherAssertions generic="2.0" operator="uddi.someoperator" authorizedName="J. Doe"
  xmlns="urn:uddi-org:api_v2">
  <publisherAssertion>
    <fromKey>F5E65...</fromKey>
    <toKey>A237B...</toKey>
    <keyedReference tModelKey="uuid:34D5..." keyName="Holding Company"
      keyValue="parent-child"
    </keyedReference>
  </publisherAssertion>
  [<publisherAssertion/>...]
</publisherAssertions>

```

<sup>21</sup> Refer to the UDDI XML schema for structure details.



This message returns all assertions made by the publisher who was authenticated in the preceding `set_publisherAssertions` or the `get_publisherAssertions` message.

## 12.8 registeredInfo

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

### 12.8.1 Sample

```
<registeredInfo generic="2.0" operator="uddi.sourceOperator" [truncated="false"]
  xmlns="urn:uddi-org:api_v2">
  <businessInfos>
    <businessInfo businessKey="F5E65..." >
      <name>My Company</name>
      <serviceInfos>
        <serviceInfo serviceKey="3D45...">
          <name>Purchase Orders</name>
        </serviceInfo>
      </serviceInfos>
    </businessInfo>
    [<businessInfo/>...]
  <businessInfos>
  <tModelInfos>
    <tModelInfo tModelKey="uuid:34D5...">
      <name>Proprietary XML purchase order</name>
    </tModelInfo>
    [<tModelInfo/>...]
  </tModelInfos>
</registeredInfo>
```

This message contains overview data about business and tModel 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.

## 12.9 relatedBusinessesList

This message returns zero or more relatedBusinessInfo structures in response to a `find_relatedBusinesses` inquiry message.

### 12.9.1 Sample

```
<relatedBusinessesList generic="2.0" operator="uddi.someoperator" [truncated="false"]
  xmlns="urn:uddi-org:api_v2">
  <businessKey>F5E65...</businessKey>
  <relatedBusinessInfos>
    <relatedBusinessInfo>
      <businessKey>A237B</businessKey>
      <name>Matt's Garage</name>
      <description>Car services in ...</description>
      <sharedRelationships>
        <keyedReference tModelKey="uuid:F5E65..."
          keyName="Subsidiary"
          keyValue="1"
        </keyedReference>
        [<keyedReference/>...]
      </sharedRelationships>
    </relatedBusinessInfo>
    [<relatedBusinessInfo/>...]
  </relatedBusinessInfos>
```

```
</relatedBusinessesList>
```

For the businessEntity *specified* in the find\_relatedBusinesses, this structure reports complete business relationships with other businessEntity registrations. Business relationships are complete between two businessEntity registrations when the publishers controlling each of the businessEntity structures involved in the relationship set assertions affirming that relationship.

Each relatedBusinessInfo structure contains information about a businessEntity that *relates* to the specified businessEntity by at least one relationship. This information about the related businessEntity comprises its businessKey, name and optional description data, along with a collection element named sharedRelationships that in turn can contain zero or more keyedReference elements. These keyedReference elements, together with the businessKey elements for specified and the related businessEntity represent the complete relationships, that is, matching publisher assertions made by the publishers for each businessEntity.

## 12.10 serviceDetail

This message returns one or more complete businessService structures in response to a get\_serviceDetail inquiry message.

### 12.10.1 Sample

```
<serviceDetail generic="2.0" operator="uddi.sourceOperator" [truncated="false"]
  xmlns="urn:uddi-org:api_v2">
  <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.

## 12.11 serviceList

This message returns zero or more serviceInfo structures in response to a find\_service inquiry message.

### 12.11.1 Sample

```
<serviceList generic="2.0" operator="uddi.sourceOperator" [truncated="false"]
  xmlns="urn:uddi-org:api_v2">
  <serviceInfos>
    <serviceInfo serviceKey="3D45..." businessKey="2E4C...">
      <name>Purchase Orders</name>
    </serviceInfo>
  </serviceInfos>
</serviceList>
```

ServiceInfo structures are abbreviated versions of businessService data, 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.

## 12.12 tModelDetail

This message returns one or more complete tModel structures in response to a get\_tModelDetail

inquiry message.

