XML encoding rules for HL7 v2 messages

Editor: Kai U. Heitmann, M. D.
University of Cologne, Germany

Co-Chairs: Paul V. Biron
Kaiser Permanente
Kai U. Heitmann, M. D.
University of Cologne, Germany

Preface

Chapter 2 of the HL7 Version 2.3.1 and 2.4 [rfHL7v231, rfHL7v24] specifies standard message structures (syntax) and content (semantics), the message definitions on the one hand, additionally an interchange format and management rules, the encoding rules for HL7 message instances on the other hand (see Figure1). The objective of this document is to present encoding rules for HL7 Version 2.3.1 and 2.4 messages based on the Extensible Markup Language XML [rfXML] that could be used in environments where senders and receivers both understand XML.

Many XML encodings could serve as alternate messaging syntaxes for HL7 Version 2.x messages. This document primarily addresses the translation between standard encoded and XML encoded HL7 Version 2.x, describing the underlying rules and principles. XML Schema [rfXMLSchema] definitions are provided for all version 2.x messages types, including the corresponding data type descriptions necessary for this specification. Document Type Definitions (DTD) are also provided as an informative appendix. The algorithms used for this specification to derive the database excerpts and to create schemas and DTDs are also presented in the informative appendix.

This document is the normative successor of the informative document “HL7 Recommendation: Using XML as a Supplementary Messaging Syntax for HL7 Version 2.3.1 – HL7 XML Special Interest Group, Informative Document” as of February, 2000 [rfINFO]. The former document is replaced by this specification, at the moment this document is successfully balloted.
Parts were added to this document, in particular about the general message structures of the HL7 v2 message standard in order to gently support persons with low knowledge of HL7 v2.

Acknowledgements

This standard is the result of about two years of intense work through e-mail, telephone conferences and meeting discussions. Special thanks to Bob Dolin and Paul Biron, who wrote the Informative Document.

This work was made possible by Frank Oemig, Lloyd McKenzie, Vassil Peytchev, Ralf Schweiger, Joachim Dudeck, Wes Rishel.

Thanks also to all members of the XML Special Interest Group and the Control/Query technical Committee for their input during the development process.

Remarks for 1st Committee Ballot

For the 1st Committee Ballot, some remarks have to be made concerning further developments of this specification.

- The suggested group names as of section XX are only partially present in the schema definitions for now. This is because the abstract message definitions (and the HL7 Database) in the corresponding chapters have not been populated with group names. Thus, many group names are constructed algorithmically (sequence) in this specification, but will be changed as soon as they are available.
- CM data types are not consequently eliminated (some relics) and thus not handled properly in this specification.
- The relationship between the v2.xml XML representation and the relationship to the Version 3 XML ITS is not discussed yet.
- XML Schema definitions of some primitive data types should have additional constraints in order to allow a more restrictive conformance testing.
- There is no support of the choice indicator in segment definitions, because no choice indicators could be found in the database.

Contents

v2.xml XML encoding rules for HL7 v2 messages.........................................................................................................................1
Preface............................................................................................................................................................................................1
Acknowledgements ........................................................................................................................................................................2
Remarks for 1st Committee Ballot ........................................................................................................................................2
Contents ..........................................................................................................................................................................................2
1 Introduction......................................................................................................................................................................................4
  1.1 Background ........................................................................................................................................................................4
  1.2 Benefits from using XML as an alternative v2 interchange format..................................................................................4
  1.3 Overview of Normative HL7 Database................................................................................................................................5
  1.4 Scope for HL7 Version 2 ...................................................................................................................................................5
    1.5 Version 2 Message Definitions......................................................................................................................................5
      1.5.1 Version 2 Hierarchical Message Structure........................................................................................................5
      1.5.2 Abstract Message Syntax Definitions......................................................................................................................6
2 Specification......................................................................................................................................................................................7
  2.1 Introduction to the XML representation............................................................................................................................7
  2.2 A first example....................................................................................................................................................................7
  2.3 Messages identifications and trigger events.....................................................................................................................8
    2.3.1 Message structure IDs .................................................................................................................................................8
  2.4 Segments...............................................................................................................................................................................8
    2.4.1 Groups of segments ...................................................................................................................................................9
1 Introduction

1.1 Background

In 1993, The European Committee for Standardization (CEN) studied several syntaxes (including ASN.1, ASTM, EDIFACT, EUCLIDES, and ODA) for interchange formats in healthcare [rfCEN]. A subsequent report extended the CEN study to look at SGML [rfDolin1997]. By using the same methodology, example scenarios, healthcare data model, and evaluation metrics, the report presented a direct comparison of SGML with the other syntaxes studied by CEN, and found SGML to compare favorably.

In February, 1998 XML became a recommendation of the World Wide Web Consortium (W3C). XML was further tested as a messaging syntax for HL7 Version 2.x and Version 3 messages [rfDolin1998]. In 1999, Wes Rishel coordinated a 10-vendor HL7-XML interoperability demonstration at the annual HIMSS Conference. All vendors rated the demo a success.

In 1999, the XML SIG developed an informative document in cooperation with Control/Query TC “HL7 Recommendation: Using XML as a Supplementary Messaging Syntax for HL7 Version 2.3.1 – HL7 XML Special Interest Group, Informative Document” that was approved as an HL7 Informative Document on membership level in February, 2000.

