GUIDE business transaction markup

Early Draft, Version 0.13, 12 September 2000

Open submission to ebXML and for use with ebXML.

Abstract

GUIDE is a XML format for describing business information interchanges between a set of endpoints exchanging transactions. GUIDE is a layered approach, so that each aspect of the GUIDE syntax is expressed as a separate markup layer. Separation into layers is a fundamental requirement in order to meet the ability to deploy the semantic web as opposed to the content-based web of today.

The objective of GUIDE is to provide a simple open business interchange system for the consistent exchange of transactions. Therefore a business when implementing a set of business interchanges with a trading partner will seek to find or create the appropriate *guides* with which to define the required business semantics of such interchanges (semantic guides).

Layers provide a high degree of flexibility in how GUIDE can be approached and utilized. Layers allow GUIDE to function as a repository classification system, a transactional payload format, or as a harmonization bridge between simple XML V1.0 DTD syntax, more complex schema dialects and proprietary or specialized markup. Layers also allow GUIDE to provide globally maintainable interchange mechanisms and provide for impact management when adopting future XML syntax enhancements within each layer.

The goal of providing simple business use scenarios then clearly defines the principles and constraints for the GUIDE system itself. The semantic guide consists of three layers (categories) of information: firstly a description of the structure and model of the physical interchange markup along with the associated process actions and verbs; secondly the datatyping and business context markup; and thirdly a top level classification mechanism that allows for the grouping together of industry vertical sets of semantic guides to facilitate location and re-use of particular business functional components.

The key to cost-effective business information interchanges is to provide a guaranteed consistent behaviour between different computer systems in a loosely coupled distributed network environment. To achieve this GUIDE purposefully limits and designs the markup syntax to only a business functional subset that can be known to be consistently programmed and implemented.

Within these constraints GUIDE is extensible to allow description of both legacy and future business record layouts and information structures. This allows GUIDE to provide a natural migration path from existing business application systems to the future with XML based ones in a simple and consistent way without requiring extensive software re-engineering. This approach also allows users themselves to select simple lightweight low cost solutions, alongside more sophisticated and extended applications. This in turn positions the GUIDE approach to gain broad adoption across the marketplace because barriers to adoption will be minimized.

Status

This draft represents the blending of current practical work in a variety of areas with XML, including the latest W3C Schema and Datatyping drafts, MSL typing markup, SOAP based interchanges, ISO11179, tpaML and ebXML related work. It is not the intention that GUIDE replace all these other initiatives, but rather that GUIDE provide a consistent way to harmonize these more complex syntaxes into a format that ordinary businesses can use reliably and consistently for basic day-to-day information interchanges. This will also allow developers to create base implementations of XML parsers and tools that are simply GUIDE compatible, that can later be extended to also support more complex syntaxes as business needs dictate. This draft is published to the ebXML and W3C for the purpose of creating a working document around which continued work can proceed. It is anticipated that early implementations of GUIDE will mature and improve with additional contributions and syntax enhancements and in no way should this current draft be seen as a completed specification prior to final release of a formal 1.0 version.

Contributors

Document Editor: TBD

Contributors:

David RR Webber.

1. Table of Contents

2. Introduction

The objective of GUIDE is to provide a simple business interchange system for data that is compatible with XML V1.0, the W3C schema, OASIS Registry and other related schema work. The acronym GUIDE stands for:

Global Uniform Interoperable Data Exchange

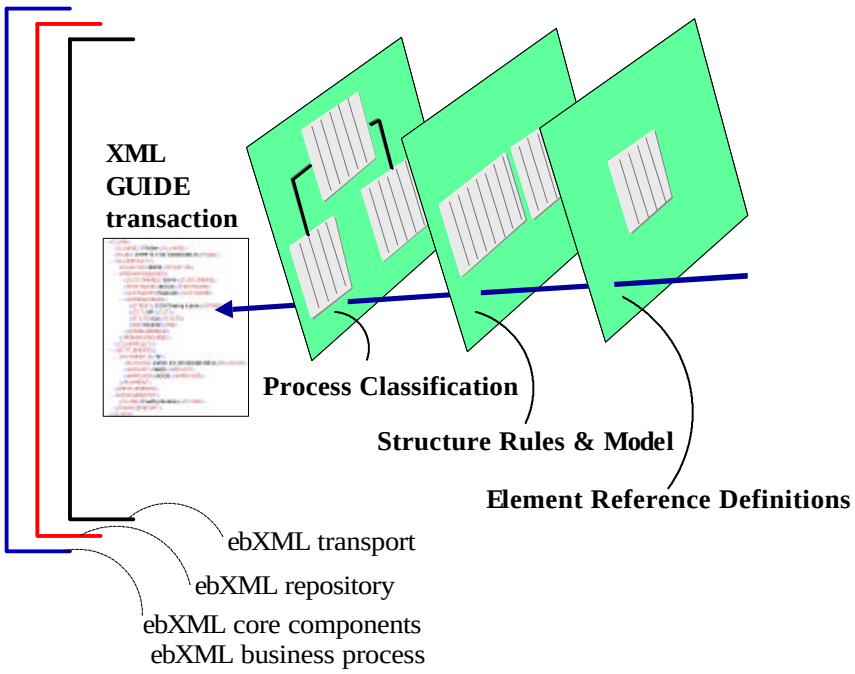
The GUIDE approach at a glance consists of three layers that allow a simple and clear view of the key aspects of information exchange.

The top level is the *classifications*. This mechanism allows you to group together industry vertical sets of transactions so you can quickly and easily find the particular business functional components that you require based on business use and context.

The core layer is then the Guide schema *structures* that carry the actual information exchanges and define how you build physical transaction instances.

On the bottom layer are the Guide *elements*; the data dictionary that specify each piece of information contained within the Guide structures. Figure 1 shows how the layers relate, and how they relate to the mechanisms described by the ebXML architecture technical specifications.

Figure 1. Guide Layers.



2.1 Design Goals

The GUIDE principles require that the syntax must be:

1) Simple to understand, learn, read and use.

2) Provide a concise feature function set thereby ensuring consistent implementations, interoperability, and low cost of adoption. Each feature must earn its place based on widespread business need and applicability.

3) Separate the datatyping and definition layer from the physical modelling layer. This ensures easy to build transaction structure syntax while providing maintainable reusable business element definitions for horizontal and vertical industry dictionaries.

4) Support traditional EDI style hierarchical structured information formats and exchanges with version control and interchange control.

5) Provide basic object oriented semantics for methods, classes, and simple inheritance to allow business exchanges of industry wide process components. [Extended complex features that require excessive levels of software complexity to engineer and lead to uncertain deployment behaviours will be specifically excluded. Examples include polymorphism, multiple inheritance, nested imports, pattern facets, and similar exotic programming features and behaviours].

6) Provide link to direct browser form rendering from GUIDE definitions to allow user presentation with multilingual support.

7) Provide a simple metaphor to migrate and express COBOL copybooks, SQL table definitions, CICS structures, program data structures, business data dictionaries and similar information content quickly and easily into.

8) Be based on the W3C XML markup syntax, with minimal use of extended features, and be consistent with and interoperable with the ebXML technical specifications.

9) Above all, provide both large industry partners and small businesses with mission critical high volume, high performance, and open public standard based interchanges. Coupled with the long term means to conduct and maintain cost effective electronic information exchanges that can be simply deployed and exploited by as large a cross-section of the workforce as possible.

2.2 Examples of GUIDE layers

The GUIDE layers are designed to separate each aspect of the markup, thereby making each layer itself simple elegant and intuitive to learn. This approach also provides a built-in reuse of commonly occurring definitions. Furthermore, within GUIDE syntax extensive use is made of common default values so that the syntax is uncluttered and particularly easy to create and manually interpret. In the past schema were expected to have complex syntax set heavily dependent on explicit semantics to define all of the aspects of structure, datatyping and semantics. The GUIDE approach also provides a mechanism for use with XML V1.0 DTD syntax; see section D of the Addendum for using this method. This provides a simple means to use GUIDE with current parser based tools.