### 12.12.1 Sample

```
<tModelDetail generic="2.0" operator="uddi.sourceOperator" [truncated="false"]
  xmlns="urn:uddi-org:api_v2">
  <tModel tModelKey="uuid:F5E65..." authorizedName="J. Doe"
operator="uddi.publishingOperator" >
  ...
  </tModel>
  [<tModel/>...]
</tModelDetail>
```

Because tModel structures are top-level data (that is, 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 operator attributes each express the distinguished name of the Operator Site that is providing the data and the operator where the data is managed.

### 12.13 tModelList

This message returns zero or more tModelInfo structures in response to a find\_tModel inquiry message.

#### 12.13.1 Sample

```
<tModelList generic="2.0" operator="uddi.sourceOperator" [truncated="false"]
  xmlns="urn:uddi-org:api_v2">
  <tModelInfos>
    <tModelInfo tModelKey="uuid:34D5...">
      <name>Proprietary XML purchase order</name>
    </tModelInfo>
    [<tModelInfo/>...]
  </tModelInfos>
</tModelList>
```

The tModelInfo structures are abbreviated versions of tModel data, suitable for finding candidate tModels, populating lists of results and then providing drill-down features that rely on the get\_xxDetail messages.

## 13 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, given in Unicode characters. The Operator Sites will truncate data that exceeds these lengths. Fields that are generated by the Operator 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.

<b>Field Name</b>	<b>Length</b>
<i>accessPoint</i>	255
<i>addressLine</i>	80
<i>authInfo</i>	4096
<i>authorizedName</i>	255
<i>bindingKey</i>	41
<i>businessKey</i>	41
<i>description</i>	255
<i>discoveryURL</i>	255
<i>email</i>	255
<i>fromKey</i>	41
<i>hostingRedirector</i>	41
<i>instanceParms</i>	255
<i>keyName</i>	255
<i>keyType</i>	16
<i>keyValue</i>	255
<i>name</i>	255
<i>overviewURL</i>	255
<i>personName</i>	255
<i>phone</i>	50
<i>serviceKey</i>	41
<i>sortCode</i>	10
<i>tModelKey</i>	255
<i>toKey</i>	41
<i>uploadRegister</i>	255
<i>URLType</i>	16
<i>useType</i>	255

## 14 Appendix E: Structured Address Example

The address structure, contained in the businessEntity structure, contains a simple list of addressLine elements. While this is useful for publishing addresses in a UDDI registry or simply printing them on paper, the address' structure and meaning remains hidden for a given businessEntity. For this reason, address structures can be adorned virtually with keyedReference elements. In fact a tModelKey attribute can be provided for an address structure and keyName key/Value attribute pairs can be provided for each addressLine element. This example is provided to demonstrate how the application of tModelKey, keyName and key/Value attributes to address structures can be used to give structure and meaning to a given address.

Let us assume that a community of several country-specific postal agencies, called "IBCPA", not existing in reality, agreed on a core set of address components for exchanging data electronically. This set currently comprises the components Street, Street number, Postal code, City, District, Region and Country.

In order to make these address components available for their use in UDDI address structures, IBCPA assigns a unique value (10, 20, ..., 70) to each address component and publishes a tModel with a save\_tModel message call that contains a tModel structure in the following form.

```
<tModel>
  <name>IBCPA.org:address:1.0</name>
  <description xml:lang="en">Codes for Address Components defined by the International
    Board of Postal Agencies</description>
  <overviewDoc>http://www.ibcpa.org/address/codes.html</overviewDoc>
  ...
</tModel>
```

IBCPA gets back the tModelKey A548.... As a result, the IBCPA set of address components can now be used by every publisher to structure their addresses in businessEntity structures. The following example shows an address structure using the IBCPA tModel in a save\_business message call.

```
<address useType="Sales office" tModelKey="uuid:A548...">
  <addressLine keyName="Street" keyValue="10">Alexanderplatz</addressLine>
  <addressLine keyName="Street number" keyValue="20">12</addressLine>
  ...
  <addressLine keyName="Country" keyValue="70">Deutschland</addressLine>
</address>
```