In August 2000 at the HL7 Board Retreat meeting in Dresden (Germany), it was decided that XML will become the 2nd normative encoding for versions 2.3.1 and 2.4 and future 2.x versions, i.e., the XML syntax that will be submitted for ANSI approval and that has the same status as the traditional syntax. An additional reason was to have XML a normative syntax for the Claims Attachment messages, which are specific to v2.3.1 by now.

1.2 Benefits from using XML as an alternative v2 interchange format

There are benefits using XML as an interchange format. Each HL7 application has coded its own parser (read) and generator (write) to process traditional (“vertical bar” encoded) HL7 messages with a certain impact on development and maintenance costs. A limited number of (expensive) tools are available.

Findings have shown that XML can serve as an implementable message specification not only for Version 3 but also for HL7 Version 2.3.1 and 2.4 messages and that the ability to explicitly represent an HL7 requirement in XML confers the ability to validate that requirement with an XML parser. Many “off-the-shelf” XML tools are available (freeware and commercial) such as parsers, transformation applications and instance viewers, which can perform much of the validation of message/document instances, so that applications don't have to. Trained personnel is much easier to find than experts familiar with vertical bar encoding rules.

Finally, an XML syntax for v2.x messages will also help vendors and providers transition from HL7 Version 2 family of standards to Version 3 by encouraging the early retooling of applications to support XML interfaces.

1.3 Overview of Normative HL7 Database

Underlying the HL7 Standard is a normative Microsoft Access database (the "HL7 Database") that contains the official definitions of events, messages, segments, fields, data types, data type components, tables, and table values.

This database arose as the German HL7 user group undertook careful analysis of the standard. They became aware that the chapters of the standard had been developed by different groups, and that there had been no distinct rules or guidelines for the development of various parts of the standard. They therefore defined a comprehensive database of the HL7 Standard (including Version 2.1 through Version 2.4 for now) to allow consistency checks of items and to support the application of the standard by the user.
Within the HL7 Database, all data added is checked for its consistency. Referential integrity among relations assures this consistency. The side effect of referential integrity is to modify the data from the standard documents because the standard is defined in the form of a document but not in the form of a database.

While developing the analytic object model for the definition of the comprehensive HL7 Database, the German HL7 user group became aware that two problems are not handled satisfactorily in the standard:

- the relationship between message types, event types, and the structure of a message;
- the relationship between fields, data types, data type components, and tables.

The XML representation of HL7 messages presented here is algorithmically derived directly from the HL7 Database. Therefore, ambiguities or errors in the standard are reflected “as is” in the XML encoding. Fixing any such errors in the XML will require making appropriate modifications to the HL7 Database. The procedures for deriving the schemas and DTDs are described in the informative appendix.

Further details of the HL7 Database as well as known problems encountered in the construction of the database have been documented by Frank Oemig et. al. (rfOemig1996), see also [rfOemig]).

1.4 Scope for HL7 Version 2

This specification presents XML encoding rules for HL7 Version 2.3.1 and 2.4 messages. Former Version of the HL7 Version 2 family of message standards are explicitly not covered, because constructs needed in this specification are not present in versions prior to v2.3.1. Therefore there is no XML encoding support for Versions prior to v2.3.1.

Versions after v2.4 are also not covered by this document, but will be added as soon as these versions are successfully balloted as an official HL7 standard.

1.5 Version 2 Message Definitions

1.5.1 Version 2 Hierarchical Message Structure

A specific HL7 version 2.x message is a hierarchical structure and is defined by a trigger, representing a real world event. A message is the atomic unit of data transferred between systems and is comprised of a group of segments in a defined sequence. Messages begin with the Message Header Segment MSH and are identified by the message type and the initiating event. A three-character code contained within each message identifies its type. For example the ADT message type is used to transmit portions of a patient’s Patient Administration (ADT) data from one system to another.

HL7 defines the content of the message as an abstract set of data elements contained in data segments. Segments are ordered sequences of fields and can be declared as required, optional or repeatable. Each segment begins with a three-character literal value that identifies it within a message (segment identifier). For example, the ADT message may contain the following segments: Message Header (MSH), Event Type (EVN), Patient ID (PID), and Patient Visit (PV1).

The semantic content of a message is transferred in the fields of the segment. Fields can be of variable length. Fields contents can be required or optional, individual fields may be repeated. Individual data fields are found in the message by their position within their associated segments. Multi-component fields are used for further subdivision of a field and facilitate the transmission of locally related semantic contents.

For each field or field component, a data type is defined. Simple data types are f. i. string of characters, time stamp etc. Complex data types comprises of two or more components. Examples are the CE data type (coded elements) which components are “coded value”, “code designator” and “code system”, or family name, which comprises of several sub-components in order to express the various parts of a person’s name. There are a number of data types defined by HL7.
All message types, trigger event codes, and segment ID codes beginning with the letter Z are reserved for locally-defined messages. No such codes will be defined within the HL7 Standard.