Example 1 GUIDE structure for a mailing address

This example illustrates a simple physical structure model with a repeated group of information. Other more complex structures are shown as later examples.

```
<mailAddress
   xmlns:guide="http://www.ebXML.org/guides/address.xml">
   <fullName>Joe H. Smith</fullName>
   <street>101 Main Street</street>
   <street>Apartment 20b</street>
   <city>Taggtown</city>
   <ZIP>10230-0001</ZIP>
   <state>AZ</state>
   <accountActive>YES</accountActive>
 </mailAddress>
```

The intent of the example here is to introduce GUIDE syntax in a familiar data construct. The associated GUIDE element and GUIDE classification layers are then shown in Example 2 and Example 3. The GUIDE structure that is referenced in Example 1 is shown here.

```
<?xml version="1.0" ?>
 <xmlGuide use="structure" name="mailingAddress" version="0.1"
    xmlns:qic="http://www.ebXML.org/guides/elements/postal.xml"
    xmlns:crm="http://www.crm.org/guides/elements/basics.xml">
  <sequence>
    <element name="fullName"      qic:base="personDetails" />
    <element name="street"        qic:base="postalStreet"
                      OCCURS="+" LIMIT="5" />
    <element name="city"          qic:base="postalCity" qic:mask="UX19" />
    <element name="ZIP"           qic:base="usPostalCode" />
    <element name="state"         qic:base="usStateCode" />
    <element name="accountActive" qic:base="crm:activeStatus" />
  </sequence>
</xmlGuide>
```

There are several important business aspects being demonstrated here. The 'qic:base' definitions are using Qualifying Indicator Codes (see Example 2 below) to provide the link between the structural layer and the element layer. This approach provides an abstract linkage between a physical element and the actual definition. This is a familiar technique from EDI where industry standard code and element definitions provide commonality across widely differing local usage terminology. An extended discussion of the qic:base mechanism and syntax is provided in section 2.3 with further examples. There are three modes that qic:base references can be used in to establish the linkage between the structural layer and the element layer definitions, *simple*, *annotated* and *local*. Examples of using simple and annotated modes are shown in example 1. The default namespace qic points to the location of the qic:base references, and an inline reference to a namespace can override this for local definitions, as with the crm: namespace use. (W3C note: simple mode can provide a harmonization with the MSL typing markup proposal and other proposals such as RELAX.)

Next, the example shows the use of structural modelling syntax. Exception based markup is a key design aspect of GUIDE. Only necessary markup is required, and common default behaviours are used to remove unnecessary verbose markup. Following this approach all element definitions are assumed to denote a unit item, unless compound or repeat indication markup is used to denote otherwise. In this respect the OCCURS / LIMIT construct is shown as the preferred occurrence indicator mechanism as it provides more concise readable syntax. The OCCURS and LIMIT attributes are both optional and therefore when used provide an immediate business rule indicator that an interchange is constrained in some important way for a business partner. This significantly improves readability and the ability to use software agent technology to scan structures for possible process clashes.

To further enhance future interoperability the OCCURS attribute only allows the loop constructs '*' and '+' that are of course equivalent to 'do while' and 'do until' with no explicit constraint other than 'end of data'. The OCCURS may not use any explicit value (the equivalent of a FOR loop), that functionality is reserved to the LIMIT attribute, and thus the use of a value that leads to potential business information processing structure clashes can be immediately detected. (Note: The OCCURS may also have a value of 'optional' for compatibility with the DTD '?' syntax usage).

The element type references in a GUIDE structure have three mechanisms as detailed in section 2.3 below. The two referential mechanisms are illustrated in Example 1 above. Referring to Example 1, first *simple* mode is illustrated by the definition of all the elements. Then *annotated* mode is illustrated by the definition of city that references the that qic:base postalCity, but then provides an annotation using a *mask*[1] by locally overriding the format to a twenty character alphanumeric string, the first character of which will always be returned capitalized (\UX19). Notice there are two aspects to this; an annotation is merely designed to allow locally annotated variations to the qic:base definitions, however where the business use or sense is different, then a new item qic:base should be created and cross-referenced to the parent (see ISO11179 data element registry specifications for full descriptions and use of these mechanisms).

Note 1: Please refer to the appendix section in this document on mask definitions for the full mask syntax use. As a brief note here the control characters used are '#' = numeric digit, 'X' = alphanumeric, 'U' = uppercase alphanumeric, and the number suffix is the repeat count (element field length).

W3C compatibility note: GUIDE prefers mask definitions to the more functionally complex W3C pattern facets. There are several business reasons for this. Firstly, mask definitions map more cleanly from legacy application language record structures that already use masks and masks are intuitive to read and learn; secondly masks provide sufficient expression for most all business application needs; and thirdly pattern facets add significantly to the complexity of the parser implementation. Thus adoption of masks obeys the design criteria for GUIDE more closely than pattern facets.

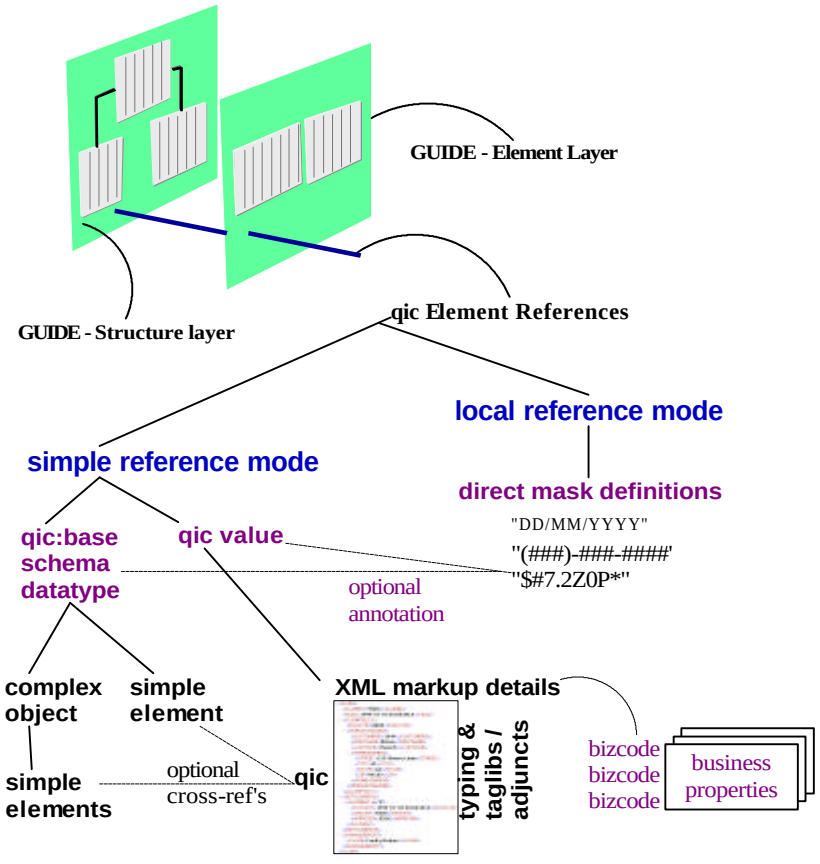
The adoption of mask over pattern facet demonstrates another capability of the GUIDE syntax, the ability to provide a simple human intuitive construct that can be mapped into a machine cryptic format for backend or cross-schema dialect compatibility (see appendix for examples of GUIDE masks and the equivalent using W3C cryptic pattern facets). A further example is that mask definitions can be mapped to POSIX and similar mask definitions as required.

2.3 Qualifier Indicator Codes (QIC) system.

The QIC system allows use of three modes of reference indicator within the XML syntax. Each mode is now examined in turn, and its usage. The QIC reference itself is designed for use with an XML parser using the namespace and IDREF mechanisms to resolve external references to extended XML definitions of elements within a GUIDE structure definition. Using this approach allows a GUIDE compliant parser to retrieve only those QIC references that are explicitly needed by the application software interfacing with the XML parser, and further to cache such references to reduce network traffic during extended GUIDE based business interchanges. For more details on parser behaviour see the sections 4.4 and 5.4 on linking and behaviour and optional use of XPath optimization.

