GJXDM Information Exchange Package Methodology, Naming and Design Rules (MNDR)

Section VI – Schema Set Naming and Design Rules

Section VII – Instance Naming and Design Rules

Working Draft .08, 13 May 2005 (FTF & Conference Call Updates)

Document identifier:

wd-ijtc-mndr-sec6-sec7-v[1].08.doc

Location:

http://www.oasis-open.org/apps/org/workgroup/legalxml-intj-exmndr/

Editor:

[List your editors here; check whether "Editor" header should be plural]
Scott Came, Justice Integrated Solutions, Inc. <scott@justiceintegration.com>

Contributors:

[List your contributors here]
[Optionally list them in the Acknowledgments appendix instead]
John Ruegg, LA County Information Systems Advisory Body (ISAB)
Scott Came, Justice Integrated Solutions, Inc
Tom Carlson, National Center for State Courts

Abstract:

[Supply your own summary of the technical purpose of the document.] This document provides a working MS Word 2000 sample from which you can start editing your own OASIS-published document. Instructions are provided as italic text in brackets, which should be deleted before publication. Full instructions are provided in the body of the document.

Status:

It should be noted that this first draft will assume adoption of existing Global JXDM standards as introduced in GTRI training materials, applicable UBL standards and relevant OASIS Integrated Justice TC draft documents.

However, a second draft could be developed for GTRI and Global review which would adopt most if not all UBL Naming and Design Standards such that Global JXDM could become a UBL conformant (or compatible?) industry specific standard. This fully UBL compliant version of Global JXDM would afford utilization of UBL standard based automated tools for future development. The major changes for UBL conformance would require mapping existing Global JXDM types into the five UBL Component types which are built (as is done in Global JXDM) on ebXML core components. These types include Unspecialized DataTypes (udt:), Specialized DataTypes (sdt:), Core Component Types (cct:), Basic Business Information Entities (BBIE) and Aggregate Business Information Entities (ABIE's , ASBIE's). This component xsd:annotation standard is documented in the UBL Core Component Parameter schema (ccts:). Although this classification of Global JXDM types would not be difficult to complete and could even be done programatically, Global JXDM policy currently does not permit modification (even extensions via xsd:annotation) of the Global JXDM dictionary. Additionally, current practice only permits segregation of Global JXDM schema into a document schema, user-defined extension schema, subset schema and constraint schema. UBL, on the other hand, creates the five component types noted above and creates a schema module for each component. For example, in UBL, all BBIE's are combined into a schema module called "UBL-CommonBasicComponents-1.0.xsd".

[Describe the status and stability of the specification and where to send comments.] This document is updated periodically on no particular schedule. Send comments to the editor.

[This is boilerplate; to use, fix the hyperlinks:] Committee members should send comments on this specification to the legalxml-intj-exmndr@lists.oasis-open.org list. Others should subscribe to and send comments to the xxx-comment@lists.oasis-open.org list. To subscribe, send an email message to xxx-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

[This is boilerplate; to use, fix the hyperlinks:] For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the XXX TC web page (http://www.oasis-open.org/committees/xxx/).

[If a Committee Specification or OASIS Standard:] The errata page for this specification is at http://www.oasis-open.org/committees/xxx/yyy.

Table of Contents

1 Introduction

[Provide an introductory chapter, indicating if any parts of it are non-normative.]

1.1 Terminology

[The following is boilerplate. Most specifications will need this and the corresponding bibliography entry.] 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 **[RFC2119]**.

2 Word Styles

[This section is provided to explain and demonstrate the styles available in the Word template attached to this sample document. It is important to use the styles provided in the template consistently and to avoid defining new styles or using raw formatting.

Delete this entire section when using this sample document to begin writing a new specification.]

2.1 Overall Style

The paper size is set to **Letter**, which is 8 ½ x 11. You may change this to **A4** or whatever other size suits your needs.

The document identifier and publication date information in the footer needs to be updated every time you publish.

Line numbers are enabled by default for easy reference by specification commenters. You may turn line numbering off.

2.2 Title Page

The title page is designed to fit a lot of metadata compactly. If you wish to create a “true” title page, you may insert a page break after the subtitle.

2.3 Headings

Heading 1 through **Heading 9** and **AppendixHeading** have been defined with a special appearance. Headings are numbered and appear in the Table of Contents. Pressing Return after a heading inserts a **Normal** paragraph style directly after.

This template sets **Heading 1** and **AppendixHeading** to start on a new page. You may set the **Heading 1** style not to start on a new page if you wish. Major headings have a horizontal rule above them.

2.4 Paragraphs

The font in the **Normal** paragraph style is 10-pt Arial. You may change this to 11-pt **Times New Roman** if you prefer a serif font; changing these two settings should change all the other relevant styles.

2.5 Lists

The **Definition term** and **Definition** paragraph styles are defined specially for this template. They produce a definition list with a hanging appearance. Pressing Return after one inserts the other directly after.

Definition term

Definition for the term.

Use **List bullet** for first-level bulleted lists. Use **List bullet 2** for second-level bulleted lists. Use **List continue** for continued paragraphs in list items.

- List bullet

  List continue.

  – List bullet 2

List continue 2.

For bibliography lists, use the **Ref** paragraph style. Use the **Ref term** character style for the bracketed text that serves as the bibliography entry key, and make each reference term into a bookmark for use as references from the text. For example, **[RFC2119]** is a generated cross-reference to the IETF RFC 2119 bibliography entry in Section 3.1ection 3.1 of this sample.

2.6 Tables

Use the following style for most tables: [To be supplied; suggestions welcome!]

2.7 Code Examples

For schema code and other normative code, use the **Code** paragraph style. It fits 71 characters. For example:

```
12345678901234567890123456789012345678901234567890123456789012345678901
         1         2         3         4         5         6         7
<simpleType name="DecisionType">
    <restriction base="string">
        <enumeration value="Permit"/>
        <enumeration value="Deny"/>
        <enumeration value="Indeterminate"/>
    </restriction>
</simpleType>
```

Use the **Code small** style if the code has very long lines. It fits 80 characters. For example:

```
123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
         1         2         3         4         5         6         7         8
<simpleType name="DecisionType">
    <restriction base="string">
        <enumeration value="Permit"/>
        <enumeration value="Deny"/>
        <enumeration value="Indeterminate"/>
    </restriction>
</simpleType>
```

For non-normative examples, use the **Example** paragraph style. For example:

GET http://<host name and path>?TARGET=<Target>...<HTTP-Version>
<other HTTP 1.0 or 1.1 components>

Use the **Example small** style if the example has very long lines. For example:

GET http://<host name and path>?TARGET=<Target>...<HTTP-Version>
<other HTTP 1.0 or 1.1 components>

2.8 Character Styles

This template defines several character styles for general text use:

- **Element** style (shortcut Ctrl-Shift-E) for `<NativeElement>` names and `<ns:ForeignElement>` names; add the angle brackets yourself
- **Attribute** style (shortcut Ctrl-Shift-A) for `attributeNames`
- **Datatype** style (shortcut Ctrl-Shift-Alt-D) for `DataType` names
- **Keyword** style (shortcut Ctrl-Shift-K) for `OtherKeyword` names
- **Variable** style (shortcut Ctrl-Shift-Alt-V) for *variable* names

3 References

3.1 Normative

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

4 STEP 1 – Domain Model Naming and Design Rules

5 STEP 2 – Domain Model – GJXDM Mapping Rules

6 Schema Set Naming and Design Rules

6.1 Sources

(all sources, including hyperlinks, will be moved to a consolidated reference section; In addition, the principal sources will also be listed at the beginning of the document;)

The following sources were utilized to develop this section of the GJXDM MNDR document:

- GTRI GJXDM Developers Workshop training material, May 2004
- Oasis Universal Business Language (UBL) Naming & Design Rules, 15 November 2004, Document-id: cd-UBL-NDR-1.0.1
- Guidelines For The Customization of UBL v1.0 Schemas, Working Draft 1.0, 04/22/04
- Department of the Navy XML Naming & Design Rules, Final Version 2.0, January 2005 (modeled after UBL Naming & Design Rules)
- OASIS Integrated Justice Working Draft – LegalXML Integrated Justice Technical Committee Methods and Options for Building Reference Documents, 19 September 2004

6.2 Schema Types and Definitions

(move and consolidate these definitions to the references defined in Section 2)

The following schema types are required for implementation of the GJXDM MNDR:

- ***GJXDM Reference Schema*** – the baseline Justice XML dictionary schema upon which GJXDM conformant document instance(s) are built.
- ***GJXDM Subset Schema*** – a conformant subset of the full GJXDM Reference Schema which may be used in-lieu of the full GJXDM Reference Schema to develop GJXDM conformant document instances.
- ***GJXDM Constraint Schema*** – a GJXDM Subset Schema with local constraints applied to the GJXDM elements and/or GJXDM complextype(s) and simpletype(s). No new elements are permitted to be defined within the GJXDM constraint schema.
- ***GJXDM Extension Schema*** – a user-defined schema comprised of ComplexTypes and SimpleTypes derived from GJXDM complex & simpleTypes using xsd:extension and xsd:restriction methods.
- ***GJXDM Document Schema*** – a user-defined schema to validate a GJXDM instance document. The document schema may optionally import a GJXDM Extension schema or directly import a GJXDM subset/constraint schema.

6.3 General Schema Set Naming and Design Rules

Schema language provides many redundant features that allow a developer to represent a logical data model many different ways. Heterogeneous data models can become an interoperability problem in the absence of a comprehensive set of naming, definition, and declaration design rules.

This section establishes rules for XML schema elements, attributes, and type creation. Because the W3C XML specifications are flexible, comprehensive rules are needed to achieve a balance between establishing uniform schema design while still providing developers flexibility across the Justice and Public Safety domain.

Adherence to these rules will ensure that semantics are unambiguous, enabling the practitioner teams to conduct straightforward comparisons and make recommendations with respect to enterprise reusability across their respective organizations. GJXDM information exchange schema(s) and XML Instances rules are modeled after the same naming and design conventions used to develop the Global Justice XML Data Model (GJXDM).

6.3.1 General Naming Rules

STA1-STA2, GNR1-GNR8, MDC1

The W3C XML Schema Definition Language has become the generally accepted schema language that is experiencing the most widespread adoption. Although other schema languages exist that offer their own advantages and disadvantages, DOJ-Global has determined that the best approach for developing a national XML exchange standard is to base its work on W3C XSD.

[STA1]	All Global JXDM information exchange schema design rules MUST be based on the W3C XML Schema 1.0 Recommendations: XML Schema Part 1: Structures and XML Schema Part 2: Datatypes.

A W3C technical specification holding recommended status represents consensus within the W3C and has the W3C Director's stamp of approval. Recommendations are appropriate for widespread deployment and promote W3C's mission. Before the Director approves a recommendation, it must show an alignment with the W3C architecture. By aligning with W3C specifications holding recommended status, DOJ-Global can ensure that its products and deliverables are well suited for use by the widest possible audience with the best availability of common support tools.

[STA2]	All Global JXDM information exchange schema and payloads MUST NOT be based on the W3C suite of technical specifications holding less than recommended status.

The English language has many spelling variations for the same word. For example, American English “program” has a corresponding British spelling “programme.” This variation has the

potential to cause interoperability problems when exchanging XML components because of the different names used by the same elements. Providing a dictionary standard for spelling will mitigate this potential interoperability issue.

[GNR1]	GJXDM information exchange XML elements, attributes and type names MUST be composed in the English language, using the primary English spellings provided in the Oxford English Dictionary.

The ebXML Core Component Technical Specifications (CCTS) provides a rule set for precisely defining the semantics of a data element in terms of a tripartite naming convention specified by ISO 11179 Part 5 (object class, [qualifiers], property term, and representation term). The three parts are combined to form the names of GJXDM information exchange XML elements, complex types, and attributes.

[GNR2]	GJXDM information exchange XML element, attribute and type names MUST be ebXML CCTS ISO 11179 compliant
[GNR3]	GJXDM information exchange XML element, attribute and type names MUST NOT include spaces, other separators, or characters not allowed by W3C XML 1.0 for XML names. The only **exception** to this rule is the use of periods in GJXDM element names.

(NEW RULE)

Element names and the derivative names MUST be consistent with the GJXDM, even when this would result in conflicts with GNR1 or GNR2.

Acronyms and abbreviations impact semantic interoperability and are to be

avoided to the maximum extent practicable. Since some abbreviations will inevitably be necessary, GJXDM maintains a normative list of authorized acronyms and abbreviations.

Appendix B provides the current list of permissible acronyms, abbreviations and word

truncations. The intent of this restriction is to facilitate the use of common semantics and to foster greater understanding. Appendix B is a living document and will be updated by the Global XSTF task force to reflect growing requirements.

[GNR4]	GJXDM information exchange XML element, attribute, and simple and complex type names MUST NOT use acronyms, abbreviations, or other word truncations, except those in the list of exceptions published in Appendix B.
[GNR5]	The acronyms and abbreviations listed in Appendix B MUST always be used.

Generally speaking, the names for GJXDM information exchange XML constructs must always be singular. The only exception permissible is where the concept itself is pluralized.

[GNR6]	GJXDM information exchange XML element, attribute and type names MUST be in singular form unless the concept itself is plural.

Example:

PersonPhysicalFeature, PhysicalFeatureType
PersonPhysicalDetails, PersonPhysicalDetailsType
personNameInitialIndicator

XML is case sensitive. Consistency in the use of case for a specific XML component (element, attribute, type) is essential to ensure every occurrence of a component is treated the same. This is especially true in a business-based data-centric environment such as that addressed by GJXDM. Additionally, the use of visualization mechanisms such as capitalization techniques assist in ease of readability and ensure consistency in application and semantic clarity. The ebXML architecture document specifies a standard use of upper and lower camel case for expressing XML elements and attributes respectively.[12] GJXDM will adhere to the ebXML standard. Specifically, UBL element and type names will be in UpperCamelCase (UCC).

[GNR7]	The UpperCamelCase (UCC) convention MUST be used for naming elements and types.

Example:

PersonName
JewelryStone

GJXDM information exchange attribute names will be in lowerCamelCase (LCC).

[GNR8]	The lowerCamelCase (LCC) convention MUST be used for naming attributes.

Example:

amountCurrencyCodeListVersionID
characterSetCode

GJXDM instance documents are designed to effect data and document electronic exchanges. Including mixed content in exchange documents is undesirable because exchange transactions are based on exchange of discrete pieces of data that must be clearly unambiguous. The white space aspects of mixed content make processing unnecessarily difficult and add a layer of complexity not desirable in information exchanges.

[MDC1]	Mixed content MUST NOT be used except where contained in an xsd:documentation element.