1.5.2 Abstract Message Syntax Definitions

Each message is defined in special notation that lists the segment IDs in the order they would appear in the message (see Figure 2). Braces, { ... }, indicate one or more repetitions of the enclosed group of segments. Of course, the group may contain only a single segment. Brackets, [ ... ], show that the enclosed group of segments is optional. If a group of segments is optional and may repeat it should be enclosed in brackets and braces, { [ ... ] }. Note that {[...] and {[...]} are equivalent.

Groups with more than a single segment are handled in a special way in this specification (see section 2.4.1), because they are named.

![Figure 2: Abstract message syntax definition for message type ADT_A01 (drawn from chapter 3 in [rfHL7v24])](image)

The brackets and braces in the Abstract Message Syntax relate to XML occurrence indicators as shown in the following:

<table>
<thead>
<tr>
<th>HL7 Abstract Message Syntax</th>
<th>Equivalent Cardinality in XML (minOccurs .. maxOccurs)</th>
<th>Equivalent XML DTD Occurrence Indicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>0 .. 1</td>
<td>?</td>
</tr>
<tr>
<td>{ }</td>
<td>1 .. unbounded</td>
<td>+</td>
</tr>
<tr>
<td>{[]} = [{ []}</td>
<td>0 .. unbounded</td>
<td>*</td>
</tr>
<tr>
<td>no bracket or brace</td>
<td>1 .. 1</td>
<td>no occurrence indicator (one exactly)</td>
</tr>
</tbody>
</table>

Table 1: Abstract Message Syntax Notations of Cardinality and their corresponding Schema and DTD Occurrence Specifications
2 Specification

2.1 Introduction to the XML representation

The XML representation presented here represents HL7 message structures as XML elements. Message structures contain segments, also represented as XML elements. Segments contain fields, again represented as XML elements. A field's data type is stored as a fixed attribute in the field's attribute list, while a field's content model contains the data type components. Other fixed attributes are used to expand abbreviations and indicate HL7 Table value restrictions.

2.2 A first example

Here a simple message in the syntax of the standard encoding rules can be seen:

| MSH|^~\&|LAB|767543|ADT|767543|199003141304-0500||ACK^^ACK|XX3657|P|2.4
| MSA|AR|Z9380|UNKNOWN COUNTY CODE
| ERR|PID^1^16^X3L

Here is the same message in the syntax of the recommended XML encoding rules:

```xml
<ACK>
  <MSH>
    <MSH.1>|</MSH.1>
    <MSH.2>^~\&</MSH.2>
    <MSH.3>LAB</MSH.3>
    <hd.1>767543</hd.1>
    <MSH.4>ADT</MSH.4>
    <MSH.5>199003141304-0500</MSH.5>
    <CM_MSG.1>ACK</CM_MSG.1>
    <CM_MSG.3>ACK_ACK</CM_MSG.3>
  </MSH>
  <MSH.10>XX3657</MSH.10>
  <PT.1>P</PT.1>
  <VID.1>2.4</VID.1>
  <PT.1>P</PT.1>
</MSH>

<MSA>
  <MSA.1>AR</MSA.1>
  <MSA.2>Z9380</MSA.2>
  <MSA.3>UNKNOWN COUNTY CODE</MSA.3>
</MSA>

<ERR>
  <ERR.1>
    <CM_ELD.1>PID</CM_ELD.1>
    <CM_ELD.2>1</CM_ELD.2>
    <CM_ELD.3>16</CM_ELD.3>
    <CM_ELD.4>X3</CM_ELD.4>
  </ERR.1>
</ERR>
</ACK>
```
As is always the case with XML when processed with a validating processor, the extra whitespace between elements (provided to make the message easier for people to read) can be removed in actual message instances, resulting in shorter messages in situations when overall message length is a factor.

The next section describes the stepwise creation of the XML representation.

2.3 Messages identifications and trigger events

A key position plays the message structure ID that is defined in the abstract message definition of a message and also given in the MSH-9 field of the message header segment. This field contains the message type, trigger event, and the message structure ID for the message.

- The first component is the message type code containing values such as ACK, ADT, ORM, ORU etc.
- The second component is the trigger event code with values like A01, O01, R01 etc. Refer to the HL7 standard document for complete listing.
- The third component is the abstract message structure ID defined by HL7 Table 0354 – Message structure.

2.3.1 Message structure IDs

The v2.xml schemas are based on the described message structure ID – a concept introduced in version 2.3.1. Looking at message definitions in 2.3.1 and later, the abstract message definition (see example a in Figure 3) and the MSH-9 field (see example b in Figure 3) contain the message type, trigger event, and the message structure ID for the message, as an example ADT^A04^ADT_A01. This indicates that the ADT message with trigger event A04 has the message structure ID ADT_A01. All messages with that structure ID are structurally the same, except for the semantics of the event (A04 in the example case). In detail, message structure code ADT_A01 describes the single abstract message structure used by the trigger events A01, A04, A05, A08, A13, A14, A28 and A31.

As a consequence encoding an A04 message, which has the ADT_A01 message structure (i.e. sequence of segments), means to use the schema definition for the ADT_A01 message. The standard documents contain tables where the message structure IDs are listed (see also section 3.1.1 for information about messages with equal structures).