The QIC system uses two related references, the *qic* reference, and the *qic:base* reference. The optional qic:base reference is provided for compatibility with schema based datatyping systems where the named reference is a base typing from within a schema layer that is pointed to by the datatypes namespace within the GUIDE element layer (see example 2). The figure 2 shows how these reference modes relate and can be used together or separately to define the business elements needed.

Figure 2. GUIDE QIC reference modes



The qic reference mode uses the '?value' pair to provide an XML IDREF lookup to the content referred to by the default element namespace for the structure (additional namespaces may be declared and then the reference indicator uses the form '?ns:value' to qualify the reference explicitly). It should be noted that the qic reference system is analogous to the concept of a Global Unique ID (GUID) system. However, qic assumes uniqueness only within context of the given registry, and therefore does not require a global registration service. This refinement is compatible with the barcodes model already used for product registration, where barcodes can be locally unique or globally unique (registered).

When either a qic or a qic:base reference indicator has a optional mask attribute, this provides an alternate picture mask. This can be used to override the length and format definition from the element reference layer, as this is the most common change required when using standard data dictionaries and therefore provides a quick and intuitive override mechanism.

With the qic="" reference, when the type definition is not preceded with a '?' reference indicator this is then the *local* reference mode use and therefore it is an in-line *character mask* definition.

Local mode is designed to be used for explicit local internal interchange usage only, or to provide a rapid means for migrating legacy data structures. All other uses should prefer the reference simple or annotated indicator mode.

Example 2 now follows on to demonstrate each of these modes and the extended capabilities that the layer approach brings to GUIDE by showing the element definition structure itself.

Example 2 GUIDE elements for a mailing address

This example shows the definition of GUIDE elements that are referenced in the Example 1 structure. The type reference indicators in example 1 and the default element namespace are used to reference the GUIDE element layer. The element layer reference can be an extended ebXML repository query reference (see ebXML repository technical specifications), or it maybe a default GUIDE reference using a simple XML IDREF lookup to retrieve the basic datatyping information from the element reference layer.

As can be seen from the sparse syntax required in Example 1 above this brings performance, readability and interoperability advantages. The GUIDE element layer is extensible without effecting already deployed GUIDE structures that reference it. Also considerable syntax overhead is saved from the GUIDE structure model itself. Each layer can focus on providing the explicit functionality required by that layer.

Additionally the element reference layer is designed to support an extensible list of information reference types, some examples have been included: XForm, SQL, EDI (igML) and MSL (schema).

The default retrieval will be ebXML repository base element definitions (example items have been used here to illustrate the concepts since the ebXML technical specifications are still a work in progress).

```
<?xml version="1.0" ?>
<!--
* GUIDE element for use with namespace and IDREF *
* reference system.                               *
*                                                 *
* Version 0.11 August, 2000                       *
*                                                 *
* Guide Extensions (taglib):                      *
*   XForm                                         *
*   SQL                                           *
*   EDI                                           *
*   MSL                                           *
 -->
<xmlGuide use="element" name="xmlg:mailingAddress" version="0.1"
  xmlns:datatypes="http://www.ebXML.org/guides/datatypes.xml"
  xmlns:qic="http://www.ebXML.org/guides/bizcodes.xml">
  <definitions>
   <bizcode qic="ADR01001" qic:base="personDetails" bizname=" fullName">
    <guide>
     <status date="21/02/2000">candidate</status>
     <maxlength>30</maxlength>
     <minlength>1</minlength>
     <datatype>string</datatype>
      <mask>X30</mask>
      <values default="">
        <value /> <!-- allowed values can go here when applicable -->
      </values>
      <localdescription xml:lang="EN" xml:space="preserve">The full name of a
       person as a single unformatted human readable string.
      </localdescription>
      <fulldescription xml:lang="EN" mimetype="HTML" >
          http://www.address.org/desc/ADR01001.htm</fulldescription>
     <labels>
       <label xml:lang="EN">Full Name</label>
       <label xml:lang="GR">Groß nam</label>
       <label xml:lang="FR">Nom complet</label>
       <label xml:lang="IT">Nome completo</label>
       <label xml:lang="ES">Nombre completo</label>
    </labels>
    <seeAlso>
     <similar>ADR04402</similar>
     <equivalent>X1205730</equivalent>
     <equivalent>HL706641</equivalent>
```

```
      <related>ADR04403</related>
     </seeAlso>
     <dependencies>
      <dependent type="required">ADR01002</dependent>
      <dependent type="required">ADR01003</dependent>
      <dependent type="required">ADR01004</dependent>
      <dependent type="required">ADR01005</dependent>
      <dependent type="optional">ADR01006</dependent>
    </dependencies>
    <attributes>
     <attribute name="xml:lang" qic:base="xml_lang_code" type="optional" />
     <attribute name="customerID" qic="ADR02105" type="optional" />
    </attributes>
   </guide>
   <extensions>
    <extension type="ADR01001:XForm">
     <item type="formcontrol">textfield</item>
    </extension>
    <extension type="ADR01001:SQL">
     <item type="datatype">varchar</item>
     <item type="length">30</item>
    </extension>
   <extension type="ADR01001:igML"> <!-- This provides EDI mapping -->
     <item type="Format">EDI X12</item>
     <item type="Message">142</item>
     <item type="SegmentRef">N1</item>
     <item type="DictSegment">N1</item>
     <item type="DictDataElement">98</item>
    </extension>
   <extension type="ADR01001:MSL"
      xmlns:xsd="http://www.w3.org/1999/XMLSchema">
    <xsd:complexType name="fullName">
     <xsd:element name="title" base="xsd:string" />
     <xsd:element name="firstName" base="xsd:string" />
     <xsd:element name="middleInitials" base="xsd:string" />
     <xsd:element name="familyName" base="xsd:string" />
     <xsd:attribute name="letters" base="xsd:NMTOKEN" use="fixed"value=","/>
    </xsd:complexType>
   </extension>
  </extensions>
  </bizcode>

  <bizcode qic="ADR01002" qic:base="addrLine" bizname="ADDR:street">
   <guide /> <!-- details go here -->
  </bizcode>
  <bizcode qic="ADR01003" qic:base="cityName" bizname="ADDR:city">
   <guide /> <!-- details go here -->
  </bizcode>
 </definitions>
 </xmlGuide>
```

The GUIDE element layer is designed to separate the datatyping and business formatting and semantic rule information from the GUIDE structure layer. The key to this mechanism is the use of qic references and the associated Bizcode business properties as illustrated above. A full discussion of the Bizcode concept is provided in the linking section below, and much background material is also available from http://www.bizcodes.org. (Bizcodes provide a simple referential system for information elements, in the same way as barcodes provide such a system for product items. Management and control of Bizcodes themselves is a separate topic and not covered in this GUIDE reference document. An ebXML implementer's reference will include information related to this once formal specifications are available).

The use of an IDREF compatible structure means that parsers can select just the particular piece of content definitions that they require. Also the use of local caching techniques removes the need to repeatedly access the same information via the network. Such performance behaviours are discussed in the section on GUIDE parser compliance.

We next discuss the GUIDE classification layer. The classification layer serves a different role to the previous two layers. It is concerned with the interoperability and understanding of the raw information content. As such it overlaps on the ebXML core component, business process and UML/XMI views of the information exchange. It is more focused on providing the human process view, but is equally applicable to future use by agent and object based machine-to-machine interchanges.

Compatibility note: by inference the GUIDE element definitions will also include ISO11179 element representations since ebXML is incorporating support for ISO11179, and ISO11179 provides the means to express legacy EDI codes and elements. Also GUIDE mask definitions support the ISO11179 format label constructs that are conceptually identical but just differ in physical semantics).

Interoperability note: It should also be noted that the physical control owner of the information in the GUIDE element layer definitions is the owner of the URI at which those definitions reside. Of course such an owner may deploy an ISO11179 or ebXML compliant repository internally to further manage ownership and version control of the information presented through the GUIDE element definitions.