6.3.2 Schema File Structure

GXS1-GXS3,(Complete after discussion with Subcommitee),GXS4-GXS6,GXS15-GXS16

Reviewers: Provide Comments to GXS1-GXS3 where defined in Appendix A

The features of W3C XML Schema allow for flexibility of use for many different and varied types of implementation. The GJXDM information exchange MNDR uses the following rules to allow for a more consistent use of these features:

[GXS4]	The root element in all GJXDM information exchange Schema modules MUST contain the following namespace declaration: "xmlns:xsd=http://www.w3.org/2001/XMLSchema."

To avoid overloading implementation systems with unnecessary documentation, developers have an option to create a schema without the documentation. This schema is for run-time and must be a functional equivalent of the documented version.

[GXS5]	GJXDM information exchange schema developers MAY provide a run-time schema devoid of documentation in addition to the fully annotated version.

6.3.2.1 Built-in Simple Types

There are 44 simple types built into XML Schema. They are specified in Part 2 of the XML Schema Recommendation. These built-in types were used in the designing of GJXDM schema. Simple types are the concrete representations of the datatypes defined by ebXML Core Components and specialized dataTypes defined by GJXDM referred to as proxied GJXDM simple types j-xsd:, rather than using the xsd:schema simple types directly. Extensions to simple types must use as their base the set of provided simple types defined in GJXDM.

The GJXDM Schema module incorporates XML Schema built-in types and fundamental CCTS types. The GJXDM Schema module declares the built-in types to be used.

[GXS6]	Any user defined types with simple content MUST be derived via extension or restriction from the GJXDM proxies defined in (namespace) for xsd:simpleTypes.

6.3.2.2 XSD:appinfo

The `xsd:appinfo` feature is used by schema to convey processing instructions to a processing application, Stylesheet, or other tool. Some users have determined that this technique poses a security risk and have employed techniques for stripping

`xsd:appinfo` from schemas. However, as GJXDM MNDR is committed to ensuring the widest possible target audience, this feature may be used to convey non-normative information. Non-normative information means non-standard.

[GXS15]	GJXDM designed schema MAY use xsd:appinfo. If used, xsd:appinfo MUST only be used to convey non-normative information. Note: appinfo is a recent addition to GJXDM in Version 3.0.2 (Additional research is required about how this element is intended to be used in future versions by the XSTF. This is an action item for the XSTF members (Tom Carlson/Scott Edson/Ellen Perry).

6.3.2.3 Extension and Restriction

An existing GJXDM type can be modified to fit the requirements of customization through XSD derivation. These modifications can include extension (adding new information to an existing type) or refinement (restricting the set of information allowed to a subset of that permitted by the existing type) [need to add more information about when restriction is to be used].

[GXS16]	Complex type extension and restriction MAY be used.

6.3.3 Namespaces and Schema Locations

NMS 1-2, NMS 4-5

Reviewers: Provide Comments to NMS1-NMS18 where defined in Appendix A

Guidance/Feedback on these rules in Appendix A need to be reviewed and vetted before developing narrative and moving examples into this section.

6.3.4 External Code List Rules

Reviewers: Provide Comments to CDL1-CDL8 where defined in Appendix A

CDL1-* Review approach to external schema and external code lists before drafting this section

6.3.5 General Type Definitions

GTD1-GTD2

Since GJXDM document and extension schema elements and types are intended to be reusable, all types must be named. This permits other types to establish elements that reference these types, and also supports the use of extensions for the purposes of versioning and customization.

[GTD1]	All types MUST be named.

Example:

```
<xsd:complexType name="PersonType">
...
</xsd:complexType>
```

GJXDM information exchange schema disallows the use of `xsd:anyType`, because this feature permits the introduction of potentially unknown types into an XML instance. GJXDM intends that all constructs within the instance be described by the schemas describing that instance - `xsd:anyType` is seen as working counter to the requirements of interoperability. In addition, use of xsd:any has been identified as a significant security risk. [cross ref to other xsd:any references]. In consequence, particular attention is given to the need to enable meaningful validation of the GJXDM document instances. Were it not for this, `xsd:anyType` might have been allowed.

[GTD2]	The xsd:anyType MUST NOT be used.

6.3.6 Complex Type Definitions

CTD1-CTD3, CTN1

Since even simple datatypes are modeled as property sets in most cases, the XML expression of these models primarily employs `xsd:complexType`. To facilitate reuse,

versioning, and customization, all complex types are named. In the GJXDM information exchange model, xsd:complexType(s) with complex content are considered classes(objects) .

[CTD1]	For every class identified in GJXDM extension and document schema, a named xsd:complexType MUST be defined.

Example:

```
<xsd:complexType name="BuildingType">
….
</xsd:complexType>
```

GJXDM classes(objects) are defined in schema as named complexTypes. The sequence of elements contained within the complex type represent the properties of the class(object). These

property elements are defined as global elements where each global element.references a corresponding complex or simple type. xsd:Sequence facilitates the use of `xsd:extension` for versioning and customization.

[CTD2]	Every GJXDM user-defined xsd:complexType definition content model MUST use the xsd:sequence element with appropriate global element references to reflect each property of its class. This does not preclude the use of xsd:choice (see *rule name*)

Example:

```
<xsd:complexType name="AddressType">

...

<xsd:sequence>

<xsd:element ref="j:LocationCityName" minOccurs="0"
maxOccurs="unbounded">

...

</xsd:element>

<xsd:element ref="j:LocationPostalCodeIDZone" minOccurs="0"
maxOccurs="unbounded">

...
</xsd:element>
...

</xsd:sequence>

</xsd:complexType>
```

There is a direct one-to-one relationship between `ebXML CoreComponentTypes` and `GJXDM PrimaryRepresentationTerms`. Additionally, there are several GJXDM SecondaryRepresentationTerms that are subsets of their parent GJXDM PrimaryRepresentationTerm. The total set of ISO 11179 Representation Terms by their nature represent GJXDM Datatypes. Specifically, for each GJXDM PrimaryRepresentationTerm or GJXDM SecondaryRepresentationTerm, an ebXML UnspecializedDatatype exists. In the GJXDM , these ebXML UnspecializedDatatypes are expressed as complex or simple types that correspond to an ebXML CoreComponentType.

`The set of valid GJXDM datatypes are based on ebXML Core Component Technical Specification v1.9 and include the following:`

1) `Amount`
2) `BinaryObject (secondary: Graphic,Picture,Sound,Video)`

3) Code
4) *DateTime (secondary: Date, Time)
5) Identifier (authorized abbreviation: ID)
6) Indicator
7) Measure
8) Numeric (secondary: Value, Rate, Percent)
9) Quantity
10) Text (secondary: Name)

Reference:GTRI May 2004 Developer Workshop

*note: DateTime element not in GJXDM but secondary Date and Time elements are included

[CTD3]	For every user-defined datatype based on the valid set of GJXDM datatypes, a named xsd:complexType or xsd:simpleType MUST be defined.

6.3.7 Complex Type Naming Rules

GJXDM identifies naming rules for types, namely for complex types based on Primary Representation Terms, Secondary Representation Terms and the ebXML Core Component Types. Each of these complex and simple types are a fully qualified type name based on ISO 11179. As such, these names convey explicit semantic clarity with respect to the data being described. Accordingly, these naming standards ensure that GJXDM xsd:complexType names are semantically unambiguous, and that there are no duplications of GJXDM type names for different xsd:type constructs.

GJXDM xsd:complexType names follow general naming rules, and append the suffix "Type" to denote a Type Name versus an Element Name.

[CTN1]	A user-defined xsd:complexType ***name*** MUST name the object suffixed by the word "Type". For example <xsd:complexType name="PersonType"> is correct And <xsd:complexType name="Person"> would be incorrect.

6.3.8 Attribute Declarations

ATD1-*

Following is UBL reasons for discouraging User-defined attributes:

Attributes are W3C Schema constructs associated with elements that provide further information regarding elements. While elements can be thought of as containing data, attributes can be thought of as containing metadata. Unlike elements, attributes cannot be nested within each other—there are no “subattributes.” Therefore, attributes cannot be extended as elements can. Attribute order is not enforced by XML processors—that is, if the attribute order in an XML instance document is different than the order in which the attributes are declared in the schema to which the XML instance document conforms, no error will result. These limitations dictate that GJXDM MNDR restrict the use of attributes to XSD built-in attributes, or to the GJXDM SuperTypeMetadataAttributeGroup defined by GJXDM.

[It would be helpful to include additional guidance regarding understanding when to define information as metadata instead of data; this has been written up in the new User’s guide – Paul Embley will send the reference to John Ruegg to be included here.]

[ATD1]	User defined attributes SHOULD NOT be used.

6.3.8.1 Global Attributes

The current GJXDM has attributes that are common to all GJXDM and proxy Code List elements. These common attributes have been declared using the `xsd:globalattributegroup` element and utilizes the following rule. These rules are included to ensure interoperability.

[ATD2]	If a Schema Expression contains one or more common attributes that apply to all elements contained or included or imported therein, the common attributes SHOULD be declared as part of a global attribute group. (For an example about how to do this, see the Global JXDM global attribute group named ”**SuperTypeMetadata”)**

[Ensure that the MNDR has a rule: Extended types MUST be derived from SuperType]

[ATD3]	For each Global JXDM user-defined element of simpleType and a xsd:restriction element; an xsd:base attribute MUST be declared and set to the appropriate GJXDM datatype.

6.3.8.2 Schema Location

GJXDM is rapidly becoming a national and potentially international standard that will be used in perpetuity by justice agencies around the globe. It is important that these users have unfettered access to all GJXDM conformant schema.

[ATD4]	Each xsd:schemaLocation attribute declaration MUST contain a system-resolvable URI, referencing the location of the schema or schema module.

5/13/05: These questions are being addressed by Scott Came in his adjustments to the NMS rules.

6.3.8.3 XSD:nil

You can indicate in a schema that an element may be nil in the instance document. Empty content vs nil:

- Empty: an element with an empty content is constrained to have no content.
- nil: an instance document element may indicate no value is available by setting an attribute - xsi:nil - equal to 'true'

Example:

```
XML Schema:
<xsd:complexType name="PersonNameType">
    <xsd:complexContent>
      <xsd:extension base="j:SuperType">
        <xsd:sequence>
…
          <xsd:element ref="j:PersonGivenName" minOccurs="0"
maxOccurs="unbounded"/>
…
          <xsd:element ref="j:PersonMiddleName" minOccurs="0"
maxOccurs="unbounded"/>
…
          <xsd:element ref="j:PersonSurName" minOccurs="0"
maxOccurs="unbounded"/>
…
        </xsd:sequence>
…
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

<xsd:element name="PersonName" type="j:PersonNameType" nillable="true"/>

<xsd:element name="PersonGivenName" type="j:PersonNameTextType"
nillable="true"/>

<xsd:element name="PersonMiddleName" type="j:PersonNameTextType"
nillable="true"/>

<xsd:element name="PersonSurName" type="j:PersonNameTextType" nillable="true"/>


XML instance document:
```

```
<PersonName>
   <PersonGivenName>Hannibal</PersonGivenName>
   <PersonMiddleName xsi:nil=”true”/>
   <PersonSurName>Lecter</PersonSurName>
</PersonName>
```

[ATD5]	For local extensions based on GJXDM version 3.0.2 and above, the xsd built-in nillable attribute MUST be used for any Global JXDM user-defined element which has simpleContent ; Note: An example from GJXDM v3.0.2 <xsd:element name="**WitnessLocationDescriptionText**" type="**j:TextType**" nillable="**true**">

6.3.8.4 XSD:anyAttribute

GJXDM information exchange schema disallows the use of `xsd:anyAttribute`, because this feature permits the introduction of potentially unknown attributes into an XML instance. GJXDM information exchange packages intend that all constructs within the instance be described by the schemas describing that instance - `xsd:anyAttribute` is seen as working counter to the requirements of interoperability. (reference security issues) In consequence, particular attention is given to the need to enable meaningful validation

of the GJXDM conformant document instances.

[ATD6]	The xsd:any attribute MUST NOT be used.

6.3.9 Element Declarations and Naming Rules

ELD1-ELD8

6.3.9.1 Elements Bound to Complex Types

W3C XSD allows for any globally declared element to be the document root element. To keep consistency in the instance documents and to adhere to the underlying process model that supports each GJXDM information exchange Schema, it is desirable to have one and only one element function as the root element. Since GJXDM follows a global element declaration scheme (See Rule ELD2), each GJXDM Schema will identify one element declaration in each schema as the document root element. This will be accomplished through an `xsd:annotation` child element for that element in accordance with the following rule:

[ELD1]	Each GJXDM `DocumentSchema` MUST identify one and only one global element declaration that defines the exchange document being conveyed in the Schema

	expression. That global element MUST include an xsd:annotation child element which MUST further contain an xsd:documentation child element that declares "*This element MUST be conveyed as the root element in any instance document based on this* `DocumentSchema`."

[Definition] Document schema:

The overarching schema within a specific namespace that conveys the business document functionality of that namespace. The document schema declares a target namespace and is likely to pull in additional schema by importing external schema modules. Each namespace will have one, and only one, document schema.

ADD A RULE PROHIBITING THE USE OF <xsd:include> in locally defined extensions

ADD A RULE PROHIBITING THE DECLARATION OF NEW TYPES WITHIN A DOCUMENT SCHEMA..

Example:

```
<xsd:element name="Rapsheet" type="RapsheetType">
<xsd:annotation>
<xsd:documentation>This element MUST be conveyed as the root element in
any instance
document based on this Schema expression</xsd:documentation>
</xsd:annotation>
</xsd:element>
```

Global elements are declared as direct children of a root schema element. Global element names, because they are globally reusable, express universally unambiguous semantics in their names. By their nature, global elements can be reused consistently across the Justice & Public Safety enterprise XML domain, wherein every occurrence will have exactly the same meaning and map to exactly the same authoritative source data. Therefore, the GJXDM information exchange schema approach uses global elements in conjunction with global complex types.

Eliminating potential barriers to Justice XML system interoperability is a key driver for choosing the global element approach. The advantages of this approach become more evident when considering inter-agency XML interoperability and data exchange. Because the recommended approach relies on global elements to carry unique semantics, there cannot be a duplicate occurrence of elements with different characteristics in the GJXDM enterprise namespace. This reduces the level of effort to analyze, map, and transform elements that are unique to a particular functional area.

[ELD2]	All element declarations MUST be global.

6.3.9.2 Element Names for complexType(s)

For each class (complexType) defined in an extension, subset or document schema, a global element name MUST be declared with the same name as the name of the corresponding `xsd:complexType` to which it is bound, with the word "`Type`" removed.