![Figure 3: Example for Abstract Message Syntax (fragment) and Message Header Segment](image)

Messages use the message structure ID as a root element for the XML instance documents. As an example the corresponding XML message fragment is shown below.

```xml
<ADT_A01>
  ...(segment elements)
</ADT_A01>
```

The element `<ADT_A01>` carries the segment elements (see following section) as child elements.

2.4 Segments

Message structures contain segments, also represented as XML elements. Segments are ordered sequences of fields. Each segment begins with a three-character literal value that identifies it within a
message (segment identifier). The v2.xml schema definition uses the segment identifier as XML element names. A MSH segment for example has <MSH> as an XML element name, a PID segment <PID> etc.

Considering the ADT_A04 example above, the corresponding XML message fragment with is shown below.

```xml
<ADT_A01>
  <MSH>
    ...(MSH field elements)
  </MSH>
  <EVN>
    ...(EVN field elements)
  </EVN>
  <PID>
    ...(PID field elements)
  </PID>
  ...(other segment elements)
</ADT_A01>
```

The element <MSH> for example carries the corresponding field elements (see following section) as child elements.

### 2.4.1 Groups of segments

Some segments are grouped by braces {...} or brackets [ ... ] to denote repetitions or optionality of the segment(s). If a group of segments is optional and may repeat it is enclosed in brackets and braces, {...}, where {...} and {[...]} are equivalent.

Groups containing more than a single segment are handled in a special way in this specification. For example in Figure 4, a group is denoted by [[ PR1 {[ ROL } ] ]]. This group is named “procedure” (see Column “Group Control” in Figure 4). Another example is the [[ IN1 [ IN2 ] {[ IN3 }] {[ ROL } ] ] group which is named “insurance”. These names also appear in the v2.xml schema definitions of the corresponding messages (and thus also in an XML instance documents containing messages of that type).

<table>
<thead>
<tr>
<th>ADT^A01^ADT A01</th>
<th>ADT Message</th>
<th>Group Control</th>
<th>Comment</th>
<th>Support Indicator</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSH</td>
<td>Message Header</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>EVN</td>
<td>Event Type</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>PID</td>
<td>Patient Identification</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>PR1, [ ROL ]</td>
<td>Procedures</td>
<td>Procedure</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>IN1, IN2, IN3, [ ROL ]</td>
<td>Insurance</td>
<td>Insurance</td>
<td></td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 4: Abstract message syntax (fragment) with named groups of segments.

During discussions of the informative document it was found useful to deepen the XML element hierarchy by adding grouping elements where defined in the abstract message syntax definition and showing more than one segment in a group. There was no explicit way to express these groups in the traditional v2 “vertical bar” encoding of messages.

About 400 different groups of that kind could be identified in the standard. Some of the groups have identical content concerning segment sequence, some of the contained segments however have
different cardinalities. As an example the group “insurance” could be found in ADR_A19, ADR_A01, ADR_A05, ADR_A06 etc. but the single segments IN1, IN2 etc. have different cardinalities within the groups. As a consequence, the v2.xml XML schema defines named group prefixed with the corresponding message structure ID where this group belong to in order to make group element identifiers unique regarding their content.

Considering the ADT_A04 example above, the corresponding XML message fragment with groups is shown below.

```xml
<ADT_A01>
  ...
  <ADT_A01.GROUP.PROCEDURE>
    <PR1>
      ...
    </PR1>
    <ROL>
      ...
    </ROL>
  </ADT_A01.GROUP.PROCEDURE>
  ...
  <ADT_A01.GROUP.INSURANCE>
    <IN1>
      ...
    </IN1>
    <IN2>
      ...
    </IN2>
    <IN3>
      ...
    </IN3>
  </ADT_A01.GROUP.INSURANCE>
  ...
</ADT_A01>
```

2.4.2 Lists

In the informative document [rfINFO] list elements (LST) were also defined in addition to groups in order to allow the wire format to reflect the inherent abstract hierarchy of segments. While groups of segments are still supported (see section above), this specification makes no longer use of LST-elements. This simplifies the resulting Schemas and DTDs and leads to more simple XML instances of v2.xml encoded messages.

2.5 Fields

The semantic content of a message is transferred in the fields of the segment. Fields contents can be required or optional, individual fields may be repeated. Individual data fields are found in the message by their position within their associated segments and are described in segment tables (see Figure 5 as an example).

<table>
<thead>
<tr>
<th>SEQ</th>
<th>LEN</th>
<th>DT</th>
<th>OPT</th>
<th>RP/#</th>
<th>TBL#</th>
<th>ITEM#</th>
<th>ELEMENT NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>ID</td>
<td>B</td>
<td>0003</td>
<td>000099</td>
<td>Event Type Code</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>26</td>
<td>TS</td>
<td>R</td>
<td>00100</td>
<td>00100</td>
<td>Recorded Date/Time</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>26</td>
<td>TS</td>
<td>O</td>
<td>00101</td>
<td>00101</td>
<td>Date/Time Planned Event</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>IS</td>
<td>O</td>
<td>0062</td>
<td>00102</td>
<td>Event Reason Code</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>250</td>
<td>XCN</td>
<td>O</td>
<td>0188</td>
<td>00103</td>
<td>Operator ID</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>26</td>
<td>TS</td>
<td>O</td>
<td>01278</td>
<td>01534</td>
<td>Event Occurred</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>180</td>
<td>HD</td>
<td>O</td>
<td></td>
<td></td>
<td>Event Facility</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: Segment attribute table of the EVN segment (fragment, drawn from chapter 3 in [rfHL7v24])
Multi-component fields are used for further subdivision of a field and facilitate the transmission of locally related semantic contents.