Compatibility note: the current EDI system is not compatible with the igML.org definitions. This is being actively worked on and a new revision with full igML XML syntax compatibility will be made available.

2.4 Classification Layer Example.

Example 3 GUIDE classification for a mailing address

This example shows the definition of GUIDE classification definitions that describe and control the information within the GUIDE layers. Also the classification layer is actually separate from the previous layers since it references the structure definitions and the element definitions, but of course these themselves may be referenced from many classification structures as required.

```
<?xml version="1.0" ?>
  <xmlGuide use="classification" name="mailingAddress" version="0.1"
     xmlns:guide="http://www.ebXML.org/guides/address.xml">
   <classification>
     <description>Provides valid address format for use in mailing via regular mail.
     </description>
     <domain list="ebxml" value="eb1015" >Transport</domain>
     <owner
       list="ebxml" value="eb0134">USPS – United States Postal Service</owner>
     <process>Shipping</process>
     <intent>Define a valid mailing address format</intent>
     <context>Delivery</context>
     <usage type="Fragment"/>
     <level type="Leaf"/>
     <title>Mailing address</title>
     <action/>
     <next/>
     <previous/>
     <fail/>
     <method/>
     <UMLmodel/>
   </classification>
</xmlGuide>
```

This section of the GUIDE layer approach provides the most interesting potential. However, further definition of the exact business role here is required before continued work can be developed. Some items have already been noted in previous sections. A list is presented here as a start point for continued development (see section 3.0 for description of each semantic item). Once the business requirements are refined then the syntax here can be extended to support those.

Business functional requirements:

1) Support for process related information showing relationships between GUIDE structures and physical business processes and other GUIDE structures.
2) Business object level representations of GUIDE structure groups.
3) Support for process modelling technologies such as UML.
4) Ability to define domains to classify GUIDE structures and element definitions by industry and process.
5) Support ebXML core component and business process technical specifications.

It is anticipated that this section will develop of the next few weeks. Particularly concerning the linkage of this classification system to support ebXML business process and core component contextual syntax, and also possibly UDDI as well. Some early prototyping has been done in this area and more is required to explicitly determine exact XML structures to provide the business functionality required.

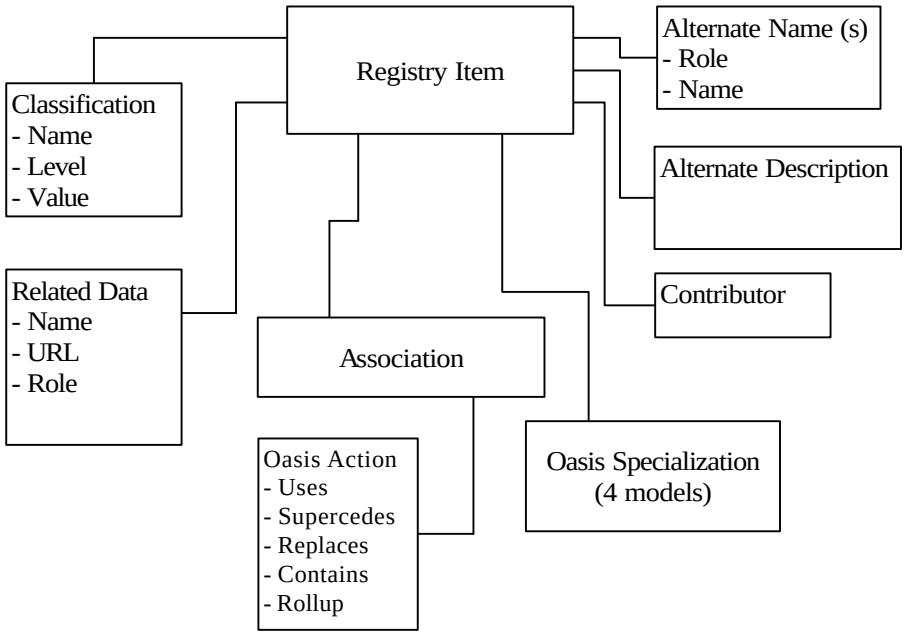
2.5 Relationship to the OASIS Registry system.

The GUIDE layers provide a natural overlay onto the classification system required by an OASIS compatible Registry system. To implement GUIDE within an OASIS registry requires that the GUIDE classification details be predefined within the OASIS registry as a set of defaults. Similarly an ebXML compatible registry change or query request can then be mapped into an OASIS equivalent based on OASIS classification and interface structures using the GUIDE approach as the harmonization bridge. Further work is underway to similarly provide a bridge to an ISO11179 compatible repository at the level of the element definition layer.

The following figure illustrates the OASIS classification model. By inspecting the GUIDE classification and element layers one can see that each facet of the OASIS model is provided for in GUIDE content. Thus OASIS has the formal specifications of registry content and GUIDE conforms to that information model. The difference is that the OASIS design is a generalized information model, while GUIDE is designed for business transactional information use such as ebXML provides.

It should be noted that additionally GUIDE has the ISO11179 owner and version context. Also GUIDE has extensions and transformation support that OASIS registry does not provide. By way of reference the current ebXML TPA work is also another classification system. The TPA system tracks both people and organizations and this can be associated by owner-reference to a GUIDE classification.

Figure 3. OASIS Registry Information Model



For more extended information on the OASIS registry specifications please see http://www.xml.org and associated content.

3. Relation to W3C XML V1.0 and W3C Schema

Generally speaking GUIDE describes behaviour as much as possible using simple XML V1.0 syntax, with use only of a limited subset of W3C schema and related XML Namespace, XLink and other work. GUIDE therefore strives to use a basic XML parser implementation to provide the required business functional behaviours. As such GUIDE may clarify or provided additional detail on specific parser behaviours. GUIDE is further designed to allow standards organizations to create definitive conformance test sets by providing a concise and business functional feature set.

4. GUIDE layers

GUIDE defines a layered approach for the information represented in a conforming semantic guide. Each of the three layers is now discussed starting from the top-most layer and working downward to the bottom.

4.1 GUIDE classifications

The classification layer is designed to provide a consistent re-use layer for the GUIDE information layers that it references. It is designed to also be the first point of contact to a GUIDE compatible interface for use by a human operator to determine the GUIDE interchanges that match their business use requirements. The GUIDE classification layer is also intended to provide management and control features to enable software agents to also access this information layer to manage business processes.

GUIDE classification semantics:

<description>	- human readable text that documents the purpose of the transaction.
<domain>	- business domain or industry that is the primary relation.
<owner>	- business organization responsible for management of this transaction.
<process>	- business functional process associated with the transaction.
<context>	- physical action being facilitated by the transaction.
<usage>	- the two values allowed here are Fragment or Standalone; a fragment is intended to be included into other standalone transactions.
<title>	-Descriptive human readable short title for the transaction.
<action>	-The action is either response, continue, or inform depending whether the transaction requires a response (two-way), is part of a workflow (continue), or inform (single-use).
<next>	- Another GUIDE transaction that directly relates to this one (response or continue actions).
<previous>	- Another GUIDE transaction that directly precedes this once (response or continue actions).
<fail>	- Should an error condition be detected in the current transaction, then this will indicate the transaction to be use to indicate the transaction failed. Any action will not then occur.
<method>	- An associated method for this transaction.
<UMLmodel>	- Container for an XMI based UML model describing this GUIDE classification item.

4.2 GUIDE structures

The GUIDE structure definitions model the actual information interchange structure required. Therefore they may function as a schema to define the physical XML transaction instances. They are designed to be simple to create with a minimum of automated editing tools being required, and to be human readable and concise. The GUIDE structure syntax contains only such semantics as necessary to define the physical interchange structure. All extended datatyping information is instead carried within the GUIDE element layer. Reference between the two layers is provided by domain neutral 'Bizcode' references, in the same way that business products use barcodes for a domain neutral definition system. A full discussion of GUIDE structure syntax is provided in the next section under linking including the keyword dictionary and the various supported information structures that can be modelled. The GUIDE structure syntax extends basic XML V1.0 hierarchical modelling syntax to include method constructs with objects and classes. Some examples are provided of these approaches.