Example:

The xsd:complexType named TelephoneNumberType would require a corresponding global element be declared. The the name of this corresponding global element must be TelephoneNumber.

```
<xsd:element name="TelephoneNumber" type="j:TelephoneNumberType"
nillable="true">

<xsd:complexType name="TelephoneNumberType">
…
</xsd:complexType>
```

[ELD3]	For every class defined as a GJXDM user-defined types, a global element bound to the corresponding xsd:complexType MUST be declared. For example, a schema defining a complexType named my:FavoritePersonType would need to declare a global element named "FavoritePerson" of objectType = my:FavoritePersonType to bind a global element name to the complexType.

6.3.9.3 Elements Representing Relationships Between Classes

A "`relationship" element` is not a class. Rather, it is an association between two classes. As such, the element declaration will bind the element to the xsd:complexType of the associated object-`class`. There are two basic methods for representing "relationship" elements – those that have qualifiers in the object-class, and those that do not.

[Need to add explicit definitions for "qualified" and "unqualified" relationships to the definitions section; one suggestion is to include the cross reference to the equivalent UBL/ISO 15000.component to help clarify the definition.]

When a "relationship" element is unqualified, it is bound via reference to the global object-class element to which it is associated using GJXDM ReferenceType elements or RelationshipType

element. When a “relationship” element is qualified, a new element MUST be declared and bound to the `xsd:complexType` of its associated object-class. This is sometime refered to as the “element inclusion” method of relating two object-classes. The latter method can be accomplished by extending the object-class parent type to include the “relationship” element. For example, PersonVehicle is a qualified “relationship” element to relate Persons to Vehicles. The “PersonVehicle” element would be added to the “PersonType” object-class via extension and the object-type “Vehicle” would then be related to “Person” where the cardinality of the relationship would be represented with the minOccurs and maxOccurs attributes.

Example of Unqualified “relationship”

```
SCHEMA
<xsd:complexType name="ChargeType">
  <xsd:complexContent>
    <xsd:extension base="j:SuperType">
      <xsd:sequence>
…
        <xsd:element ref="ext:ChargeArrestReference" minOccurs="0"
maxOccurs="unbounded"/>
…
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="CaseCharge" type="ext:ChargeType"/>
<xsd:element name="ChargeArrestReference" type="j:ReferenceType"/>

INSTANCE DOCUMENT
<j:Arrest j:id="arrest01">
…
</j:Arrest>

<ext:CaseCharge>
…
  <ext:ChargeArrestReference j:ref="arrest01"/>
…
</ext:CaseCharge>


Example of a Relationship-type element in an instance document:

<j:Arrest j:id="arrest03">
…
</j:Arrest>

<ext:CaseCharge j:id=”charge05”>
…
</ext:CaseCharge>


<j:Relationship name=”ChargeArrest” subject=”charge05”
object=”arrest03” />
```

Example of Qualified “relationship”

```
snippet of an extension to a complexType to relate two entities

  <xsd:complexType name='DrivingIncidentType'>
    <xsd:annotation>
      <xsd:documentation>A structure that describes details of an
incident involving a vehicle.</xsd:documentation>
    </xsd:annotation>
    <xsd:complexContent>
      <xsd:extension base='j:DrivingIncidentType'>
        <xsd:sequence>
          <xsd:element ref='DrivingIncidentInvolvedVehicle'
minOccurs='0' maxOccurs='unbounded' />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name='DrivingIncidentInvolvedVehicle' type='VehicleType'
>
    <xsd:annotation>
      <xsd:documentation>A vehicle that was involved in a driving
incident, whether damaged or not.</xsd:documentation>
    </xsd:annotation>
  </xsd:element>
```

[ELD4]	When a “relationship” element is unqualified, it is bound via reference to the global object-class element to which it is associated. When a “relationship” element is qualified, a new element MUST be declared and bound to the xsd:complexType of its associated object-class.

6.3.9.4 ComplexTypes with SimpleContent

Elements bound to ebXML core-component types and GJXDM j-xsd: proxy built-in xsd dataTypes.

[ELD5]	For every user-defined simpleType, an xsd:restriction element MUST be declared

6.3.9.5 Empty Elements

Empty elements may cause application or XML processing difficulty and must be avoided

[ELD6]	Empty elements MUST NOT be declared. (need to wordsmith this some) Extension schemas MUST NOT declare elements of type SuperType. Extension schemas MUST NOT declare complex types that extend SuperType without adding additional elements.

6.3.9.6 Global Elements with simpleContent

The GJXDM Global elements with simpleContent are reused in multiple contexts. Their reuse in a specific context is typically identified in part through the use of qualifiers. However, these qualifiers do not change the nature of the underlying concept of the GJXDM core component that the element is derived from. As such, qualified global elements with simpleContent are always bound to the same type as that of their unqualified corresponding xsd: built-in datatype, j-xsd or GJXDM (ebXML core component) datatype.

Example:

```
<xsd:element name="StreetNumberText" type="j:TextType"/>
```

[ELD7]	Global simpleType elements declared with Qualified Properties must be of the same type as their corresponding Unqualified Property.

6.3.9.7 XSD:Any Element

GJXDM MNDR disallows the use of `xsd:any`, because this feature permits the introduction of potentially unknown elements into an XML instance. GJXDM MNDR intends that all constructs within the instance be described by the schemas describing that instance - `xsd:any` is seen as working counter to the requirements of interoperability. In consequence, particular attention is given to the need to enable meaningful validation of the document instances. Were it not for this, `xsd:any` might have been allowed. (include security concerns)

[ELD8]	The `xsd:any` element MUST NOT be used.

6.3.10 Schema Documentation and Annotations

DOC1,NMS18

6.3.10.1 Embedded documentation

The information about each GJXDM user-defined “Type” or “Element” must be documented in schema. Fully annotated Schemas are valuable tools to implementers to assist in understanding the nuances of the information contained therein. GJXDM information exchange schema annotations will consist of a set of metadata elements.

The absence of an optional annotation inside the structured set of annotations in the documentation element implies the use of the default value. For example, there are several annotations relating to context such as `BusinessContext` or `IndustryContext` whose absence implies that their value is "`all contexts`".

The following general documentation rule describes the documentation requirements for GJXDM user-defined “Types” or “Elements”.

[DOC1]	The xsd:documentation element for every GJXDM user-defined Element MUST contain a structured set of annotations in the following sequence and pattern: • Version (optional): An indication of the evolution over time of the Datatype. • Definition(mandatory): The semantic meaning of an Element • Cardinality(mandatory): Indication whether the complexType Element (Property) represents a not-applicable, optional, mandatory and/or repetitive characteristic of the parent complexType • AssociatedObjectClassQualifier (optional): Associated Object Class Qualifiers describe the 'context' of the relationship with another complexType object. That is, it is the role the contained Element plays within its association with the containing complexType object. • AssociatedObjectClass (mandatory); Associated Object Class is the Object Class at the other end of this association. It represents the Aggregate Business Information Entity contained by the Association Business Information Entity. (For example: the element PersonName within the complexType PersonType has Name as the AssociatedObjectClass contained in the Aggregate Business Information Entity called PersonType.) • AlternativeBusinessTerms (optional): Any synonym terms under which the Element is commonly known and used in the business. • Examples (optional): Examples of possible values for the Element Context Element should go here as well if not defined in a later NDR, so this is a

	placeholder for now. The list of Context Drivers is in the UBL ccts: core-component parameters schema. The current values for Context drivers are: 1 Business Process 2 Product Classification 3 Geopolitical Region 4 Official Constraint 5 Business Process Role 6 Supporting Role 7 System Capabilities

875

876 Example Extension Element Documentation:

877

878

```
879 <xsd:annotation>
880   <xsd:documentation>
881     <Component>
882       <ElementName>ChargeArrestReference</ElementName>
883 <Version>1.0</Version>
884       <Definition> A reference to the Arrest which resulted in the
885 filing of a charge..</Definition>
886       <Cardinality>0..1</Cardinality>
887       <ObjectClass>Charge</ObjectClass>
888       <PropertyTerm>Arrest</PropertyTerm>
889       <RepresentationTerm>Reference</RepresentationTerm>
890       <AssociatedObjectClass>Reference</AssociatedObjectClass>
891     </Component>
892   </xsd:documentation>
893 </xsd:annotation
```

894

895

896

[NMS18]	Also, if adopted, the UBL core-components parameters schema (place where schema annotation/documentation elements are defined including context drivers) could be added back into this mndr. But for now, holding off on the UBL metadata per subcommittee member requests. Could be included within extension schema as superattributegroup maybe called component metadata?? Namespace could be referenced directly from document schema or via extension schema.

897

898 6.3.11 Schema Version Numbering Rules

899 VER1-VER11 (I need to add schema filenames to this referencing the namespace standards or
900 moving the filenames to this spot)

901 GJXDM user-defined namespaces are suffixed with a `document-id and version`. The
902 GJXDM MNDR has decided to include versioning information as part of the document-id
903 component of the namespace. The version information is divided into `major` and `minor` fields.
904 The `minor` field has an optional `revision` extension. For example, the namespace URI for a
905 Charging Document schema has this form:

```
http://myNS.com/<schema Type>/<document-name>/<document version>

note: where <schema Type> denotes  document or extension
```

The major-version field is "1" for the first release of a namespace. Subsequent major releases increment the value by 1.

For example, the first namespace URI for the first major release of the Charging document has the form:

```
http://myNS.com/document/Complaint/1.0
```

The second major release will have a URI of the form:

```
http://myNS.com/document/Complaint/2.0
```

The distinguished value "0" (zero) is used in the minor-version position when defining a new major version. In general, the namespace URI for every major release of the Complaint domain has the form:

Example:

```
http://myNS.com/<schema type>/<name>/<major-number>.0[.<revision>]
```

[VER1]	Every GJXDM information exchange schema and schema module major version ***draft*** MUST have a version number of the form: <major>.0[.<revision>]

When a major version reaches Standard status the [.<revision>] must not be present.

[VER2]	Every GJXDM Information exchange Schema and schema module major version ***Standard*** MUST have a version number of the form: <major>.0

For each document produced by the TC, the TC will determine the value of the <document name> variable. In GJXDM MNDR, the major-version field of a namespace URI must be changed in a release that breaks compatibility with the previous release of that namespace. If a change does not break compatibility then only the minor version need change. Subsequent minor releases begin with minor-version 1.

940
941 Example
942
943 The namespace URI for the first minor release of the Complaint document has this form:
944
945 `http://myNS.com/document/Complaint/1.1`
946

[VER3]	Every minor version release of a GJXDM Information exchange schema or schema module ***draft*** MUST have a version number of the form: <major >.<non-zero>[.<revision>]

947
948 When a minor version reaches Standard status the [.<revision>] must not be present.
949

[VER4]	Every minor version release of a GJXDM information exchange schema or schema module ***Standard*** MUST have a version number of the form: <major >.<non-zero>

950
951 Once a schema version is assigned a namespace, that schema version and that namespace
952 will be associated in perpetuity. Any change to any schema module mandates association
953 with a new namespace**.**
954

[VER5]	For GJXDM information exchange schema minor version changes, the <document name> MUST NOT change.

955
956
957
958 If a GJXDM schema namespace URI changes then any schema that imports the new version of
959 the namespace must also change (to update the namespace declaration). And since the
960 importing schema changes, its namespace URI in turn must change. The outcome is twofold:
961
962 • There should never be ambiguity at the point of reference in a namespace
963 declaration or version identification. A dependent schema imports precisely
964 the version of the namespace that is needed. The dependent schema never
965 needs to account for the possibility that the imported namespace can change.
966
967 • When a dependent schema is upgraded to import a new version of a schema,
968 the dependent schema's version (in its namespace URI) must change.
969
970 Version numbers are based on a logical progression. All major and minor version

numbers will be based on positive integers. Version numbers always increment positively by one.

[VER6]	For every GJXDM information exchange schema and schema module, the major version number MUST be a sequentially assigned, incremental number greater than zero.

[VER7]	For every GJXDM information exchange schema and schema module, the minor version number MUST be a sequentially assigned, incremental non-negative integer.

In keeping with rules NMS1 and NMS2, each schema minor version will be assigned a separate namespace. A minor revision (of a namespace) imports the schema module for the previous version.

For instance, the document schema defining:

```
http://myNS.com/document/Complaint/1.2
```

will import the namespace:

```
http://myNS.com/document/Complaint/1.1
```

The `version 1.2` revision may define new complex types by extending or restricting `version 1.1` types. It may define brand new complex types and elements. It must not use the XSD redefine element to change the definition of a type or element in the `1.1` version.

The opportunity exists in the `version 1.2` revision to rename derived types. For instance if `version 1.1` defines `Address` and `version 1.2` specializes `Address` it would be possible to give the derived `Address` a new name, e.g. `NewAddress`. This is not required since namespace qualification suffices to distinguish the two distinct types.

The minor revision may give a derived type a new name only if the semantics of the two types are distinct.

For a particular namespace, the minor versions of a major version form a linearly-linked family. The first minor version imports its parent major version. Each successive minor

version imports the schema module of the preceding minor version.

Example

```
http://myNS.com/document/Complaint/1.2
```

imports

```
http://myNS.com/document/Complaint/1.1
```

which imports

```
http://myNS.com/document/Complaint/1.0
```

Will delete VER8 and VER9 rules.

[VER8]	A GJXDM information exchange minor version document schema MUST import its immediately preceding version document schema.

To ensure that backwards compatibility through polymorphic processing of minor versions within a major version always occurs, minor versions must be limited to certain allowed changes. This guarantee of backward compatibility is built into the `xsd:extension` mechanism. Thus, backward incompatible version changes can not be expressed using this mechanism.

[VER9]	GJXDM information exchange schema and schema module minor version changes MUST be limited to the use of xsd:extension or xsd:restriction to alter existing types or to add new constructs.

Ensuring semantic compatibility across minor versions is essential. Semantic compatibility in this sense pertains to preserving the business function.

[VER10]	GJXDM information exchange schema and schema module minor version changes MUST not break semantic compatibility with prior versions. Minor versions maintain forward compatibility. **(have added additional language in Appendix A)**

Major versions of schema do NOT necessarily have to maintain forward compatibility with the previous major version.

[VER11]	GJXDM information exchange schema and schema module major version changes MAY break semantic and/or structural compatibility with prior versions. No backward

	compatibility is guaranteed.

1036

1037 6.3.12 Import versus Include

1038 GXS14, SSM1-SSM4

1039

1040 6.3.12.1 Schema Modularity

1041

1042 GJXDM MNDR supports modularity in schema design. The full GJXDM schema may be
1043 modularized by creating multiple subset schemas from GJXDM.

1044

[SSM1]	GJXDM Schema MAY be split into a smaller subset schema, but only one GJXDM subset can be created for a given document schema, because the GJXDM schema must reside in one and only one namespace. In other words, an IEP can not use multiple versions of a GJXDM subset schema.