In the v2.xml specification, individual fields are represented by the three-character literal segment ID of the corresponding segment plus their individual position within the segment (sequence). The first field (Event Type Code) in segment EVN for example is named EVN.1, the second EVN.2 etc. An example of an EVN segment, traditionally encoded and v2.xml encoded is shown below.

In the traditional sequence oriented approach, empty fields (containing no data) are denoted as two vertical bars “||” in sequence to express the empty contents. This is essential in sequence oriented approaches. In v2.xml an element with no contents simply can be omitted. In the example above there is no information for EVN.5, thus the element <EVN.5> is omitted in the corresponding XML instance.

The content model of each field is a reference to the field’s data type. Fixed attributes are used to express the component’s item number, table reference, long name, and data type.

The example below shows the attribute group (schema definition) of the EVN.2 field element defining an attribute group that is associated with the corresponding EVN.2 element.

Assuming that sender and receiver uses the same schemas, the actual XML instance that is sent between them is significantly shorter, because the fixed attributes need not to be sent over the wire. If a receiver gets XML instance that is validated against the schema and that contains an EVN.2 element, the receiving parser can make use of the fixed attributes that are “merged” with the conveyed contents during the validation process. The following example shows on the left hand, what is actually sent over the wire, and on the right hand what a validating parser gets after the validation process against the standard v2.xml schema definitions. In addition to the EVN.2 example, an OBX segment fragment is shown also.

<table>
<thead>
<tr>
<th>this fragment is sent over the wire:</th>
<th>this is the segment/field after validated by the receiver against the v2.xml schema:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;EVN.2&gt;</code>199901061000&lt;/EVN.2&gt;</td>
<td><code>&lt;EVN.2 Item=&quot;100&quot; Table=&quot;0&quot; LongName=&quot;Recorded Date/Time&quot; Type=&quot;TS&quot;&gt;199901061000&lt;/EVN.2&gt;</code></td>
</tr>
</tbody>
</table>
2.6 Data types

For each field or field component, a data type is defined. Some data types are primitive, i.e., they have no components. Composite data types are comprised of data type components, which, like fields, have a data type of their own and a long name. Some data type components also specify an HL7 Table that contains enumerated values for use in the component.

2.6.1 Primitive data types

Some data types are primitive, in which case they have no components. Simple data types are, e.g., string of characters, time stamp etc.

A field for which a primitive data type is defined simply contains the information without additional nesting or hierarchy. As an example the 2nd field of the EVN segment (see Figure 5) is of type TS, which is a primitive data type. The corresponding XML instance fragment looks like the following example:

```
<EVN.2>199901061000</EVN.2>
```

The v2.xml schema definitions define all primitive data types as “string” (XML schema), in DTDs #PCDATA is used.

2.6.2 Composite data types

Complex data types comprises of two or more components. As an example, consider the CE data type (coded elements) which components are “coded value”, “code designator” and “code system” etc. The standard defines the individual components of the composite data types in chapter 2 (see example below, alternatively a table is presented (Figure 6, not presented in the standard, but used for later reference in this specification).

Components: <identifier (ST)> ^ <text (ST)> ^ <name of coding system (IS)> ^ <alternate identifier (ST)> ^ <alternate text (ST)> ^ <name of alternate coding system (IS)>

<table>
<thead>
<tr>
<th>Data structure</th>
<th>seq</th>
<th>component description</th>
<th>Table ID</th>
<th>Data type code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE</td>
<td>1</td>
<td>identifier (ID)</td>
<td>0</td>
<td>ID</td>
</tr>
<tr>
<td>CE</td>
<td>2</td>
<td>text</td>
<td>0</td>
<td>ST</td>
</tr>
</tbody>
</table>
Analogous to field components, data types components are modeled by specifying the data structure name plus their individual position within the data type component (sequence). As an example, the first component of data type CE is defined as CE.1, the second as CE.2 and so on. The following example shows a CE data type encoded traditionally ("vertical bar") and as v2.xml fragment.

![Example of CE data type components drawn from [rfHL7v24] (above) and a table representation of the same definition](image)

Also, empty components may be omitted in the v2.xml encoding, whereas empty components in the traditional encoding must specify an empty component by two component delimiters "^^" in sequence in order to preserve sequence.

Where a field has a data type with multiple components but only a single component is populated with information, the corresponding data type element of the component may not be omitted. For example, if a field of type CE only carries information in the first component only (i.e. the identifier of a coded element) the correct v2.xml encoding is shown as in the following example with an OBX.3 field:

![Incorrect: Correct:](image)

Data type components of composite data types are modeled similarly to fields. The content model of each component contains reference to the component's data type. Fixed attributes are used to express the component’s data type, long name, and table, as shown here for CE.1:

![Data type components of composite data types](image)
2.6.3 Wildcard

In the HL7 Database, some data types (components) specify “WILDCARD” or “varies” in order to express an undefined type (data of any type can be specified). This data type is modeled as type="anyType" in XML Schema and as (ANY) in DTD definitions.