GUIDE structure semantics

```
<xmlGuide use="structure" name="mailingAddress" version="0.1"
    xmlns:element="http://www.ebXML.org/guides/elements/postal.xml">
```

The GUIDE prolog attribute use="structure" identifies the particular layer, the name="" is a naming label that reflects the root element name within the XML transaction instance, the version="0.1" provides a versioning mechanism to allow selection of a particular version of a transaction.

```
<sequence>  - Indicates the start of a compound sequence dependent
structure component.
<element name="fullName"        qic:base="personDetails" />
```

The GUIDE element syntax defines an individual XML tag item, followed by a qic:base="" or qic="" definition of the associated element layer definition or mask datatyping. When the qic:base/qic definitions are both omitted then the element syntax defines a container item for further compound sequence structure. Can have an optional OCCURS attribute and LIMIT attribute where applicable. Such an item will then be followed by a further sequence tag to aid readability.

```
    <element name="street"          qic:base="postalStreet"
                        OCCURS="+" LIMIT="5" />
```

Note: **OCCURS=""** may have values of '*', '+', or 'optional'; the **LIMIT** qualifier is optional and is a single numeric value that denotes the upper bound occurrences.

4.2.1 Container Structure for GUIDE fragments

The GUIDE structure syntax allows the use of container structures with the use of the includeXML tag. An example is provided here.

Example 4. Container structure with GUIDE fragments.

```
<?xml version="1.0" ?>
<xmlGuide use="structure" name="travelItinery" version="0.1"
    xmlns:element="http://www.ebXML.org/guides/elements/travel.xml">
<!-- Declare the main structure of the transaction -->
   <sequence>
    <ELEMENT name="passenger"/>
    <ELEMENT name="itinerary"/>
    <ELEMENT name="car.rental" OCCURS="optional"/>
    <ELEMENT name="contact" OCCURS="optional"/>
    <!-- Local definitions of items to complete the whole transaction format -->
    <ELEMENT name="contact"    qic="?TRV01203" />
    <ELEMENT name="main"       qic="?TRV00230" />
    <ELEMENT name="fax"        qic="?TRV00230" />
    <ELEMENT name="mobile"     qic="?TRV00230" />
  </sequence>
<!-- Include in the GUIDE fragments, naming each root element to match the well-
formed XML usage -->
    <includeXML root="passenger" source="People-GUIDE.xml"
                lookup="SYSTEM" mimetype="text/XML" version="000" />
    <includeXML root="itinerary" source="Route-GUIDE.xml"
                lookup="SYSTEM" mimetype="text/XML" version="000" />
    <includeXML root="rental" source="Auto-GUIDE.xml"
                lookup="SYSTEM" mimetype="text/XML" version="001" />
</xmlGuide>
```

The use of includeXML is preferred to the various W3C Schema insert/include mechanisms draft proposals as it is simple, concise and supports versioning implicitly. The includeXML can of course be mapped to equivalent W3C mechanisms at a future point internally or via W3C schema complex typing mechanisms. This further enhances the value of includeXML as it can be used today, and maybe mapped to and used with enhanced linking in the future.

Next the GUIDE structure provides a number of extended capabilities for handling business process structure needs.

4.2.2 Extended GUIDE structure mechanisms

The basic structure definitions can be extended to provide the following structural functionality:

1) Open elements (for use with multiple business partners requiring local definitions that are known but unspecified).

2) Unordered structures of elements where order is not significant.

3) Associative datatyping based on data value context.

4) Elements with elements (attributes).

These behaviours of GUIDE structures are now described separately with an example of each use. The first example is the use of open elements. To achieve this the basic container structure approach for inserting GUIDE fragments is used with a namespace prefix on the particular open element definitions. Setting the namespace value then controls the specific substitution reference that occurs.

Example 5. Open elements using namespace definition.

```
<?xml version="1.0" ?>
<xmlGuide use="structure" name="travelItinery" version="0.1"
    xmlns:element="http://www.ebXML.org/guides/elements/travel.xml"
    xmlns:open="http://www.ota.org/guides/route.xml">
<!-- Declare the main structure of the transaction -->
  <sequence>
    <ELEMENT name="passenger"/>
    <ELEMENT name="itinerary"/>
    <ELEMENT name="car.rental" OCCURS="optional"/>
    <ELEMENT name="contact" OCCURS="optional"/>
    <!-- Local definitions of items to complete the whole transaction format -->
    <ELEMENT name="contact"    qic="?TRV01203" />
    <ELEMENT name="main"       qic="?TRV00230" />
    <ELEMENT name="fax"        qic="?TRV00230" />
    <ELEMENT name="mobile"     qic="?TRV00230" />
  </seqeunce>
<!-- Include in the GUIDE fragments, naming each root element to match the well-
formed XML usage -->
    <includeXML root="passenger" source="People-GUIDE.xml"
                lookup="SYSTEM" mimetype="text/XML" version="000" />
    <includeXML root="itinerary" source="open:Route-GUIDE.xml"
                lookup="SYSTEM" mimetype="text/XML" version="000" />
    <includeXML root="rental" source="Auto-GUIDE.xml"
                lookup="SYSTEM" mimetype="text/XML" version="001" />
</xmlGuide>
```

The <includeXML open:Route-GUIDE.xml reference is therefore dependent on the namespace URL reference. The next example illustrates the use of an unordered list of items; in unordered structures of elements where order is not significant.

To achieve this functionality the GUIDE structure uses a parameter on the sequence construct, this then infers that all items within the sequence block are optional (the default behaviour is items are required). Then items that are required must therefore be explicitly marked as such using the OCCURS construct.

Example 6. Unordered list of items using the 'sequence' construct.

```
<?xml version="1.0" ?>
<xmlGuide use="structure" name="travelItinery" version="0.1"
    xmlns:element="http://www.ebXML.org/guides/elements/travel.xml"
    xmlns:open="http://www.ota.org/guides/route.xml">
<!-- Declare the main structure of the transaction -->
   <sequence>
    <ELEMENT name="passenger"/>
    <ELEMENT name="itinerary"/>
    <ELEMENT name="car.rental"/>
    <ELEMENT name="contact"/>
    <!-- Local definitions of items to complete the whole transaction format -->
    <ELEMENT name="contact"    qic="?TRV01203" />
    <sequence order="any">
      <ELEMENT name="main"     qic="?TRV00230" OCCURS="+"  LIMIT="1" />
      <ELEMENT name="fax"      qic="?TRV00230" />
      <ELEMENT name="mobile"   qic="?TRV00230" />
    </sequence>
  </sequence>
</xmlGuide>
```

The next example illustrates the use of associative typing support.

This feature is designed to provide a context mechanism for data formatting directives. An example would be the difference between local access telephone number formats, as compared to an international telephone number format. The context is provided by an associative element that provides the context.

Example 7a. Associative datatyping support and nested elements (attributes).

```
<element name="telephone" qic:base="usDial | ukDial | otherDial" associate="dialformat" >
    <element name="dialformat" qic="TEL01001" />
</element>

<element name="usDial" qic:base="usTelephone" />
<element name="ukDial" qic:base="ukTelephone" />
<element name="otherDial" qic:base="genericTelephone" />
```

Therefore by setting the value of the 'dialformat' nested element (aka attribute) of the telephone element, the particular data format can be associated automatically. This example also illustrates the use of nested elements (attributes) within a GUIDE structure.

The example 7b shows the GUIDE element layer definition of the 'dialformat' associative item. Within the XML document instance itself you would simply see the <telephone> element and its associated dialformat nested element. So for a UK telephone number the result would simply be :

```
<telephone dialformat='UK' >1823-452121</telephone>
```

within the XML document and the correct formatting is automatically associated.