1045

1046

1047 GJXDM based document schemas will be developed over time, each of which expresses a
1048 separate business function for transaction data or a business document. The GJXDM MNDR
1049 schema modularity approach is structured so that users can reuse individual document schemas
1050 as is or with modification for local usage.

1051

1052 Additionally, a document schema can import individual schema modules without having to import
1053 the entire GJXDM reference schema module. Each document schema will define its own
1054 dependencies. The GJXDM MNDR schema modularity model ensures that logical associations
1055 exist between document and that individual modules can be reused to the maximum extent
1056 possible.

1057

1058 This is accomplished through the use of document schema, internal schema modules and
1059 external schema modules as shown in Figure 1-1.

1060

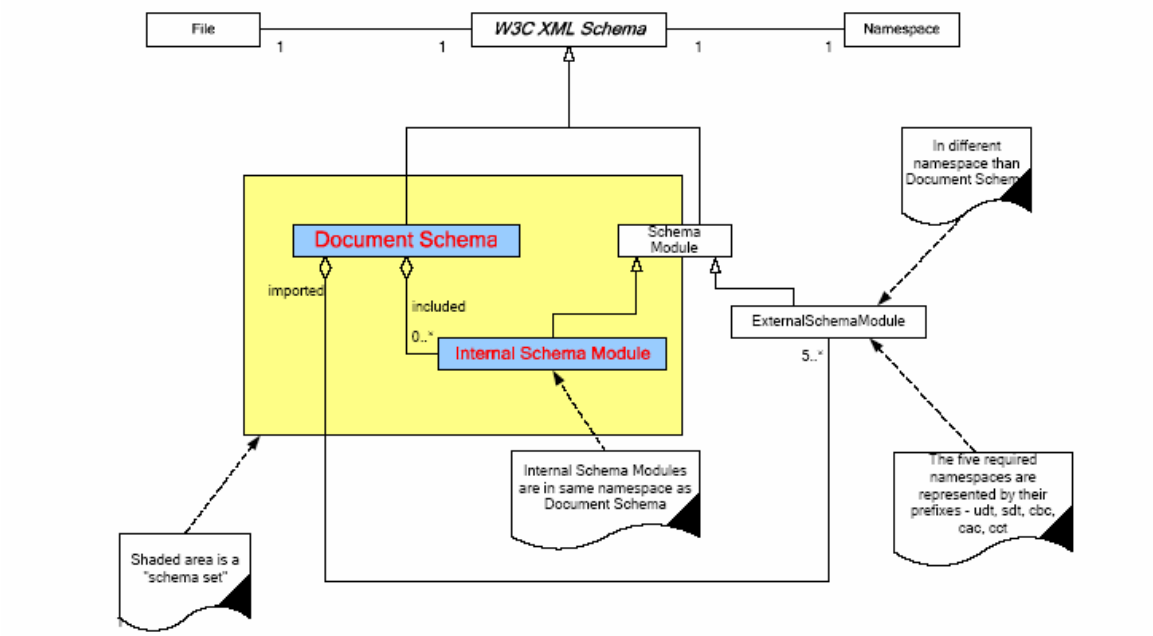


Figure 1-1 shows the one-to-one correspondence between document schemas and namespaces. It also shows the one-to-one correspondence between files and schema modules. As shown in figure 1-1, there are two types of schema in the GJXDM MNDR – document schema and schema modules (subset, extension, CodeList(s)…) Document schemas are always in their own namespace. Schema modules may be in a document schema namespace as in the case of internal schema modules, or in a separate namespace as in the GJXDM proxy CodeList(s), GJXDM subset(s) and optionally the GJXDM extension schema. Both types of schema modules are conformant with W3C XSD.

A namespace is an indivisible grouping of types. A “piece” of a namespace can never be used without all its pieces. For larger namespaces, schema modules – internal schema modules – may be defined. GJXDM document schemas may have zero or more internal modules that they include. The document schema for a namespace then includes those internal modules.

[Definition] Internal schema module –

A schema that is part of a schema set within a specific namespace.

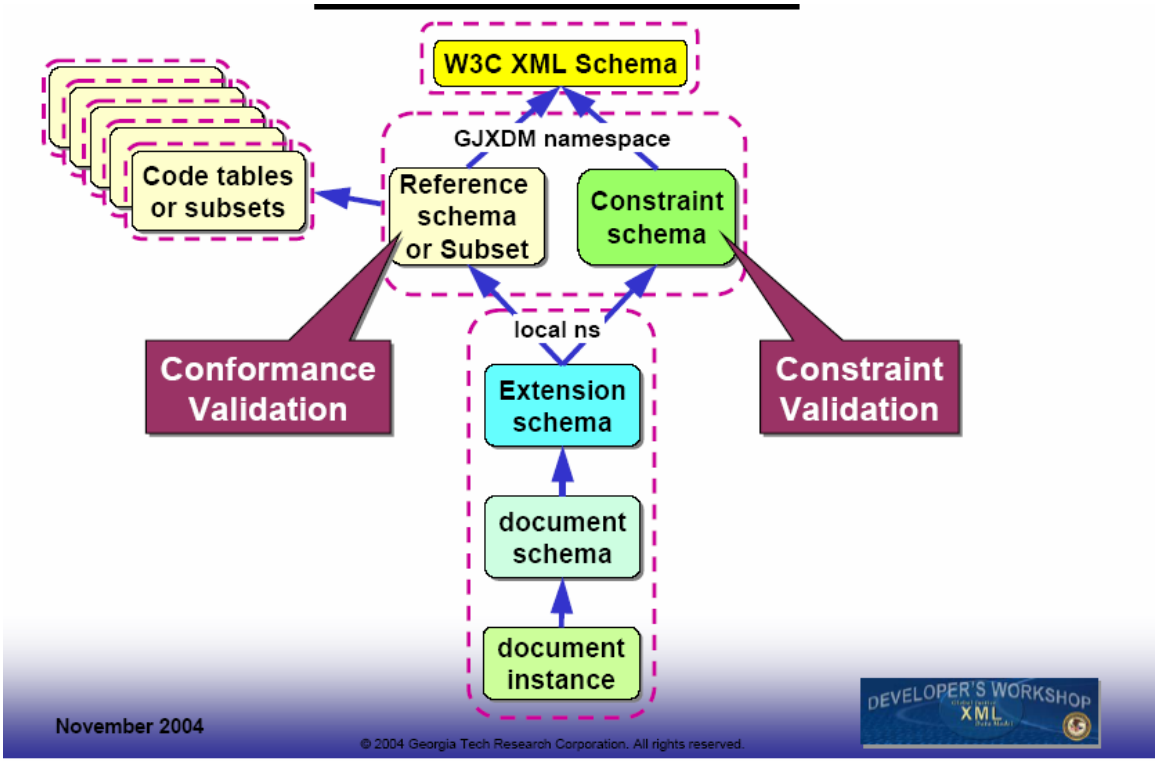


Figure 1.2 shows how the GJXDM MNDR namespace standard for segregating GJXDM schema from developer schema .

Any schema module, be it a document schema, may import other schemas from other namespaces.

6.3.12.2 Limitations on Import

If two namespaces are mutually dependent then clearly, importing one will cause the other to be imported as well. For this reason there must not exist circular dependencies between schema modules. By extension, there must not exist circular dependencies between namespaces. A namespace “A” dependent upon type definitions or element declarations defined in another namespace “B” must import “B’s” document schema.

[SSM2]	A document,or extension , schema in one namespace that is dependent upon type definitions or element declarations defined in another namespace MUST only import the document schema from that namespace.

To ensure there is no ambiguity in understanding this rule, an additional rule is necessary to address potentially circular dependencies as well – schema A must not import internal schema modules of schema B.

[SSM3]	A document schema in one namespace that is dependant upon type definitions or element declarations defined in another namespace MUST NOT import internal schema modules from that namespace.

6.3.12.3 Module Conformance

GJXDM has defined a set of naming and design rules to ensure maximum interoperability and standardization.

[SSM4]	Imported schema modules MUST be fully conformant with GJXDM information exchange schema naming and design rules.

6.3.12.4 Internal and External Schema Modules

Developers will create schema modules which, as illustrated in Figure 1.1 and Figure 1.2, will either be located in the same namespace as the corresponding document schema, or in a separate namespace.

[SSM5]	GJXDM schema modules MUST either be treated as external schema modules or as internal schema modules of the document schema.

6.3.12.5 Internal Schema Modules

The MNDR does not support the use of Include statements in schema. As a result, this section is not required. (Should consider including a rubric to define a “not supported feature” indicator to clearly identify those sections that are excluded on purpose. In other words, better to explicitly exclude sections instead of leaving them out entirely)

Internal schema modules do not declare a target namespace, but instead reside in the namespace of their parent schema. All internal schema modules will be accessed using xsd:include.

[SSM6]	All internal schema modules MUST be in the same namespace as their corresponding document schema.

Internal schema modules will necessarily have semantically meaningful names. Internal schema module names will identify the parent schema module, the internal schema module function, and the schema module itself.

[SSMx] Each internal schema module MUST be named {ParentSchemaModuleName}{InternalSchemaModuleFunction}{schema module}

6.3.12.6 External Schema Modules

GJXDM is dedicated to maximizing reuse. As the complex types and global element declarations will be reused in multiple schemas, a logical modularity approach is to create GJXDM schema modules based on collections of reusable types and elements.

[SSM7]	GJXDM schema module(s) MAY be created for reusable components.

As identified in rule SSM2, developers will create external schema modules. These external schema modules will be based on logical groupings of contents. The set of possible schema modules includes:

- GJXDM Reference Schema
- GJXDM Subset Schema(s)
- GJXDM Extension Schema(s)
- GJXDM proxy Code List(s)
- Non-GJXDM External Schema(s) and Code List(s)
- If adopted, CCTS Core Component Parameters (for documentation/annotation of new schema elements)

6.3.13 Controlling refinement (final)

GXS8

XSD:final is utilized to stop further restriction or extension on complexTypes

[GXS8]	The xsd:final attribute MUST be used when schema developers want to prevent restriction or extension of a user-defined simpleTypes or complexType extensions.

6.3.14 Controlling Type and Element Substitution (block)

GXS9

What schema methods are available to prevent XML/Schema developers from extending, restricting and/or using type or element substitution on Reference Schema component(s) or element(s)?

Schema provides the xsd:attribute named “block” which can be used to block extensions/restrictions and/or substitutions to Components (ComplexTypes & SimpleTypes) and Elements. Following is the block attribute syntax which would be used in Schema at the element or component level as deemed appropriate:

block = “#all”	No extension, restriction or substitution allowed
block = “restriction”	No restriction allowed
block = “extension”	No extension allowed
block = “substitution”	No substitution allowed
block = “restriction,extension”	No restriction or extention allowed (but substitution okay)
block = “restriction,substitution”	No restriction or substitution (but extension okay)
block = “extension,substitution”	No extension or substitution (but restriction okay)

Attributes for preventing Type extension/restriction:

Final=”#all”	no extension or restriction of Type allowed
Final=”restriction”	no restriction of Type allowed
Final=”extension”	no extension of Type allowed

<xsd:complexType name="Publication" final="#all" …> Publication cannot be extended nor restricted

<xsd:complexType name="Publication" final="restriction" …> Publication cannot be restricted

<xsd:complexType name="Publication" final="extension" …> Publication cannot be extended

[GXS9]	The xsd:block attribute MUST be used when schema developers want to prevent use of “type substitution” and “element substitution” in XML instance documents.

6.3.15 Character encoding

IND3

[IND3]	

	In conformance with ISO/IETF/ITU/UNCEFACT Memorandum of Understanding Management Group (MOUMG) Resolution 01/08 (MOU/MG01n83), all GJXDM XML SHOULD be expressed using UTF-8.

6.3.16 XSD:notation

XSD:notation is used to declare the format of non-XML data. A notation in XML is just like the notation declarations in DTDs. The main difference is that W3C XML Schema notations are namespace-aware and can be imported between schemas. When these declarations are used, the notations are used in xsd:enumeration facets to create simple types.

The notation datatype is used to declare links to external non-XML content (for example, image data) and then associate that content with an external application that handles it.

Notation is a built-in legacy simple type and are very seldom used in production applications, and not optimal for the Justice and Public Safety community.

[GXS10]	xsd:notations MUST NOT be used.

6.3.17 XSD:all

Used within a group, xsd:all has the same meaning as when it is used directly under xsd:complexType, except that there are no minOccurs and maxOccurs attributes and it cannot be marked as optional. The xsd:all compositor requires occurrence indicators of minOccurs=0 and maxOccurs=1. The xsd:all compositor allows for elements to occur in any order. The result is that in an instance document, elements can occur in any order, are always optional, and never occur more then once. Such restrictions are inconsistent with data-centric scenarios such as most of the work in the Justice & Public Safety community.

Another disadvantage of xsd:all is that it cannot be repeated any further. This limits the use of xsd:all to the first occurrence of its set of elements. If a content model requires an element that occurs more than once, then xsd:all cannot be used.

[GXS11]	The xsd:all element MUST NOT be used.

6.3.18 XSD:choice

The `xsd:choice` compositor allows for any element declared inside it to occur in the instance document, but only one. While `xsd:choice` is a very useful construct in situations where customization and extensibility are not a concern, GJXDM MNDR recommends against using `xsd:choice because it` cannot be extended. If extension is not a concern, then xsd:choice may be used.

[GXS13]	The xsd:choice element SHOULD NOT be used where customization and extensibility are a concern.

6.3.19 Packaging of Schemas (i.e., zip) and Naming rules

NMS17 (Get feedback from subcommittee before completing)

Reference minimum requirements of each folder (mike hulme paper). We could state a <DocumentSchemaName>IEP.zip folder must exist at the schemaLocation of where the Document Schema resides which contains the following subfolders: (and list minimum and likely contents of each subfolder):

<Schema> Folder

<Instance> Folder

<Documentation> Folder

6.4 Subset Schema naming and Design Rules

GJXDM subset schema(s) provide for limiting the full GJXDM set of object classes and set of element(s) within each class down to a subset which is relevant for a specific Document Schema or specific domain within a local jurisdiction.

6.4.1 Rules for Conformant Subset Schemas

Gtri training material ppt.

Any instance which validates against a schema subset must be able to validate against the full GJXDM reference schema.

Conformant Subset Schemas MUST NOT:

- Add local components
- Flatten type structures
- Modify namespaces
- Change object types
- Change element or type names
- Change type inheritance
- Make the subset inconsistent with the full reference schema

6.4.2 Subset Namespace and Filename Rules

NMS 3, NMS 7-9 (Get feedback from subcommittee before completing)

6.4.3 Design guidelines (e.g., use SSGT)

A “no cost” software tool for generating conformant GJXDM subset schema(s) may be found at:

http://gjxdmtools.gtri.gatech.edu/ssgt/subset

Other GJXDM schema generation tools may be used or developed by other parties, both open source and proprietary.