2.6.4 CM data types

A special data type, the CM data type, was used in HL7 v2.3.1 (and prior to that version) to express that the explicit data type of the content is undefined (i.e. a type="anyType" in XML Schema or (ANY) in DTD definitions). This data type was eliminated in v2.4 in order to allow more restricted conformance testing. All data types CM were renamed and refined to specific data types CM_name to represent a unique content in the context they are used.

2.7 Design considerations

As noted above, there are many possible XML representations of HL7 messages. This section describes those factors considered in deciding on the particular approach presented in this specification.

2.7.1 XML Schema Optimization

XML schema optimization means balancing functional, technical, and practical requirements. Some metrics are fairly straightforward to quantify (e.g. message length), while others are less so. There is a risk that the easily quantifiable measurements will assume significance out of proportion to other metrics. All relevant metrics must be factored together in the determination of the optimal XML representation.

2.7.1.1 Message Length

Message length minimization techniques are employed to decrease the total number of characters (including data and/or markup) comprising a message. The optimal techniques used to minimize SGML messages are not necessarily the same as those best suited to minimize XML messages. Techniques used here, common to both SGML and XML, include the use of abbreviations and the assumption that a slot not sent represents a null value. In some cases modeling components as XML attributes as opposed to elements results in further minimization. This specification represents HL7 message structures, segments, and fields as XML elements. A field's data type is represented as a fixed attribute, while data type components are represented as XML elements. Full SGML provides even greater minimization capacity with the use of SHORTTAG, OMITTAG, and SHORTREF techniques, resulting in very small messages that are not valid XML, and are therefore not employed here.

The greater the percentage of data characters (as opposed to markup characters) in an average message, the less important any additional overhead imposed by changing from the standard HL7 encoding rules to XML becomes. Data from the Duke HL7 production environment suggests that on average, data characters comprise about 70% of overall message length. (Data from Duke courtesy of Al Stone, and posted to the HL7 SGML/XML SIG List Server 1/15/98 and 1/16/98.) As a result, the XML encoding recommended here will result in messages that are approximately ten times longer, although this estimate has yet to be subjected to rigorous testing nor is officially published.

2.7.1.2 Structural Complexity

Krueger [rKrueger] describes the use of “structural complexity” as a metric to analyze HL7 messages. “It would be nice to be able to estimate or compare the time needed by human users to understand or implement different messages or the time needed for a parsing program to analyze different messages.” The exact determinates of structural complexity were outside the scope of Krueger's work, although he comments that “empirical investigations must be carried out to monitor the effort users will take to understand and implement different HL7 messages”. Potential components of this metric are listed below. In some cases, the metric will be the time and/or space complexity required to carry out the
functions. We agree with Krueger that “it does not make any sense to expect absolute results. However, relative (i.e. comparable) results could also be a valuable source of information.”

- Message Creation: Encompasses the processing requirements to create a message.
- Message Augmentation: Augmentation might include changing the format of a field or data type component or transforming the message from one syntax to another.
- Message Debugging: Determine why an application is generating an HL7-invalid message.
- Message Filtering: Filtering might include sending only a subset of the message to a particular message receiver.
- Message Routing: Routing includes extracting from the message what is necessary to determine where to send it.
- Message Parsing: Parsing can include message validation and extraction of field values and data type components.

2.7.1.3 Localization

The HL7 Standard describes the responsibilities for parties sending and receiving HL7 messages. These responsibilities enable exchange of messages that contain localizations (or local variations or z-segments). Consequent to these requirements, an XML representation needs to fulfill the following design considerations:

- Senders can introduce, in a standardized manner, local variations into standard HL7 messages where necessary. The expression of local variations is formalized such that their location in a message can be algorithmically determined by receivers. This formalization expresses localizations as changes to the standard schemas within the internal subset of a transmitted message instance.
- Receivers can use well-formed XML processors or validating XML processors. Receivers using validating processors do not have to fall back to using a non-validating processor in those cases when the sender includes localized content in their messages.

The v2.xml schemas and DTDs are crafted to fulfill these requirements.

2.7.1.3.1 DTD Localization

The sender includes differences from the standard DTD in the internal subset of a message instance. This is enabled by expressing all content models as parameter entities, which can then be redefined in the internal subset. For example, rather than this:

```xml
<!ELEMENT FOO (FOO.1?, FOO.2.LST?, FOO.3.LST?, FOO.4?)>
```

an equivalent content model like this is stated:

```xml
<!ENTITY % FOO.CONTENT "(FOO.1?, FOO.2.LST?, FOO.3.LST?, FOO.4?)">  
<!ENTITY FOO %FOO.CONTENT;>
```

so that the content of FOO can be changed in the internal subset like this:

```xml
<!DOCTYPE ORU_RO1 SYSTEM "hl7_v24.dtd" [
<!ENTITY % FOOCONTENT "(FOO.1?,FOO.2.LST?,FOO.3.LST?,FOO.4?, FOO.5?)">  
<!ELEMENT FOO.5 (#PCDATA)> ]]>
```

Receivers using non-validating XML processors can ignore the entire DOCTYPE declaration. Senders are not required to create or provide an explicit representation of the transformation from the localized DTD to an HL7 standard DTD.