Example 7b. GUIDE element definition of the associative element.

```
<?xml version="1.0" ?>
<!--
* GUIDE element for use with associative element *
* reference system.                               *
*                                                 *
* Version 0.1  July,2000                          *
* Associative datatyping example                  *
*                                                 *
  -->
<xmlGuide use="element" name="xmlg:associatives" version="0.1"
xmlns:datatypes="http://www.ebXML.org/guides/associatives.xml"
xmlns:qic="http://www.ebXML.org/guides/bizcodes.xml">
   <definitions>
    <bizcode qic="TEL01001" bizname="dialformat">
     <guide>
      <status date="21/02/2000">candidate</status>
      <maxlength>5</maxlength><minlength>1</minlength>
      <datatype>string</datatype>
       <mask>X5</mask>
       <values default="US">
          <value>US</value><value>UK</value><value>Other</value>
       </values>
       <localdescription xml:lang="EN" xml:space="preserve">This is a nested
element for use with associative telephone number formatting.
</localdescription>
      <fulldescription xml:lang="EN" mimetype="XML" >
http://www.telephone.org/samples/TEL01001.XML</fulldescription>
      <labels>
        <label xml:lang="EN">Telephone Country</label>
     </labels>
      <seeAlso/>
     <dependencies>
      <dependent type="required">TEL01002</dependent>
      <dependent type="required">TEL01003</dependent>
      <dependent type="required">TEL01004</dependent>
    </dependencies>
  </guide>
   <extensions>
    <extension type="TEL01001:XForm">
     <item type="formcontrol">textfield</item>
    </extension>
  </extensions>
  </bizcode>
 </definitions>
</xmlGuide>
```

This concludes the section on extended GUIDE structure mechanisms.

4.3 GUIDE elements

The GUIDE element reference system is designed to support the traditional data dictionary functionality and provide the basis to migrate existing EDI code and element dictionaries to an XML syntax foundation. GUIDE elements are also designed to be the foundation for information sharing across industry domains by standardizing sets of definitions.

Also included in the element definitions are the dependencies and basic semantic checks on the data content. These are designed to allow either a compliant parser, or an associated business application to validate information content according to the definitions.

The GUIDE element definitions are also designed to facilitate transformation of information. This includes not only language representation changes, but also semantic changes. The element definitions therefore contain multiple representations through the use of the 'extension' concept, to extend the syntax supported. This mechanism is extensible to include any formatting that can be modelled using XML syntax. Typical examples include EDI, SQL, xhtml, XForm, and UML so that agent based technologies can create representations of information in whatever formats are provided by the GUIDE element definition extensions.

Datatyping is provided by a combination of primitive datatypes combined with the use of innovative rich XML mask syntax. The objective is to provide a concise, intuitive and human readable syntax for element definitions that can also be migrated to easily from legacy less semantically rich mask formats such as COBOL, RPG, CICS and so forth.

The default datatyping system will also be compatible with the W3C datatyping system, once this has been finalized as a recommendation, since each GUIDE primitive datatype and mask can be resolved as a machine-readable cryptic datatype as proposed currently in the W3C system.

The GUIDE element structure carries the essential information about an element, and so is variant of the ISO11179 data element table. This is broadly similar to the ISO11179 definitions of elements, but differs in that the mechanisms and syntax are designed to facilitate machine-to-machine XML interfacing rather than human data dictionary management. All other functionality, such as ownership, versioning, and other ISO11179 functionality can be managed from using additional control structures that are beyond the scope of GUIDE functionality.

GUIDE element semantics

```
<xmlGuide use="element" name="mailingAddress" version="0.1"
    xmlns:datatypes="http://www.ebXML.org/guides/datatypes.xml"

```

The GUIDE prolog attribute use="**element**" **identifies the particular layer, the** name="" **is a naming label that reflects the classification name associated with the XML element definitions, the** version="**0.1**" **provides a versioning mechanism to allow selection of a particular version of element definitions.**

<definitions> - **Denotes start of definitions within the XML instance.**
<bizcode ID="**ADR01001**" bizname="**PERSON:fullName**">

The Bizcode header provides an XML IDREF compliant token to perform a physical link to retrieve the particular fragment of the XML instance. Bizname provides a default tagname for this Bizcode item.

<guide> - **The prolog to the GUIDE definitions themselves.**
<maxlength> - **Maximum permitted number of characters.**
<minlength> - **Minimum permitted number of characters.**
<datatype> - **valid GUIDE datatype value.**
<mask> - **valid mask definition for the datatype and format.**
<values> - **outer container for value set where applicable.**
<value> - **specific value**
<Description xml:lang="**EN**" xml:space="**preserve**">
- **Human readable description of the item.**
<labels> - **Used when rendering the item to a form or report.**
<label xml:lang="**EN**"> - **Specific label content for a particular language.**
<seeAlso> - **outer container for related items.**
<similar> - **item from another business domain that equates.**
<equivalent> - **item from business domain that is a substitute.**
<related> - **item that is only related to this item for searches.**
<dependencies> - **outer container for dependent items.**
<dependent type="**required**">**ADR01002**</dependent>
- defines another Bizcode reference to an item that is required or optional relative to this item within a transaction.
<extensions> - **outer container for extended definitions.**
<extension type="**ADR01001:XForm**">
- **defines the particular type of extension (syntax) as an IDREF compliant link reference (allowing direct retrieval of this as a fragment). Format is the Bizcode:Type.**
<item type="**formcontrol**">**textfield**</item>
- **individual items from within the extension definition.** Reflects the specific syntax of the extension itself. There can be as many item details as required.

Next we need to understand how the three GUIDE layers interact with each other.

4.4 GUIDE linking

The linking mechanism used in GUIDE is based on namespaces. The reserved word *guide* namespace declared in the root tag of the XML transaction instance establishes the reference to the next GUIDE layer as needed. Therefore a XML transaction will use the guide namespace to reference the GUIDE structure schema that defines the structural rules, and the GUIDE structure will in turn use its own *element* namespace to locate the default element definitions associated with the structure. The element definitions can also optionally access the *datatypes* namespace to locate datatyping information. This provides an extensible datatype model.

To provide the equivalent of fragments processing, a special *include* tag is provided. However, fragments that are themselves included, may not have further *include* tags within them, thus ensuring that only one level of nesting is provided. Furthermore, permitting only the single guide namespace with a single control structure ensures that the true structure of transactions is available and exposed. This contrasts with other early schema implementations that used in-line namespace definitions to retrieve multiple structure schemas, thus creating a system where the true transaction structure could not be determined. GUIDE avoids this by only allowing the single guide namespace for including the structure linkage.

This linkage mechanism is designed to be simple and business functional and to avoid any complex constructs that make parser implementation and behaviour complex or uncertain. This necessarily restricts the complex use of cascading links, and in particularly linking can only be nested one layer deep, and all recursive references are explicitly not provided.

For legacy compatibility GUIDE linking can also be achieved using http style query/response requests when using DTD references as illustrated in the addendum. These interchanges can be done using ebXML repository API conformant query/response mechanisms once these technical specifications are available, or using W3C Protocol (new working group) compliant mechanisms once those specifications are available.

4.5 Type systems

The GUIDE element definitions use basic business datatypes. All of these are supported by the current W3C datatyping proposal, however the W3C has extended complex behaviours in their datatyping that are not required for GUIDE business datatypes. The table here shows the explicit GUIDE datatypes that are used in the GUIDE element layer definition syntax and their equivalent W3C types. GUIDE typing is provided in a simple syntax that is easier to use when combined with and associated XML mask. This syntax can then be equated to W3C datatyping as required internally by parsing software.

GUIDE	W3C
string	string
numeric (includes integer and decimal)	number
logical	boolean
date	datetime
time	datetime
text	string with space=`preserve`

Any item that does not have a datatype explicitly assigned is treated as a simple string by default. See addendum section on masks for how default datatyping is also associated with explicit mask definitions by default.

5. GUIDE implementation

The GUIDE system has been designed to allow the use of a basic XML parser compatible to XML V1.0 with extensions for namespaces, and ability to recognize basic schema and datatyping syntax extensions. Such extensions are designed to be a minimal subset of the full W3C recommendations to minimize the implementation burden and ensure consistent behaviour. This technique is familiar to implementers in the HTML environment where extended features are avoided to ensure consistent behaviour across platforms and product implementations. A specific set of functionality will be documented in the appendix once the formal W3C specifications are available. Additionally it is envisioned that GUIDE compatible methods implemented in Java and C++ will be available to simply link into a Java or C++ parser implementation to provide a GUIDE compatible parser by taking advantage of the open architecture that the W3C DOM (document object model) specifications provide in a XML compliant parser implementation.