6.5 Constraint Schema Naming and Design Rules

6.5.1 Rules for Conformant Constraint Schemas

Gtri training stuff

GJXDM Constraint schemas embed localized constraints into GJXDM definitions.

The GJXDM namespace remains the same – just change the schema location attribute(s).

YOU CAN

1. Change object types
2. Change/drop type inheritance
3. Create differently constrained types based on a single GJXDM type
4. Make local component definitions
5. Add localized constraints
6. Force elements to appear or not appear

YOU CANNOT

1. Change element names
2. Change tag order or hierarchy
3. Define new components to be referenced outside of this schema
4. Leave out components required by instances

For example, a component is required if it:

•May appear in a valid instance.

•May be referred to by a schema outside GJXDM.

•Is required by another required component.

You may omit entities which are not required, including:

•xmlns – namespace prefix declarations

•xsd:import

•xsd:complexType and xsd:simpleType elements that are "top-level" (direct children of the xsd:schema element)

•xsd:element elements that are children of the xsd:schema element

•xsd:attribute elements that are children of the xsd:schema element

•xsd:element elements that are contained in a type definition

•xsd:attribute elements that are contained in a type definition

•xsd:attribute elements that appear in SuperType MetadataAttributeGroup.

•xsd:enumeration elements for enumerations that are not relevant to the applications or instances

•xsd:annotation or xsd:documentation elements

6.5.2 Constraint Namespace and Filename Rules

NMS14-16 (Complete after subcommittee vets these rules)

Reviewers: Provide Comments to NMS14-NMS16 where defined in Appendix A (6/28: this is pending Scott Came's document update).

6.6 Extension Schema Naming and Design Rules

NMS10-11 (Get feedback from subcommittee before completing)

Reviewers: Provide Comments to NMS10-NMS11 where defined in Appendix A (6/28: this is pending Scott Came's document update).

Defines common local extensions

6.6.1 Extension Patterns

Ruegg paper intro

6.6.1.1 Scenarios for mapping business data/documents to GJXDM

Question for TC – Is "Cascading Extension" what we now call "Concrete Typing"? If so, then we will update the term to be Concrete Typing instead of Cascading Extension. As a note, I believe UBL terminology is "Extension by Inclusion". 6/28: Have settled on CONCRETE TYPING as the term.

Scenario #1

Every property I need is in the GJXDM namespace where I need it

Scenario #2

I need additional "property" elements within a currently defined GJXDM Type

Scenario #3

I need to create new relationships between GJXDM types or elements by:

- Inclusion (defining a "relational" Element to extend GJXDM type)
- Reference (use GJXDM ReferenceType Elements or add ReferenceType Elements)
- GJXDM Relationship Element (explicit named referencing)

Scenario #4

I need a new type that doesn't inherit or extend any elements of GJXDM

Scenarios #2 - #3 represent methods for extending/restricting the GJXDM components elements and relationships to define the content of a specific document or data exchange set of elements.

6.6.1.2 Design Patterns for extensions and restrictions to GJXDM components

The following methods may be applied for modifying GJXDM components in creating GJXDM compliant Reference Documents and Data exchanges:

Method 1 : Extension using "Type Substitution"

Method 2: Extension using a "Cascading Extensions" construction to "Avoid Type Substitution"

Method 3: Extension using ReferenceType elements for Many-to-Many relationships

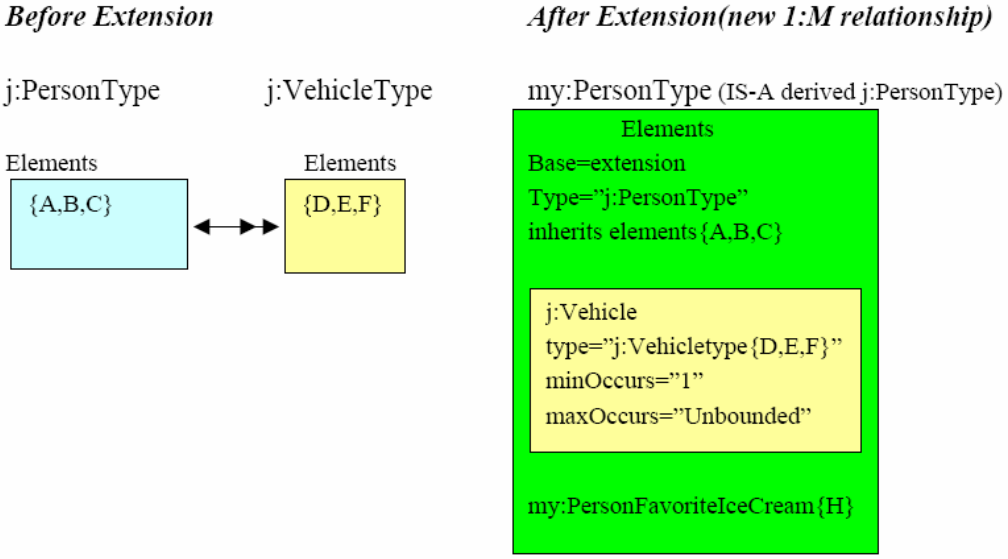
Method 4: Extensions using GJXDM Relationship Element for Many-to-Many relationships

Methods 1 and 2 address methods to modify GJXDM components and Methods 3 and 4 address methods for resolving many-to-many relationships.

Method 1 : Extension using "Type Substitution"

This method and the remaining methods will all assume we are extending PersonType(component) {A,B,C}with a 1:M or M:M "*relationship element*" to the VehicleType (component). Depending on the method we will use j:Vehicle {D,E,F}, j:VehicleReference {F}, my:PersonVehicle{G} and Relationship Qname="xxxx" {R1,R2}to illustrate the extension method. I will also add a "property element" as "my:PersonFavoriteIceCream" {H}

Type Substitution



This **Method-1** construction requires a type substitution declaration in the XML instance to dynamically invoke my:PersonType to replace j:PersonType at runtime. Following is an XML snippet of what is required:

```
<my:SampleDocument>
        <j:Person  xsi:type="my:PersonType"  >  ***type substitution***
                <j:PersonElements {A,B,C}>
                <j:Vehicle>
                        <j:Vehicle Elements {D,E,F}>
                </j:Vehicle>
                <my:PersonFavoriteIceCream/ {H}>
        </j:Person>
</my:SampleDocument>
```

1386

Method 2 : Extension using "Cascading Extensions" to "Avoid Type Substitution"

This method extends PersonType(component) {A,B,C}with a 1:M "*relationship element*" to the VehicleType (component) {D,E,F} using my:PersonVehicle{G}. I will also add a "property element" as "my:PersonFavoriteIceCream" {H}

Extension Using "Cascading Extensions"

Before Extension

j:PersonType

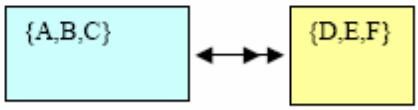
Elements

{A,B,C}

j:VehicleType

Elements

{D,E,F}



After Extension(new 1:M relationship)

my:PersonType (IS-A derived j:PersonType)

Elements

Base=extension

Type="j:PersonType"

inherits elements{A,B,C}

j:Vehicle

type="j:Vehicletype {D,E,F}"

minOccurs="1"

maxOccurs="Unbounded"

my:PersonFavoriteIceCream{H}

This **Method-2** construction eliminates the "type substitution" declaration in the method-1 XML instance and avoids requirement to dynamically invoke my:PersonType. Following is an XML snippet of what it looks like:

```
<my:SampleDocument>
        <my:Person  >  *** No type substitution required ***
                <j:PersonElements {A,B,C}>
                <j:Vehicle Elements {D,E,F}>
                <my:PersonFavoriteIceCream/ {H}>
        </my:Person>
</my:SampleDocument>
```

note: Cascade of **my:Person** under **my:SampleDocument** instead of dynamically substituting j:Person with my:PersonType as done in Method-1.

1387

Method 3 : Extension using ReferenceType elements for Many-to-Many relationships

This method extends PersonType(component) {A,B,C}with a M:M "*relationship element*" to the VehicleType (component) {D,E,F} using j:VehicleReference {F}. I will also add a "property element" as "my:PersonFavoriteIceCream" {H}. Both elements will be added using a **Method-2** approach.

Extension Using "Cascading Extensions" and Reference Elements

Before Extension

j:PersonType

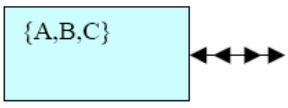
Elements

{A,B,C}

j:VehicleType

Elements

{D,E,F}



After Extension(new M:M relationship)

my:PersonType (IS-A derived j:PersonType)

Elements
Base="extension"
Type="j:PersonType"
inherits elements{A,B,C}

j: VehicleReference
type="j:ReferenceType{D,E,F}"
minOccurs="1"
maxOccurs="Unbounded"

my:PersonFavoriteIceCream{H}

my:VehicleType (IS-A derived j:VehicleType)

Elements
Base="extension"
Type="j:VehicleType"
inherits elements{D,E,F}

j: PersonReference
type="j:ReferenceType{A,B,C}"
minOccurs="1"
maxOccurs="Unbounded"

1388

This **Method-3** construction includes the j:VehicleReference element in my:PersonType and j:PersonReference in my:VehicleType for relating one or more persons to one or more vehicles by reference instead of the "Inclusion" method for adding "*relationship elements*" to a component. In the following XML example, this avoids duplicating Vehicle#1 elements for Person_A and Person_B. Instead, Person_A has a VehicleReference j:ref="xx" and Person_B has a VehicleReference j:ref="xx" and Vehicle#1 is only defined once in the XML instance. This would not be the case if we instead used "Inclusion" with the element j:Vehicle as shown in Method-2 above. Following is an XML snippet of what it looks like:

```
<my:SampleDocument>
        <my:Person j:id="yy" >  (Person_A)
                <j:PersonElements {A,B,C}>
                <j:VehicleReference j:ref="xx"/ > (pointer to Vehicle#1)
                <my:PersonFavoriteIceCream/ {H}>
        </my:Person>
        <my:Person j:id="zz" >  (Person_B)
                <j:PersonElements {A,B,C}>
                <j:VehicleReference j:ref="xx"/ > (pointer to Vehicle#1)
                <my:PersonFavoriteIceCream/ {H}>
        </my:Person>
        <my:Vehicle j:id="xx" > (Vehicle#1)
                <j:Vehicle elements{D,E,F}  >
        </my:Vehicle >
        <my:Vehicle j:id="tt" > (Vehicle#2)
                <j:Vehicle elements{D,E,F}  >
                <j:PersonReference j:ref="zz"> (pointer to Person B)
        </my:Vehicle >
</my:SampleDocument>
```

In this example Person A and Person B has-a relationship to Vehicle#1 and Vehicle#2 has-a relationship to Person B

Note: The (VehicleReference) relationship of Person A and Person B to Vehicle#1 may not be the same as the (PersonReference) relationship of Vehicle#2 to Person B.

For example Person A and Person B may have the relationship of "photographed" Vehicle#1 but the Vehicle#2 to Person B relationship may be Vehicle#2 was "VandalizedBy" Person B.

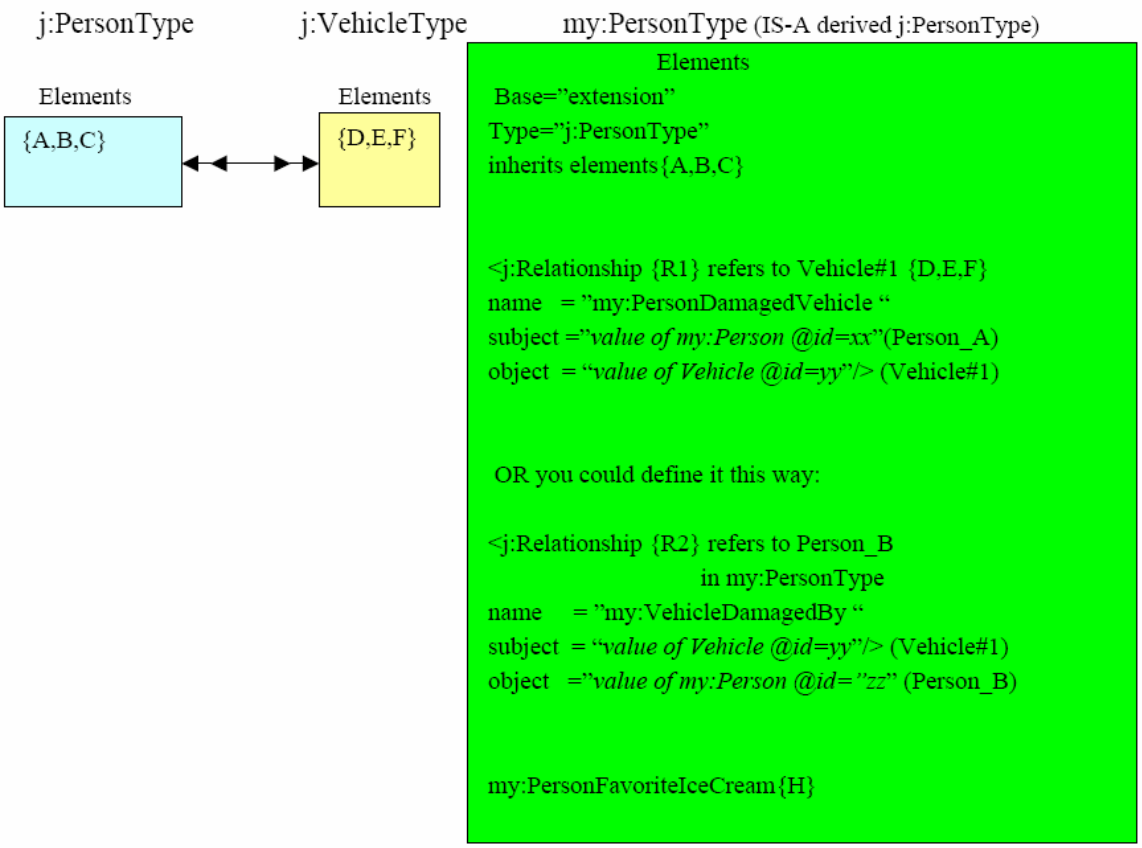
1389

Method4 : Extension using GJXDM Relationship element for Many-to-Many relationships

This method extends PersonType(component) {A,B,C}with a M:M "*relationship element*" to the VehicleType (component) {D,E,F} using j:Relationship Elements {R1,R2}. I will also add a "property element" as "my:PersonFavoriteIceCream" {H}. All elements will be added using a **Method-2** approach.