2.7.1.3.2 Schema Localization

The schemas provided in this specification can easily be localized by the same approach described above. This is done by restrictions or extensions to the existing definitions. Like in the DTD definitions, the method to declare all content models also as schema-“ENTITIES”, i.e. as group or attributeGroup, can also be found in the schema definitions.
### 2.7.1.4 "Looseness" of a DTD

XML is a formal grammar that can be used to encode HL7 business rules. When an XML processor validates that a message is valid per its schema or DTD, it is also validating that a message is conformant to those HL7 rules that are explicitly represented in the XML schema or DTD.

Some HL7 rules are easy to explicitly represent within an XML DTD as well as in an XML schema, such as the optionality and repetition of a field within a segment. However, with XML schema we can express more HL7 rules explicitly, define more constraints as that is possible with a corresponding DTD.

Representing such rules within a DTD, while possible, may conflict with other design considerations. Therefore, determining the "looseness" of the DTD, or the degree to which HL7 business rules are explicitly represented in the DTD, is itself a design consideration. The same design consideration applies to XML schema.

You can carry HL7-valid messages in the constructs defined by this specification, but you can also carry a lot of HL7-invalid messages. An XML processor can't do much to help validate that a message received is a valid HL7 message. The decision in the XML representation presented here is to capture as many HL7 business rules as reasonably possible, both in terms of DTDs and XML schemas. This includes enabling a validating parser to verify the optionality, repetition, and ordering of segments within messages and fields within segments; and the correct use of data types and their components within fields. Easing the burden on the application with regard to structural validity (e.g., are all the pieces in the proper place) is itself a big win, despite the fact that the application will still have to perform semantic validation (e.g., is that code really a valid SNOMED code).

### 2.8 Processing rules for v2.xml messages

#### 2.8.1 XML Application processing rules

The original and enhanced processing rules described in chapter 2 of the v2.x standard are not affected by this specification. However, concerning the exchange of XML message between sender and receiver, additional assumptions are made in terms of "well-formed" and "valid" XML documents.

The *sender* of a v2.xml XML message is required to create both well-formed and valid documents. The validation has to be made against the corresponding XML schema definitions (see section 3.1.2).

The *receiver* who accepts a v2.xml XML message is required to check well-formedness of the XML instance. He may (but is not required to) validate the message against the schemas.

#### 2.8.2 Message continuation

Message continuation as described in chapter 2 of the v2.x standard is not supported in this v2.xml specification. It is assumed that XML aware systems, for which this specification is written, are able to accept stream character messages of an arbitrary length.

#### 2.8.3 Message delimiters

In constructing a traditionally encoded v2 message, certain special characters are used. They are the segment terminator, the field separator, the component separator, subcomponent separator, repetition separator, and escape character. The segment terminator is always a carriage return (in ASCII, a hex 0D). The other delimiters are defined in the MSH segment, with the field delimiter in the 4th character position, and the other delimiters occurring as in the field called Encoding Characters, which is the first field after the segment ID. The delimiter values used in the MSH segment are the delimiter values used throughout the entire message. In the absence of other considerations, HL7 recommends the suggested delimiter values.

At any given site, the subset of the possible delimiters may be limited by negotiations between applications. This implies that the receiving applications will use the agreed upon delimiters, as they appear in the Message Header segment (MSH), to parse the message.
In the v2.xml encoding the message delimiter characters are contained in the MSH.1 and MSH.2 element of the MSG segment, as well. Because the message delimiter characters are meaningless in the v2.xml encoding, they are constant as shown in the example fragment of the MSH segment. Please note, that the special character “&” must be escaped in order to be included in an XML instance document (see also section 2.8.6)

```
<ACK>
  <MSH>
    <MSH.1> | </MSH.1>
    <MSH.2> ^~\& </MSH.2>
  ...
```

2.8.4 Escape characters used in v2 data types

When a field of data type TX, FT, or CF is being encoded, the escape character may be used to signal certain special characteristics of portions of the text field. The escape character is whatever display ASCII character is specified in the escape character component of MSH-2-encoding characters.

The use of these escape sequences is not altered by this specification. It is not recommended to substitute the escape sequences by HTML tags, for example (i.e. a \H\ and \N\ escape sequence in an field of type FT, which starts a highlighted character presentation, may not be replaced by the corresponding HTML-tags <B> </B> etc.).

The message content is thus not affected by this specification. The only exception is the use of special characters in DTDs, schemas and XML instance documents as described in section 2.8.6.

2.8.5 Message building rules

The message building rules remains the same as described in chapter 2 of the Standards. However there are some exceptions if the v2.xml encoding is used.

- Segment, field, component and subcomponent separators are not used but represented by individual elements.
- If a value for a field is not present, the corresponding element can be omitted (if not required by the schema definition).
- For groups of segments defined in the v2.xml specification, additional group elements are introduced. (In the standard encoding, groups are not explicitly encoded).