Furthermore a limited but powerful base functionality of the GUIDE system can be built today using any DOM compliant XML parser implementation and a scripting language with access to the DOM, such as JavaScript. Examples of this use can be found in the tutorial sample forms, see section 7.

5.1 Transactions

The GUIDE specifications are designed to provide the means for industries to develop compatible business transactions. Transactions themselves can be structured to match either a single business interaction, or a series of related interactions. The GUIDE classification layer provides the means for industries to document and specify such sets of related transactions and make them available in a consist format that can be reused. Also GUIDE compatible parsers and business applications can then have available all the needed business semantics to be able to process and control such transactions.

5.2 Relationship of and use of Bizcodes

The Qualified Indicator Code (QIC) is tied into the Bizcode mechanism that provides the linkage between GUIDE structures and the associated element definitions and is designed to be a neutral reference code. Use of neutral reference codes is already an established practice within dictionaries of industry element definitions. Therefore many industries already have codes that they can use as QIC references.

The preferred Bizcode QIC structure is a three-letter code, followed by a five-digit number, where the three-letter code denotes the industry or group assigning the codes, and the five-digit number is a sequentially assigned value. It is anticipated that as part of the ebXML repository technical specifications there will also be guidelines established for managing globally unique names under which Bizcode QIC references can be classified.

Currently the barcodes used for product labelling are managed in a similar fashion by having formally registered barcodes alongside locally defined barcodes. With Bizcode QIC labels, since they are tightly coupled to a GUIDE structure and also stored within a GUIDE element repository this already provides excellent separation to avoid conflicts on QIC values assigned within an industry. Also, unlike barcodes where there are many tens of millions already assigned, Bizcodes required a much more limited number since they are reusable across many products. An example is the food industry where there are over seven million barcodes in use, but less than ten thousand unique element definitions (product attributes) are being used to describe all those products.

The current GUIDE element structure is designed to be compatible with ISO11179 based reference registries. The role of ISO11179 registries is to harmonize information classification within a corporation or large government agency for human analytical and business system design purposes. The role of ebXML and GUIDE repositories extends beyond that to include XML based machine-to-machine information interchanges that reference XML repositories via an XML based API and interface specifications.

Therefore GUIDE can be used in tandem with ISO11179, where the ISO registry manages the content that the GUIDE system exposes to ebXML aware systems.

5.3 Operations and Verbs

Within a transaction there may be an associated action, such as ‘confirm’, or ‘respond’ that the trading partner requires. The ‘action’ tag has been provided with the classification layer, along with next, previous and fail. These are designed to allow a process to be defined and for software agents to then interact with these control structures and the actual physical process itself. More work is needed in this area however to provide a complete specification and the business functional needs that are required to be met.

5.3.1 One-way operation

TBD.

5.3.2 Request-response operation

TBD.

5.4 GUIDE compliant parser implementation

A GUIDE compliant parser is essentially an XML V1.0 parser with some extensions to support the limited schema syntax and IDREF links that GUIDE requires. Therefore a GUIDE compliant parser is simpler to implement, while providing a full range of business transaction interchange capabilities and support.

5.5 GUIDE conformance testing

The set of conformance suites will be available as an extension of the current NIST XML conformance testing work.

5.6 Reusable Bindings

The GUIDE structure system provides support for reusable binding. These mechanisms are supported by the GUIDE structure system. Elements may have a qic:base reference instead of hard-coded typing definitions. This may also indicate that the item referenced has a complex structure rather than a single element structure.

To maintain the separation construct of layers, the ability to redefine and re-use element definitions must be controlled within the element layer, while structure redefinitions are within the structure layer. Optionally classification entries should be created to manage such extended use definitions to fully document the context and details of such interchanges.

6. Orchestration

The earlier SOAP specification calls for a complete business orchestration language, that is to be defined. GUIDE is well positioned to make this functionality available to ebXML compliant interchanges. However this initial release is designed with a limited scope to ensure that consistent interchanges can be engineered within a realistic timeframe. Subsequent phases of development can extend the GUIDE classification layer to include more business orchestration features.

7. Tutorial and Use Case

This section presents a short example by the way of an illustration of how to work with and prepare a GUIDE transaction. The example uses the DTD syntax (see addendum notes) and provides the source code so that you may test this right now with a working example. You will require Microsoft IE5.0 or later and an internet connection to test the live version. (status: work in progress, TBC).

The tutorial provides a significant aspect of the GUIDE approach, namely making the whole process of defining an interchange intuitive and straightforward. The major steps in the tutorial are:

1) Document the purpose of the GUIDE to be created, the owner, and how it relates to other GUIDE transactions and the overall business process and context. An HTML form allows user to quickly enter content and then generate a valid GUIDE XML classification instance.
2) Create a well-formed XML document instance that represents a sample GUIDE transaction. An HTML form allows the user to quickly enter this and reviews and displays the element list as they are entering the content. Once complete, form generates a basic XML GUIDE structure schema and associated DTD that matches the well-formed document instance structure.
3) Using the element list from step 2 create qic:base definitions for each of the elements. This may a locally defined qic:base set, or may involve referencing one or more XML repositories to map the element list to the industry standard qic:base definitions.
4) Publish the completed GUIDE set of classification, structure and element layers as a working draft to an appropriate industry XML repository.

8. Addendum

A 1. References

W3C Working Draft "XML Schema Part 1: Structures". This is work in progress.

W3C Working Draft "XML Schema Part 2: Datatypes". This is work in progress.

A 1.1 Notes on URI, XML namespaces & schema locations

Namespace use to be defined with regard to the W3C namespace recommendation.

A 1.2 Relative URIs

Throughout this document you see fully qualified URIs used as references. The use of a fully qualified URI is simply to illustrate the referencing concepts.

B 1. Notes on MASKS and patterns

The text here provides a base specification of mask syntax for use with GUIDE elements and structures. It should be noted that this picture mask syntax is highly sophisticated and has been in common use for over ten years in business applications. As such this is a robust and proven method that has already transcended earlier crude and limited mask systems such as that found in COBOL. The text here provides the necessary behaviourable details for implementers to describe the exact usage.

Each picture mask type has an associated implied primitive datatype associated with it: date has date type; time has time type; number has numeric type; logical has Boolean type; and all others a basic string type.

B 1.1 Picture Masks

These are categorized by the basic datatyping element that they can be used in combination with. Content that already conforms to the mask is not modified but simply placed in the DOM as is. Content that does not conform to the mask (such as text in a numeric field) results in '*' characters being placed in the DOM to the full length of the mask, so 'ABC' in a field defined as #6.## would result in '*********', and so on.

B 1.2 String Type Pictures

Examples of string type pictures

XML Element Contents	*Picture Mask (shorthand)*	*Full Expanded Mask*	*Resulting DOM Content*
portability	X6	XXXXXX	portab
portability	UX3	UXXX	Port
portability	XXXXing	XXXXing	porting
realtime	XXXX-XXXX	XXXX-XXXX	real-time
BOLD!	L5	LLLLL	bold!

B 1.3 String Type Pictures

The positional directives and mask characters as explained below.

Directive Character	*Holds a Place for*
X	any character.
U	a character to be converted to upper case.
L	a character to be converted to lower case.
#	a digit (0-9) only.

B 1.4 Numeric Pictures

The following are examples of Numeric pictures.

Data contents of XML element	*Picture*	*Result*
-1234.56	######.##	^^1234.56
-1234.56	N######.##	^^-1234.56
-1234.56	N###,###.##C	^^-1,234.56
-1234.56	N######.##L	-1234.56^^
-1234.56	N######.##P*	-**1234.56
0	N######.##Z*	**********
-13.5	N##.##-DB;	DB13.50
45.3	N##.##+CR;	CR45.30
-13.5	N##.##-(,);	(13.50)
4055.3	$######.##	$^^4055.30

(The ^ symbol represents one space character.)