Extension Using "Cascading Extensions" and Relationship Elements

Before Extension ***After Extension(new M:M relationship)***



This **Method-4** construction includes two j:Relationship elements in my:PersonType for relating one or more persons to one or more vehicles by reference instead of the "Inclusion" method for adding "*relationship elements*" to a component. This avoids duplicating Vehicle#1 elements for Person_A and Person_B. Instead, Person_A has a Relationship Qname="my:PersonDamaged" with subject="xx" and Person_B with subject="zz" and Vehicle#1. Vehicle#1 is defined once in the

1390

Qname="my:VehicleDamagedBy" with subject="yy" (Vehicle_1) and object="zz" (Person_B). Vehicle#1 is defined once in the XML instance. This would not be the case if we instead used "Inclusion" with the element j:Vehicle as shown in Method-2 above. Following is an XML snippet of what it looks like:

```
<my:SampleDocument>

        <!--- Person_A --->
        <my:Person j:id="xx" >
                <j:PersonElements {A,B,C}>
                <j:Relationship name=" my:PersonDamagedVehicle " {R1}
                                        subject ="xx"(Person_A)
                                        object  = "yy"/> (Vehicle#1)
                <my:PersonFavoriteIceCream/ {H}>
        </my:Person>

        <!--- Person_B --->
        <my:Person j:id="zz" >
                <j:PersonElements {A,B,C}>
                <j:Relationship name=" my:VehicleDamagedBy " {R2}
                                        subject ="yy"( Vehicle#1)
                                        object  = "zz"/> (Person_B)
                <my:PersonFavoriteIceCream/ {H}>
        </my:Person>

        <!--- Vehicle_1 --->
        <j:Vehicle j:id="yy" > (Vehicle#1)
                <j:Vehicle elements{D,E,F}  >
        </j:Vehicle >

</my:SampleDocument>
```

There are many "*relationship elements*" already defined in GJXDM which should be used before defining a new relation between objects. For example, j:PropertyDisposition, j:CaseWitness, j:CaseCharge etc. all are describing "*relationship elements*" or non-hierarchical relationships between two GJXDM Components (Classes). The ReferenceType and RelationshipType elements need only be used when many-to-many relationships in your data model need to be resolved.

1391

6.6.1.3 Schema type refinement versus instance type substitution

Inclusion element approach. (Need IJIS writeup on type substitution.)

Using method 2 to include new elements into GJXDM components for extensions versus "type substitution" provides the schema developer with greater control over XML Instances and a greater assurance that XML Instance authors are not inadvertently substituting new types which the developer never intended. Also, "type substitution" at run-time may provide implementation errors for vendor product(s) due to the "dynamic" substitution of validating schema within an Instance XML document.

6.6.1.4 Substitution groups, redefines

GXS7, GXS12

XSD:substitution is "a feature of W3C XML Schema, allowing you to define groups of elements that may be used interchangeably in instance documents. They are not declared as element groups, but through the substitutionGroup attribute of xsd:element global definitions."

The GJXDM information exchange MNDR has made the decision to not allow the use of substitution groups due to the following issues:

- SubstitutionGroup "may work with some schema processors but relies on a liberal interpretation of the Recommendation, which may lead to interoperability issues."
- SubstitutionGroup "introduces multiple names for the same GJXDM element leading to confusion"

[GXS7]	The xsd:SubstitutionGroups feature MUST NOT be used.

<Redefine> enables you to create a local schema which includes all of the components and elements of the full GJXDM and then <redefine> selected GJXDM component(s) or element(s) within your local schema. The redesigned (redefined) GJXDM component(s) and element(s) OVERRIDE the definitions/structures defined in the Official GJXDM dictionary. All your XML instance tags from the redefines will appear to be original GJXDM elements instead of extensions|restrictions to GJXDM elements.

There is concern that the XML instance, when using <redefine>, will be indistinguishable from a schema comprised of GJXDM elements only because the local extension|restriction elements will all have the GJXDM namespace. For that reason, the GJXDM MNDR has formulated the following rule:

[GXS12]	The xsd:redefine element MUST NOT be used.

6.7 Document Schema Naming and Design Rules

NMS 12-13

Reviewers: Provide Comments to NMS12-NMS13 where defined in Appendix A (6/28: This is pending Scott Came's document update.)

6.7.1 Based on patterns (solutions in Context)

???? – Context driver discussion???? (6/28: The purpose of this section is to explain the various methods to create an extension, not the reason for the extension. However, this entry should be moved to the extension section, with an entry noting that since these solutions would always be non normative, (always based on subjective decisions based on the individual circumstances) further guidance is pending maturity of the GJXDM.

6.7.2 Document Schema Shell

Gtri Document schema shell

The basic GJXDM MNDR document schema template can be summarized in the following steps:

1)Define the target namespace of the document

- Reference/import the GJXDM
- Define the root element and type for the document
- Extend the root type from GJXDM DocumentType (or a derivative)
- Add document content

2) Create a local element and reuse a GJXDM type

- use actual GJXDM type
- extend GJXDM type

3) Add local components

- Declare globally
- Name all types
- Provide definitions
- Use naming conventions

6.7.3 Strongly-typed relationships

In Document Schema???? Not sure what this section is to address. Gtri Named RelationshipType???

6.7.4 Weakly-typed relationships

Use of ReferenceType or extensions which provide qualifiers to the ReferenceType via extension??

6.8 Tools for Schema Development)

6/28: After some discussion, group decided to replace a specific list with URLs to other tool sources, such as:

GTTAC Virtual Help Desk (once it's ready)

OJP Web Site

Robin Cover's tool sources

7 Instance Naming and Design Rules

7.1 Root Element

RED1

In XSD, every global element is eligible to act as a root element in an instance document. Rule ELD1 requires the identification of a single global element in each GJXDM MNDR schema to be carried as the root element in the instance document. GJXDM MNDR exchange documents (XML instances) must have a single root element as defined in the corresponding GJXDM MNDR document schema.

[RED1]	Every GJXDM instance document must have as its root element the single global element defined in its IEPD document schema.

7.2 XML Instance validation

IND1,IND4,IND7

Business information exchanges require a high degree of precision to ensure that application processing and corresponding business cycle actions are reflective of the purpose, intent, and information content agreed to by both trading partners. Schemas provide the necessary mechanism for ensuring that instance documents do in fact support these requirements.

[IND1]	All GJXDM instance documents MUST validate to a corresponding Document schema.

[IND4]	All GJXDM instance documents MUST contain the following namespace declaration in the root element: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

It is Recommended that Validating parsers SHOULD be able to override the schemaLocation attribute in processing XML Instance Documents. (note: Relying on XML Instance schemaLocation values without verifying before processing is a Security Risk)

7.3 Character encoding

IND2,IND3

[IND2]	All GJXDM instance documents MUST always identify their character encoding with the XML declaration.

[IND3]	In conformance with ISO/IETF/ITU/UNCEFACT Memorandum of Understanding Management Group (MOUMG) Resolution 01/08 (MOU/MG01n83), all GJXDM XML SHOULD be expressed using UTF-8.

7.4 Empty content

IND5,IND6

Usage of empty elements within XML instance documents are a source of controversy for a variety of reasons. An empty element does not simply represent data that is missing. It may express data that is not applicable for some reason, trigger the expression of an attribute, denote all possible values instead of just one, mark the end of a series of data, or appear as a result of an error in XML file generation. Conversely, missing data elements can also have meaning - data not provided by a trading partner. In information exchange environments, different trading partners may allow, require or ban empty elements. GJXDM MNDR has determined that empty elements do not provide the level of assurance necessary for business information exchanges and as such will not be used.

[IND5]	GJXDM conformant instance documents MUST NOT contain an element devoid of content unless explicitly indicated by the xsi:nil=”true” attribute

Absence of data should only be represented by using the “nillable” attribute as defined in Rule ATD5.

[IND6]	The absence of a construct or data in a GJXDM instance document MUST NOT carry meaning.

Example:

```
Valid:

<PersonName>
      <PersonGivenName>John</PersonGivenName>
      <PersonMiddleName xsi:nil="true"/>
      <PersonSurName>Doe</PersonSurName>
</PersonName>

Invalid:
```

```
<PersonName>
      <PersonGivenName>John</PersonGivenName>
      <PersonMiddleName/> ***EMPTY Content***
      <PersonSurName>Doe</PersonSurName>
</PersonName>
```

7.5 Evaluate UBL Instance Document Specifications

Tbd (John Ruegg)

Appendix A. GJXDM MNDR Checklist

The following checklist incorporates relevant UBL XML naming and design rules as defined in UBL Naming and Design Rules version 1.0.1, 15 November 2004. UBL rules modified for GJXDM information exchange schema(s) are highlighted in GREEN. Additional rules are included to address GJXDM information exchange specific Naming & Design rules drawn from GTRI GJXDM Training Workshop materials and related documents from IJIS and OASIS LegalXML Integrated Justice Technical Committee documents. The checklist is in alphabetical sequence as follows:

Attribute Declaration Rules	(ATD)
Code List Rules	(CDL)
ComplexType Definition Rules	(CTD)
ComplexType Naming Rules	(CTN)
Documentation Rules	(DOC)
Element Declaration Rules	(ELD)
General Naming Rules	(GNR)
General Type Definition Rules	(GTD)
General XML Schema Rules	(GXS)
Instance Document Rules	(IND)
Modeling Constraints Rules	(MDC)
Namespace Rules	(NMS)
Root Element Declaration Rules	(RED)
Schema Structure Modularity Rules	(SSM)
Standards Adherence Rules	(STA)
Versioning Rules	(VER)

A.1 Attribute Declaration Rules

[ATD1]	User defined attributes SHOULD NOT be used. (I found no Global JXDM GTRI Training material which allowed or disallowed adding user-defined attributes – Is this rule valid???)
[ATD2]	If a Schema Expression contains one or more common attributes that apply to all elements contained or included or imported therein, the common attributes MUST be declared as part of a global attribute group. (For example: see the Global JXDM global attribute group named ”**SuperTypeMetadata”)**
[ATD3]	For each `Global JXDM user-defined element of simpleType and a xsd:restriction` element; an `xsd:base` attribute MUST be declared and set to the appropriate GJXDM datatype. `Note: the set of valid GJXDM datatypes are based on ebXML Core Component Technical Specification v1.9 and include the following 10 simpleTypes:` • Amount • BinaryObject (secondary: Graphic, Picture,Sound,Video) • Code • DateTime (secondary: Date, Time) • Identifier (authorized abbreviation: ID) • Indicator • Measure • Numeric (secondary: Value, Rate, Percent) • Quantity • Text (secondary: Name) Reference:GTRI May 2004 Developer Workshop
[ATD4]	Each xsd:schemaLocation attribute declaration MUST contain a system-resolvable URI, referencing the location of the schema or schema module.

[ATD5]	The xsd built in nillable attribute MUST be used and set nillable=”true” for any Global JXDM user-defined element which has simpleContent. Note: An example from GJXDM v3.0.2 : <xsd:element name="**WitnessLocationDescriptionText**" type="**j:TextType**" nillable="**true**">
[ATD6]	The `xsd:any attribute` MUST NOT be used.

1567
1568

A.2 Code List Rules

[CDL1]	All Global JXDM Code Lists MUST be part of a Global JXDM or externally maintained Code List; they MUST NOT be included in a document or extension schema..
[CDL2]	The Global JXDM SHOULD identify and use external standardized code lists whenever practical rather than develop its own Global JXDM-native code lists.
[CDL3]	The Global JXDM information exchange developer, through extension/restriction, MAY design and use ancontextually defined code list where an existing GJXDM code list needs to be extended, or where no suitable external code list exists.
[CDL4]	All GJXDM maintained or information exchange developer Code Lists MUST be enumerated using the GJXDM Code List Schema Module. (See Global JXDM standards for **NonStandardCodeType** properties.) TRUE???? **Need to clarify the intent of NonStandardCodeType with the XSTF. (Tom Carlson will investigate).**
[CDL5]	The name of each GJXDM information exchange Code List Schema MUST be of the form: {`Owning Organization`}_{`Code List Name`}_[version number}].xsd ;
[CDL6]	An `xsd:import` element MUST be declared for every code list required in a GJXDM information exchange schema. Each codelist MUST be in its own namespace; the namespace identifier MUST be consistent with the same rules as extension schemas.

[CDL7]	When creating a local code list, MUST follow the UBL code list schema and annotation rules. (need to coordinate this rule with CDL4)

1569
1570

A.2 Code List Rules

[CDL8]	Users of the GJXDM MAY identify any subset they wish from an identified code list for their own trading community conformance requirements.
[CDL9]	The `xsd:schemaLocation` MUST include the complete URI used to identify the relevant code list schema.

1571

A.3 ComplexType Definition Rules

[CTD1]	For every class identified in GJXDM extension and document schema, a named `xsd:complexType` MUST be defined.
[CTD2]	Every `GJXDM user-defined xsd:complexType` definition content model MUST use the `xsd:sequence` element with appropriate global element references to reflect each property of its class.
[CTD3]	For every user-defined datatype based on the valid set of GJXDM datatypes, a named `xsd:complexType` or `xsd:simpleType` MUST be defined.

1572
1573
1574
1575

A.4 ComplexType Naming Rules

[CTN1]	A user-defined `xsd:complexType` ***name*** MUST name the object suffixed by the word `"Type"`. For example <xsd:complexType name="PersonType"> is correct

	And <xsd:complexType name=”Person”> would be incorrect.

A.5 Documentation Rules

[DOC1]	The `xsd:documentation` element for every GJXDM user-defined Element MUST contain a structured set of annotations in the following sequence and pattern: • Version (optional): An indication of the evolution over time of the Datatype. • Definition(mandatory): The semantic meaning of an Element • Cardinality(mandatory): Indication whether the complexType Element (Property) represents a not-applicable, optional, mandatory and/or repetitive characteristic of the parent complexType • AssociatedObjectClassQualifier (optional): Associated Object Class Qualifiers describe the 'context' of the relationship with another complexType object. That is, it is the role the contained Element plays within its association with the containing complexType object. • AssociatedObjectClass (mandatory); Associated Object Class is the Object Class at the other end of this association. It represents the Aggregate Business Information Entity contained by the Association Business Information Entity. (For example: the element PersonName within the complexType PersonType has Name as the AssociatedObjectClass contained in the Aggregate Business Information Entity called PersonType.) • AlternativeBusinessTerms (optional): Any synonym terms under which the Element is commonly known and used in the business. • Examples (optional): Examples of possible values for the Element Context Element should go here as well if not defined in a later NDR, so this is a placeholder for now. The list of Context Drivers is in the UBL ccts: core-component parameters schema. The current values for Context drivers are:

	8 Business Process 9 Product Classification 10 Geopolitical Region 11 Official Constraint 12 Business Process Role 13 Supporting Role 14 System Capabilities

A.6 Element Declaration Rules

[ELD1]	Each `GJXDM information exchange document schema` MUST identify one and only one global element declaration that defines the exchange document being conveyed in the Schema expression. That global element MUST include an xsd:annotation child element which MUST further contain an `xsd:documentation` child element that declares "*`This element MUST be conveyed as the root element in any instance document based on this Schema expression.`*"

[ELD2]	All element declarations MUST be global
[ELD3]	For every class defined as a GJXDM user-defined types, a global element bound to the corresponding `xsd:complexType` MUST be declared. For e.g. a schema defining a complexType named my:FavoritePersonType would need to declare a global element named "FavoritePerson" of objectType = my:FavoritePersonType to bind a global element name to the complexType.

A.6 Element Declaration Rules

[ELD4]	When a "relationship" element is unqualified, it is bound via reference to the global object-class element to which it is associated. When a "relationship" element is qualified, a new element MUST be declared and bound to the xsd:complexType of its associated object-class.
[ELD5]	For every user-defined simpleType, an xsd:restriction element MUST be declared
[ELD6]	Empty elements MUST NOT be declared.
[ELD7]	Global simpleType elements declared with Qualified Properties must be of the same type as their corresponding Unqualified Property.
[ELD8]	The `xsd:any` element MUST NOT be used.

A.7 General Naming Rules

[GNR1]	GJXDM information exchange XML element, attribute and type names MUST be in the English language, using the primary English spellings provided in the Oxford English Dictionary.
[GNR2]	GJXDM information exchange XML element, attribute and type names MUST be ebXML CCTS ISO 11179 compliant

[GNR3]	GJXDM information exchange XML element, attribute and type names MUST NOT include spaces, other separators, or characters not allowed by W3C XML 1.0 for XML names. The only **exception** to this rule is the use of periods in GJXDM element names.
[GNR4]	GJXDM information exchange XML element, attribute, and simple and complex type names MUST NOT use acronyms, abbreviations, or other word truncations, except those in the list of exceptions published in Appendix B.
[GNR5]	The acronyms and abbreviations listed in Appendix B MUST always be used.

[GNR6]	GJXDM information exchange XML element, attribute and type names MUST be in singular form unless the concept itself is plural.
[GNR7]	The UpperCamelCase (UCC) convention MUST be used for naming elements and types.
[GNR8]	The lowerCamelCase (LCC) convention MUST be used for naming attributes.

A.8 General Type Definition Rules

[GTD1]	All types MUST be named.
[GTD2]	The `xsd:anyType` MUST NOT be used.

A.9 General XML Schema Rules

[GXS1]	GJXDM subset schema or constraint schema MUST conform to the following physical layout as applicable: • XML Declaration • <!-- ===== Required Documentation Comments Block ===== --> • <!-- ===== Name (common): …… ===== --> • <!-- ===== Authoring agency/jurisdiction/generation date: …… ===== -->> • <!-- ===== Description of business usage: …… ===== --> • <!-- ===== xsd:schema Element With Namespaces Declarations ===== --> • xsd:schema element to include Attribute definitions

	attributeFormDefault =”unqualified” elementFormDefault= “qualified” followed by Namespace Declarations in this order: • Target namespace • Default namespace • <!-- ===== Imports ===== --> • External Codelist Namespaces • xmlns:xsd • External Codelist import schemaLocations and namespaces • <!-- ===== Global Attributes ===== --> • Global Attributes and Attribute Groups • <!-- ===== Complex Types and Simple Types ===== --> • <!-- ===== in alphabetized order xsd:TypeDefinitions ===== --> • Complex and Simple Types • <!-- ===== Attribute Declarations SHOULD BE in alphabetized order ===== --> • <!-- ===== Element Declarations SHOULD BE in alphabetized order ===== -->
[GXS2]	GJXDM extension schema MUST conform to the following physical layout as applicable: • XML Declaration • <!-- ===== Required Documentation Comments Block ===== --> • <!-- ===== Name (common): ……. ===== --> • <!-- ===== Authoring agency/jurisdiction/generation date: ==== --> • <!-- ===== Description of business usage: …… ===== --> • <!-- ===== xsd:schema Element With Namespaces Declarations ===== --> • xsd:schema element to include Attribute definitions attributeFormDefault =”unqualified” elementFormDefault= “qualified” followed by Namespace Declarations in this order: • Target namespace for Extension schema (http://{my namespace}/…/extension[/name]/version • No Default namespace, a token such as ext: should be used for the Extension schema targetNamespace (eg. xmlns:ext=”http://{my namespace}…/extension/…”) • Declare the GJXDM schema, subset schema or constraint schema

	namespace (eg. xmlns:j="http://www.it.ojp.gov/jxdm/{jxdm version}") • xmlns:xsd • <!-- ===== Imports ===== --> • External GJXDM reference , subset schema or constraint schema import namespace and relevant schemaLocation • <!-- ===== Extended/Restricted GJXDM Complex Types and Simple Types = --> • <!-- ===== in alphabetized order xsd:TypeDefinitions ===== --> • Complex and Simple Types • <!-- ===== Element Declarations in alphabetized order ===== -->
[GXS3]	GJXDM Document schema MUST conform to the following physical layout as applicable: • XML Declaration • <!-- ===== Required Documentation Comments Block ===== --> • <!-- ===== Name (common): ……. ===== --> • <!-- ===== Authoring agency/jurisdiction/generation date: …… ===== --> • <!-- ===== Description of business usage: …… ===== --> • <!-- ===== xsd:schema Element With Namespaces Declarations ===== --> • xsd:schema element followed by Namespace Declarations in this order: • Target namespace for Document schema • No Default namespace, a token such as doc: or rap: (for a rapsheet) should be used for the Document schema targetNamespace • Declare the "optional" GJXDM extension schema namespace (eg. xmlns:ext="http://{my namespace}…/extension/…") • Declare the GJXDM schema, subset schema or schema constraint namespace (eg. xmlns:j="http://www.it.ojp.gov/jxdm/{jxdm version}") • xmlns:xsd • <!-- ===== Imports ===== --> • External "optional" GJXDM extension schema import namespace and schemaLocation

	• External GJXDM schema, subset schema or constraint schema import namespace and schemaLocation • <!-- ===== Root Element ===== --> • Root Element Declaration • Root Element Type Definition • <!-- ===== Type Definition ===== --> • Define Root Type; extend from j:DocumentType; where complexType name="{Root Element Name}Type" (eg. for Root Element Name "CitationDocument" type="doc:CitationDocumentType" then the complexType name = "CitationDocumentType" which is the Root Element Name suffixed with the word "Type"). • Optionally, define local name space type definitions in alphabetical order • Optionally, define Elements in alphabetical order
[GXS4]	The root element in all GJXDM information exchange Schema modules MUST contain the following declaration: "xmlns:xsd=http://www.w3.org/2001/XMLSchema."
[GXS5]	GJXDM information exchange schema developers MAY provide a run-time schema devoid of documentation in addition to the fully annotated version.
[GXS6]	GJXDM defined xsd:simpleTypes SHOULD be used as the base for any user-defined simpleTypes via extension or restriction to the GJXDM simpleType.
[GXS7]	The `xsd:SubstitutionGroups` feature MUST NOT be used.
[GXS8]	The xsd:final attribute MUST be used to control extensions.
[GXS9]	The `xsd:block` attribute SHOULD be used to restrict use of "type substitution" and "element substitution" in XML instance documents.
[GXS10]	`xsd:notations` MUST NOT be used.
[GXS11]	The `xsd:all` element MUST NOT be used.

[GXS12]	The `xsd:redefine` element MUST NOT be used.
[GXS13]	The `xsd:choice` element SHOULD NOT be used where customization and extensibility are a concern.

A.9 General XML Schema Rules

[GXS14]	The `xsd:include` feature MUST only be used within a document schema.
[GXS15]	GJXDM designed schema MAY use xsd:appinfo. If used, xsd:appinfo MUST only be used to convey non-normative information. Note: appinfo is a recent addition to GJXDM in Version 3.0.2
[GXS16]	Complex Type extension and restriction MAY be used where appropriate.

A.10 Instance Document Rules

[IND1]	All GJXDM instance documents MUST validate to a corresponding Document schema.
[IND2]	All GJXDM instance documents MUST always identify their character encoding with the XML declaration.
[IND3]	In conformance with ISO/IETF/ITU/UNCEFACT Memorandum of Understanding Management Group (MOUMG) Resolution 01/08 (MOU/MG01n83), all GJXDM XML SHOULD be expressed using UTF-8.
[IND4]	All GJXDM instance documents MUST contain the following namespace declaration in the root element: `xmlns:xsi`="http://www.w3.org/2001/XMLSchema-instance"
[IND5]	GJXDM conformant instance documents MUST NOT contain an element devoid of content unless explicitly indicated by the xsi:nil=”true” attribute
[IND6]	The absence of a construct or data in a GJXDM instance document MUST NOT carry meaning.

A.11 Modeling Constraints Rules	
[MDC1]	Mixed content MUST NOT be used except where contained in an `xsd:documentation` element.

A.12 Namespace and Schema Filename Rules	
[NMS1]	Every GJXDM information exchange schema MUST have a namespace declared using the `xsd:targetNamespace` attribute.
[NMS2]	Every GJXDM information exchange schema version MUST have its own unique namespace.
[NMS3]	A GJXDM schema subset MUST declare the same xsd:targetNamespace as the GJXDM baseline schema.
[NMS4]	GJXDM namespaces MUST only contain GJXDM conformant schema modules.

A.12 Namespace and Schema Filename Rules	

[NMS5]	GJXDM published namespaces MUST never be changed. The namespace names for GJXDM reference schema releases are of the form:
	
	`http://www.it.ojp.gov/jxdm/{major version . minor release . revision}`
	
	`For example the following namespace http://…/jxdm/3.1.1 would be major-release 3 and minor-release 1.1 of the GJXDM schema.`

[NMS6]	Each `GJXDM imported CodeList` schema MUST be maintained in a separate namespace. The proxy Codelist URL MUST be of the form: "http://`www.it.ojp.gov/jxdm/{GJXDM` version}/proxy/{external original codelist name}/{original codelist version}" . For example, xmlns:j-usps = "http://…/jxdm/3.0.2/proxy/usps_states/1.0" represents the United

	States Postal Services States Code table version 1.0 which is imported into GJXDM schema version 3.0.2 . The proxy namespace "j-usps" is a concatenation of the Justice namespace 'j' and the original source code-list namespace "usps" forming the general rule for Proxy Namespace as "j-xsd"; where xsd is the same as the "original" codelist namespace. Proxy schemas provide an intermediary between external namespaces and the GJXDM, and use of Proxies guarantee SuperType metadata on elements based on entities from external namespaces.
[NMS7]	Each schema, for a GIEP group, MUST be maintained in a separate namespace but will share a common group path. That GIEP group path MUST be of the form: "http://.../`jxdm/{GJXDM` version}/<GIEP group name>/<GIEP version>/". For example, xmlns = "http://.../jxdm/3.0/Citation/1.0./" represents a local namespace copy of a GIEP group of schemas that support the exchange or representation of a Citation reference document. The version of 1.0 is assigned to the GIEP name space by the provider of the reference document. Note: All associated subset, document, constraint and extension schemas must be placed within this path. If any of the associated schemas, within the GIEP group name change, the GIEP group version MUST change. **[Will adjust/add rules regarding the namespace and schema locations for each of these files.]**
[NMS8]	A `GJXDM Subset` schema module MUST be in its own user-namespace schema location and location MUST be of the form: "http://.../`jxdm/{GJXDM` version}/<GIEP Group name>/<GIEP version>/ <name>/". For example, xmlns = "http://.../jxdm/3.0/Citation/1.0/jxdm-SUB-Citation.xsd" represents a local namespace copy of a conformant subset of the full GJXDM v3.0 schema. The GIEP Citation Version 1.0 contains a subset schema named "jxdm-SUB-Citation.xsd".

[NMS9]	The GJXDM Subset schema module using a local schemaLocation MUST have a schema filename of the following form: jxdm-SUB[-name].xsd For example, the local GJXDM schema filename for targetNamespace= "http://www.it.ojp.gov/jxdm/3.0" would be: schemaLocation="http://.../jxdm/3.0/Citation/1.0/jxdm-SUB-Citation.xsd" would represent a subset schema named "jxdm-SUB-Citation.xsd in the GIEP Group Citation version 1.0.
[NMS10]	Each GJXDM Extension schema module, associated with a GIEP group, MUST be maintained in a separate local namespace and MUST be of the form: "http://.../jxdm/{GJXDM version}/<GIEP Group name>[/<GIEP version>]/ <name>/)". For example, xmlns = "http://.../jxdm/3.0/Citation/1.0/jxdm-EXT-Citation.xsd" represents a local namespace copy of a extension schema containing user-defined Types derived from extending or restricting GJXDM version 3.0 schema complexTypes and simpleTypes. The GIEP Citation Version 1.0 contains a extension schema named "jxdm-EXT-Citation.xsd".
[NMS11]	The GJXDM Extension schema module using a local schemaLocation MUST have a schema filename of the following form: jxdm-EXT[-name].xsd For example, the local GJXDM schema filename for targetNamespace= "http://www.it.ojp.gov/jxdm/3.0" would be:

	schemaLocation="http://.../jxdm/3.0/Citation/1.0/jxdm-EXT-Citation.xsd" would represent a extension schema named "jxdm-EXT-Citation.xsd in the GIEP Group Citation version 1.0.
[NMS12]	Each GJXDM Document schema module MUST be maintained in a separate local namespace. The local namespace URI MUST be of the form: "http://.../jxdm/{GJXDM version}/<GIEP Group name>[/<GIEP version>]/ < name>/)". For example, xmlns = "http://.../jxdm/3.0/Citation/1.0/jxdm-DOC-Citation.xsd" represents a local namespace copy of a document schema. The GIEP Citation Version 1.0 contains a document schema named "jxdm-DOC-Citation.xsd".
[NMS13]	The GJXDM Document schema MUST have a schema filename of the following form: jxdm-DOC-<rootElement>].xsd For example, the local GJXDM schema filename for targetNamespace= "http://www.it.ojp.gov/jxdm/3.0" would be: schemaLocation="http://.../jxdm/3.0/Citation/1.0/jxdm-DOC-Citation.xsd" would represent a document schema named "jxdm-DOC-Citation.xsd in the GIEP Group Citation version 1.0.
[NMS14]	Each GJXDM Constraint schema module MUST be maintained in a separate namespace. The Constraint schema targetNamespace MUST be of the form: "http://www.it.ojp.gov/jxdm/{GJXDM version}"