B 1.5 Positional directives for Numeric pictures

Character	*Holds a place for*
#	holds a place for a digit.
.	indicates the location of the decimal point. For example, '####.###' defines a numeric variable of four whole digits and three decimal digits.

B 1.6 Date Pictures

The typical date formats are MM/DD/YYYY, DD/MM/YYYY or YYYY/MM/DD (American, European, Scandinavian).

B 1.7 Examples of Date Pictures

The date used in the following examples is March 21, 1992.

Picture	*Result and Notes*

MM/DD/YY	03/21/92
DD/MM/YY	21/03/92
YY/MM/DD	92/03/21
DD/MM/YY	21-03-92 when the Environment 'Date Separator' parameter is set to '-'
DD-MM-YYYY	21-03-1992 where '-' is a mask character
YY.DDD	92.081
##/##/##	03/21/92, when XML parser default is set to American, 21/03/92, when XML parser is set to European.
MMMMMMMMMM^DDDD, ^YYYY	March^^^^^^21st,^1992
MMMMMMMMMM^DDDD, ^YYYYT	March^21st,^1992 with trimming directive
WWWWWWWWW^-^W	Saturday^^^-^7
WWWWWWWWW^-^WT	Saturday^-^7 with trimming directive

(The ^ symbol represents one space character.)

Trim Text

Invoked by adding the directive T to the variable picture. This directive instructs XML parser to remove any blanks created by the positional directives 'WWW...' (weekday name), 'MMM...' (month name), or 'DDDD' (ordinal day, e.g. 4th, 23rd). Since these positional directives must be specified in the picture string using the maximum length possible, unwanted blanks may be inadvertently created for names shorter than the specified length. The Trim Text directive will remove all such blanks.

If a space is required nevertheless, it must be explicitly inserted in the picture string as a mask character, (the ^ symbol is used to indicate a blank character), e.g., 'TWWWWWWWWW^DDDD MMMMMMMMM,^YYYY'

Zero Fill

Invoked by adding the functional directive Z to the variable picture. This directive instructs XML parser to fill the entire displayed variable, if its value is zero, with the "Character:" value. If you don't specify any character the variable is filled with blanks.

Picture Mask Date

When you define the attribute Date for a variable, you must also select the format for the date item (see below). You can change this default picture and place in it any positional directives and mask characters you need.

B 1.8 Positional Directives for Date Pictures

Picture	*Meaning*
DD	A place holder for the number of the day in a month
DDD	The number of the day in a year
DDDD	The relative day number in a month
MM	A place holder for the number of the month in a year
MMM...	Month displayed in full name form (up to 10 'M's in a sequence). e.g. January, February. If the month name is shorter than the number 'M's in the string, the rest of the 'M' positions are filled with blanks. N/A
YY	A place holder of the number of the year
YYYY	A place holder for the number of the year, represented in full format (e.g. 1993)
W	Day number in a week
WWW...	Name of day in a week. The string can be from 3 to 10 'W's. If the name of the day is shorter than the number of 'W's in the string, the rest is filled with blanks.
/	Date separator position.
-	Date separator position (alternate).
MM/DD/YYYY	Full American format date.

B 1.9 Time Picture Masks

The XML parser defines the default picture mask HH/MM/SS for an element of datatype Time

Examples of Time Pictures

Picture	*Result*	*Comments*
HH:MM:SS	08:20:00	Time displayed on 24-hour

		clock.
HH:MM:SS	16:40:00	Time displayed on 24-hour clock.
HH:MM PM	8:20 am	Time displayed on 12-hour clock.
HH:MM PM	4:40 pm	Time displayed on 12-hour clock.
HH-MM-SS	16-40-00	Using Time Separator of '-'

Time Masks

Additional notes on positional directives for Time pictures

Directive Character(s)	*Function*	*Legal Range of Values*
HH	Place holder for the hour	00-99
MM	Place holder for the minutes	00-59
SS	Place holder for the seconds	00-59
PM	Place holder for the AM/PM attribute. PM restricts the maximum value of the HH directive to 12	AM or PM

C 1. Notational Conventions

The keywords "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].

The namespace prefixes “xmlbc” and "xmlg" used in this document are associated with the GUIDE namespaces "http://xmlguide.org/bizcodes/", and "http://xmlguide.org/guide/".

Throughout this document, the namespace prefix "xsi" is assumed to be associated with the URI "http://www.w3.org/1999/XMLSchema-instance" which is defined in the XML Schemas specification [11]. Similarly, the namespace prefix "xsd" is assumed to be associated with the URI "http://www.w3.org/1999/XMLSchema" which is defined in [10]. The namespace prefix "tns" is used to indicate whatever is the target namespace of the current document. All other namespace prefixes are samples only. In particular, URIs starting with “http://my.org” represent some application-dependent or context-dependent URI [4].

This specification uses an informal syntax to describe the XML grammar of a guide:

- The syntax appears as an XML instance, but the values indicate the data types instead of values.
- Characters are appended to elements and attributes as follows: “?” (0 or 1), “*” (0 or more), “+” (1 or more).
- Elements names ending in “…” (such as <element…/> or <element…>) indicate that elements/attributes irrelevant to the context are being omitted.
- Grammar in bold has not been introduced earlier in the document, or is of particular interest in an example.
- <-- extensibility element --> is a placeholder for elements from some “other” namespace (like ##other in XSD).
- Examples starting with <?xml contain enough information to conform to this specification; others examples are fragments and require additional information to be specified in order to conform.

D 1. Example of using GUIDE structure with DTD syntax.

This same example as previously illustrated for a GUIDE simple physical structure model with a repeated group of information is now used to illustrate the use of GUIDE with a DTD approach where a GUIDE compliant parser is not available. Being able to use the GUIDE approach using a DTD and a scripting language such as JavaScript means not only backward compatibility with XML V1.0, but also that GUIDE interchanges can be constructed with today's development tools and environments.

Example 8. Using GUIDE with a DTD.

```
<?xml version="1.0" ?>
<!DOCTYPE mailAddress SYSTEM
  "http://www.ebXML.org/guidesdtd/address.dtd" []>
 <mailAddress>
   <fullName>Joe H. Smith</fullName>
   <street>101 Main Street</street>
   <street>Apartment 20b</street>
   <city>Taggtown</city>
   <ZIP>10230-0001</ZIP>
   <state>AZ</state>
   <accountActive>YES</accountActive>
 </mailAddress>
```

The GUIDE structure DTD that is referenced above in Example 8 is shown here.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT mailAddress (fullName, street+, city, ZIP, state, accountActive)>
<!-- establish link to qic reference location -->
<!ATTLIST mailAddress
        qicref CDATA #FIXED 'http://www.ebXML.org/qic/datatypes.xml'>
<!ELEMENT ZIP (#PCDATA)>
<!ATTLIST ZIP
       qic CDATA #FIXED '#####-####'>
<!ELEMENT accountActive (#PCDATA)>
<!ATTLIST accountActive
       qic CDATA #FIXED 'U3'>
<!ELEMENT city (#PCDATA)>
<!ATTLIST city
       qic CDATA #FIXED '?ADR01003' qic_mask CDATA #FIXED 'UX19'>
<!ELEMENT fullName (#PCDATA)>
<!ATTLIST fullName
       qic CDATA #FIXED '?ADR01004'>
<!ELEMENT state (#PCDATA)>
<!ATTLIST state
       qic CDATA #FIXED '?ADR01005'>
<!ELEMENT street (#PCDATA)>
<!ATTLIST street
       qic CDATA #FIXED '?ADR01002'  LIMIT NMTOKEN #FIXED '5' >
```

The qic attributes provide the means for the JavaScript or similar language tool that can access the XML DOM of the parser to easily retrieve the information needed to provide the GUIDE compliant referencing mechanisms support. The example shown is supported by the Microsoft IE5.0 environment and can work in either local or remote accessing modes. See section 4.4 on GUIDE linking for more details.