[NMS15]	A GJXDM Contraint schema modules MUST be in its own user-namespace schema location and location MUST be of the form: "http://.../jxdm/{GJXDM version}/<GIEP Group name>[/<GIEP version>]/ <name>/)". For example, xmlns = "http://.../jxdm/3.0/Citation/1.0/jxdm-CON-Citation.xsd" represents a local namespace copy of a conformant subset of the full GJXDM v3.0 schema. The GIEP Citation Version 1.0 contains a constraint schema named "jxdm-CON-Citation.xsd".
[NMS16]	The GJXDM Constraint schema module using a local schemaLocation MUST have a schema filename of the following form: jxdm-CON[-name].xsd For example, the local GJXDM schema filename for targetNamespace= "http://www.it.ojp.gov/jxdm/3.0" would be: schemaLocation="http://.../jxdm/3.0/Citation/1.0/jxdm-CON-Citation.xsd" would represent a constraint schema named "jxdm-CON-Citation.xsd in the GIEP Group Citation version 1.0. Note: The CON represents Constraint Schema and the optional [name] provides for maintaining multiple constraint schemas for different reference document schemas. The version suffix identifies the specific version of the local constraint schema.
[NMS17]	Tbd - Placeholder for defining namespace for the .zip package which could be setup as another local namespace /supplemental and provides developers a convenient place to fetch the schema packages for their local implementations. -OR- the Document Schema Root Element could use <documentation source=URL of supplemental documentation> -OR- other options?? GTRI Training Material is silent on artifact filename standards for .xsd, .xml, .xsl , .doc etc. This [NMSxx] section has adopted a modified version of the UBL filename standards. Note: The convention of prefixing the <name> with jxdm- is borrowed from UBL but the prefixes of ES, CS were extracted

	from GTRI training material. For consistency, I added the prefixes DS and SS so that all developer schema follows the same naming rules.
[NMS18]	Also, if adopted, the UBL core-components parameters schema MAY be included. (place where schema annotation/documentation elements are defined including context drivers) could be added back into this mndr. But for now, holding off on the UBL metadata per subcommittee member requests. Could be included within extension schema as superattributegroup maybe called component metadata?? Namespace could be referenced directly from document schema or via extension schema. (John R will update this section to incorporate Scott C's revisions. Essentially, this information would be included either within the GJXDM schema xsd:documentation annotation structures) (context; classification)
[NMS19]	**Note: This rule was added by Scott Came on 5/10; need to adjust the other NMS rules to be consistent.** An IEPD MUST be delivered as a ZIP file. The ZIP file MUST have the following structure and contents. The root directory in the ZIP archive must contain the following files and directories: • The GIEPD Overview document file, in a suitable file format, and named "GIEPD Overview" with a file extension corresponding to the format • A directory named "domain model artifacts" • A directory named "mapping artifacts" • A directory named "schemas" • A directory named "sample instances" The directory named "domain model artifacts" MUST contain all artifacts related to the domain model of the IEP (rules about these artifacts are contained in section 4 of this specification). The directory named "mapping artifacts" MUST contain all artifacts related to the mapping of the domain model to GJXDM, as discussed in section 5 of this specification. The directory named "sample instances" MUST contain one or more sample XML instances that are valid against the document, extension, constraint, and subset schemas in the IEP. Each sample XML instance MUST associate referenced IEP namespaces by using the xsi:schemaLocation attribute on the root element; the xsi:schemaLocation attribute MUST use a relative URL, valid within the IEP structure documented here, to locate the schema for each namespace.

	The directory named “schemas” MUST contain the following: • A file named document-schema.xsd that contains the document schema • A file named extension-schema.xsd that contains the extension schema, if the IEP uses an extension schema • A file named constraint-schema.xsd that contains the constraint schema, if the IEP uses a constraint schema • A directory named “subset” that contains the subset schema set; the sub-directory structure underneath the “subset” directory must match the directory structure of the GJXDM distribution version being used To the extent that schemas in the IEP import each other, the schemaLocation attribute in the each schema’s xsd:import element(s) MUST use a relative URL to locate the imported schema. The relative URL MUST be valid within the above structure. The subset and constraint schemas included within the IEP MUST retain the GJXDM namespace identifier for whatever version of GJXDM is being used in the IEP. Subset and constraint schemas MAY NOT alter the GJXDM namespace identifier. Extension and document schemas included within the IEP MUST specify a targetNamespace on the xsd:schema element that contains a namespace identifier conformant with the other rules in this section.

A.13 Root Element Declaration Rules

[RED1]	Every GJXDM instance document must use the global element defined as the root element in the schema as its root element.

A.14 Schema Structure Modularity Rules

[SSM1]	GJXDM Schema MAY be split into multiple subset schema modules.

[SSM2]	A document schema in one namespace that is dependent upon type definitions or element declarations defined in another namespace MUST only import the document schema from that namespace.
[SSM3]	A document schema in one namespace that is dependant upon type definitions or element declarations defined in another namespace MUST NOT import internal schema modules from that namespace.
[SSM4]	Imported schema modules MUST be fully conformant with GJXDM information exchange schema naming and design rules.
[SSM5]	GJXDM schema modules MUST either be treated as external schema modules or as internal schema modules of the document schema.
[SSM6]	All internal schema modules MUST be in the same namespace as their corresponding document schema.
[SSM7]	GJXDM schema module(s) MAY be created for reusable components.

A.15 Standards Adherence rules

[STA1]	All Global JXDM information exchange schema design rules MUST be based on the W3C XML Schema Recommendations: XML Schema Part 1: Structures and XML Schema Part 2: Datatypes.
[STA2]	All Global JXDM information exchange schema and messages MUST be based on the W3C suite of technical specifications holding recommendation status.

A.16 Versioning Rules

[VER1]	Every GJXDM information exchange schema and schema module major version committee ***draft*** MUST have a version number of the form: <major>.0[.<revision>]
[VER2]	Every GJXDM Information exchange Schema and schema module major version OASIS ***Standard*** MUST have a version number of the form: <major>.0
[VER3]	Every minor version release of a GJXDM Information exchange schema or schema module ***draft*** MUST have a version number of the form: <major >.<non-zero>[.<revision>]

A.16 Versioning Rules Continued

[VER4]	Every minor version release of a GJXDM information exchange schema or schema module ***Standard*** MUST have a version number of the form: <major >.<non-zero>
[VER5]	For GJXDM information exchange schema minor version changes, the <document name> MUST NOT change.
[VER6]	For every GJXDM information exchange schema and schema module, the major version number MUST be a sequentially assigned, incremental number greater than zero.
[VER7]	For every GJXDM information exchange schema and schema module, the minor version number MUST be a sequentially assigned, incremental non-negative integer.

[VER10]	GJXDM information exchange schema and schema module minor version changes MUST not break semantic compatibility with prior versions., nor may they break existing document instances that are based on any earlier minor version of the last major version. For example, an instance document buiild on a 1.1 minor version must be able to be processed by any later minor release, for example, a 1.9 version, Minor versions maintain forward compatibility. .
[VER11]	GJXDM information exchange schema and schema module major version changes MAY break semantic and/or structural compatibility with prior versions. No backward compatibility is guaranteed.
	(5/13/05: Need to include rule(s) explaining how to use revision; Sylvia Webb will provide additional clarification.. NOTES: - To support pre-release work (alpha suffix) - Post release, useful when there have been no functional/behavior changes, but changes in comments, documentation, or syntax definitions (numeric suffix).

Appendix B. Approved Acronyms and Abbreviations

The following Acronyms and Abbreviations have been approved by the GJXDM MNDR Subcommittee for GJXDM use:

NEED TO REVIEW GJXDM AND PUT THEM HERE

A Dun & Bradstreet Data Universal Numbering System (DUNS) number *must* appear as "DUNS".

"Identifier" *must* appear as "ID".

"Uniform Resource Identifier" *must* appear as "URI"

This list will henceforth be maintained by the GJXDM XSTF committee, and additions included in current and future versions of the GJXDM standard will be maintained and published separately.

Appendix C. Technical Terminology

Ad hoc schema processing	Doing partial schema processing, but not with official schema validator software; e.g., reading through schema to get the default values out of it.
Aggregate Business Information Entity (ABIE)	A collection of related pieces of business information that together convey a distinct business meaning in a specific Business Context. Expressed in modeling terms, it is the representation of an Object Class, in a specific Business Context.
Application-level validation	Adherence to business requirements, such as valid account numbers.
Assembly	Using parts of the library of reusable GJXDM components to create a new kind of business document type.
Business Context	Defines a context in which a business has chosen to employ an information entity. The formal description of a specific business circumstance as identified by the values of a set of *Context Categories*, allowing different business circumstances to be uniquely distinguished.

Business Object	An unambiguously identified, specified, referenceable, registerable and re-useable scenario or scenario component of a business transaction. The term business object is used in two distinct but related ways, with slightly different meanings for each usage: In a business model, business objects describe a business itself, and its business context. The business objects capture business concepts and express an abstract view of the business's "real world". The term "modeling business object" is used to designate this usage. In a design for a software system or in program code, business objects reflects how business concepts are represented in software. The abstraction here reflects the transformation of business ideas into a software realization. The term "systems business objects" is used to designate this usage.
business semantic(s)	A precise meaning of words from a business perspective.

Business Term	This is a synonym under which the Core Component or Business Information Entity is commonly known and used in the business. A Core Component or Business Information Entity may have several business terms or synonyms.
class	A description of a set of objects that share the same attributes, operations, methods, relationships, and semantics. A class may use a set of interfaces to specify collections of operations it provides to its environment. See interface.
class diagram	Shows static structure of concepts, types, and classes. Concepts show how users think about the world; types show interfaces of software components; classes show implementation of software components. (OMG Distilled) A diagram that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships. (Rational Unified Process)
classification scheme	This is an officially supported scheme to describe a given *Context Category*

Common attribute	An attribute that has identical meaning on the multiple elements on which it appears. A common attribute might or might not correspond to an XSD global attribute.
component	One of the individual entities contributing to a whole.
context	Defines the circumstances in which a Business Process may be used. This is specified by a set of Context Categories known as Business Context. (See Business Context.)
context category	A group of one or more related values used to express a characteristic of a business circumstance.
Document schema	A schema document corresponding to a single namespace, which is likely to pull in (by including or importing) schema modules.
Core Component	A building block for the creation of a semantically correct and meaningful information exchange package. It contains only the information pieces necessary to describe a specific concept.

Core Component Type	A Core Component which consists of one and only one Content Component that carries the actual content plus one or more Supplementary Components giving an essential extra definition to the Content Component. Core Component Types do not have business semantics.
Datatype	A descriptor of a set of values that lack identity and whose operations do not have side effects. Datatypes include primitive pre-defined types and user-definable types. Pre-defined types include numbers, string and time. User-definable types include enumerations. (XSD) Defines the set of valid values that can be used for a particular *Basic Core Component Property* or *Basic Business Information Entity Property*. It is defined by specifying restrictions on the *Core Component Type* that forms the basis of the *Datatype*. (CCTS)
Generic BIE	A semantic model that has a "zeroed" context. We are assuming that it covers the requirements of 80% of business uses, and therefore is useful in that state.
instance	An individual entity satisfying the description of a class or type.

Instance constraint checking	Additional validation checking of an instance, beyond what XSD makes available, that relies only on constraints describable in terms of the instance and not additional business knowledge; e.g., checking co-occurrence constraints across elements and attributes. Such constraints might be able to be described in terms of Schematron.
Instance root/doctype	This is still mushy. The transitive closure of all the declarations imported from whatever namespaces are necessary. A doctype may have several namespaces used within it.
Intermediate element	An element not at the top level that is of a complex type, only containing other elements and attributes.
Internal schema module	A schema module that does not declare a target namespace.
Leaf element	An element containing only character data (though it may also have attributes). Note that, because of the XSD mechanisms involved, a leaf element that has attributes must be declared as having a complex type, but a leaf element with no attributes may be declared with either a simple type or a complex type.

Lower-level element	An element that appears inside a business message. Lower-level elements consist of intermediate and leaf level.
Object Class	The logical data grouping (in a logical data model) to which a data element belongs (ISO11179). The *Object Class* is the part of a *Core Component*'s *Dictionary Entry Name* that represents an activity or object in a specific *Context*.
Namespace schema module:	A schema module that declares a target namespace and is likely to pull in (by including or importing) schema modules.
Naming Convention	The set of rules that together comprise how the dictionary entry name for *Core Components* and *Business Information Entities* are constructed.
(XML) Schema	An XML Schema consists of components such as type definitions and element declarations. These can be used to assess the validity of well-formed element and attribute information items (as defined in [XML-Infoset]), and furthermore may specify augmentations to those items and their descendants.

Schema module	A collection of XML constructs that together constitute an XSD conformant schema. Schema modules are intended to be used in combination with other XSD conformant schema.
Schema Processing	Schema validation checking plus provision of default values and provision of new infoset properties.
Schema Validation	Adherence to an XSD schema.
semantic	Relating to meaning in language; relating to the connotations of words.
Top-level element	An element that encloses a whole business message. Note that GJXDM business messages might be carried by messaging transport protocols that themselves have higher-level XML structure. Thus, a GJXDM top-level element is not necessarily the root element of the XML document that carries it.
type	Description of a set of entities that share common characteristics, relations, attributes, and semantics. A stereotype of class that is used to specify an area of instances (objects) together with the operations applicable to the objects. A type may not contain any methods. See class, instance. Contrast interface.

Appendix D. References

[CCTS] ISO 15000-5 ebXML Core Components Technical Specification

[ISONaming] *ISO/IEC 11179,* Final committee draft, Parts 1-6.

(RFC) 2119 S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

(RFC) 3121 http://www.faqs.org/rfcs/rfc3121.html

[UBLChart] UBL TC Charter, http://oasis-open.org/committees/ubl/charter/ubl.htm

[XML] *Extensible Markup Language (XML) 1.0* (Second Edition), W3C Recommendation, October 6, 2000
(XSD) *XML Schema*, W3C Recommendations Parts 0, 1, and 2. 2 May 2001.

(XHTML) XHTML™ Basic, W3C Recommendation 19 December 2000: http://www.w3.org/TR/2000/REC-xhtml-basic-20001219

Appendix E. Acknowledgments

The following individuals were members of the committee during the development of this specification:

- Jane Doe, Example Corp.
- A. Nonymous (chair), Example Corp.
- John Smith, Example Corp.
- Karl Best, OASIS
- John Doe, Other Examples, Inc.
- Eve Maler, Sun Microsystems
- Norman Walsh, Sun Microsystems

In addition, the following people made contributions to this specification:

- Joe Blow, Example Corp.

Appendix F. Revision History

[This appendix is optional, but helpful. It should be removed for specifications that are at OASIS Standard level.]

Rev	Date	By Whom	What
wd-.01	2005-02-10	John Ruegg	Initial version
wd-.02	2005-03-10	John Ruegg	Incorporates decision to omit UBL component terminology for this first version
wd-.03	2005-04-14	John Ruegg	Updated with Subcommittee member comments.

Appendix G. Notices

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.