A receiver who accepts a v2.xml XML message is required to check well-formedness of the XML instance and may (but is not required to) validate the message against the schemas. Like described in chapter 2 of the standard,

- he may ignore segments, fields, components, subcomponents, and extra repetitions of a field that are present but were not expected,
- treat segments that were expected but are not present as consisting entirely of fields that are not present,
- treat fields and components that are expected but were not included in a segment as not present,

but in terms of validating against the v2.xml schema definition, the cardinality of the components is of course determined by the v2.xml schema.

2.8.6 Special characters in Schemas/DTDs

Certain characters within the HL7 Database must be “escaped” before inclusion in a schema. The ampersand is a reserved XML metacharacter.
Where an ampersand occurs in the long name of a field, it is converted to an XML entity representation “&amp;.” An example is “Critical Range for Ordinal & Continuous Obs” that becomes “Critical Range for Ordinal & Continuous Obs”.

Because the Schema/DTD wraps the value of attribute LongName in single quotes, when a single quote occurs in the long name of a field, it is converted to an XML entity representation “&apos;”, e.g. “Contact's Tel. Number” becomes “Contact&apos;s Tel. Number”).

Because data types become XML element names (as described in the section that follows), internal plus-signs (“+”) are not allowed. There is a data type in the HL7 Database (CE_TABS_136+262+263) that contains internal plus-signs (“+”). These are converted to underscores (“_”), e.g. “CE_TABS_136+262+263” becomes “CE_TABS_136_262_263”.

The same rules apply to XML instance documents.

2.9 Translating Between Standard Encoding And XML Encoding

In environments where not all senders and receivers understand this XML encoding it may be necessary to translate instance messages between the standard encoding and this XML encoding and vice versa. This recommendation does not require that any such translations be supported nor does it prescribe how such transformations should be performed in environments where they are supported.

Because of several important differences between the standard encoding and this XML encoding, translations between the two encodings is not straightforward although it is not hard. The issues described in section 2.8 need to be taken into account when performing the translations.

2.10 Outstanding Problems

2.10.1 Database

As noted above, the HL7 Database is used for all schema/DTD creation of this specification. Ambiguities or errors in the standard are reflected “as is” in the XML encoding. Fixing any such errors in the XML will require making appropriate modifications to the HL7 Database.

2.10.2 Choices of segments

A choice of one segment from a group of segments is indicated by using angle brackets to delimit the group and vertical bar delimiters between the several segments.

Example: The ORM^O01, as described in chapter 4 of the standard, allows a choice of order detail segments. The choice would be represented as follows:

```xml
<OBR|RQD|RQ1|RXO|ODS|ODT>
```

This specification does not cover choices because no choice definitions could be found in the HL7 database for version 2.3.1 and v2.4 while creating this specification.

2.10.3 Conformance with HL7 Version 3 XML Representation

The XML messaging syntax in this document is not the same as that which is being proposed for Version 3.0 [rfHL7v3ITS]. The Version 3 XML project is still a moving target, and consensus on XML representation has not yet been reached. Given the different philosophies underlying Versions 2.x and 3.0, there will probably be some differences.
3 Appendix

3.1 Normative Appendix

3.1.1 List of messages with equal message structures

As mentioned the v2.xml schemas are based on the message structure ID – a concept introduced in version 2.3.1. Looking at message definitions in 2.3.1 and later, the abstract message definition and the MSH-9 field contain the message type, trigger event, and the message structure ID for the message, as an example ADT^A04^ADT_A01. This indicates that the ADT message with trigger event A04 has the message structure ID ADT_A01. All messages with that structure ID are structurally the same, except for the semantics of the event (A04 in the example case). In detail, message structure code ADT_A01 describes the single abstract message structure used by the trigger events A01, A04, A05, A08, A13, A14, A28 and A31.

As a consequence encoding a A04 message, which has the ADT_A01 message structure, means to use the schema definition for the ADT_A01 message. The standard documents contain tables where the message structure IDs.

<table>
<thead>
<tr>
<th>Version</th>
<th>Chapter</th>
<th>HL7 Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>v2.3.1</td>
<td>2.24.1.9</td>
<td>Message structure table 0354</td>
</tr>
<tr>
<td>v2.4</td>
<td>2.17.3</td>
<td>Message structure table 0354</td>
</tr>
</tbody>
</table>

3.1.2 List of Schemas and DTDs

This specification provides two sets of schema definition files: XML Schema definitions (xsd), and Document Type Definitions (dtd), as shown by the following table, where version means “v231” for Version 2.3.1 and “v24” for Version 2.4. There is a set for each HL7 version (v2.3.1 and v2.4)

<table>
<thead>
<tr>
<th>Schema (in directory xsd)</th>
<th>DTD (in directory dtd)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hl7_version.xsd</td>
<td>hl7_version.dtd</td>
<td>single schema that contains all HL7 v2.xml definitions for version</td>
</tr>
<tr>
<td>messages_version.xsd</td>
<td>messages_version.dtd</td>
<td>schema for all message definition</td>
</tr>
<tr>
<td>segments_version.xsd</td>
<td>segments_version.dtd</td>
<td>schema for all segment definition</td>
</tr>
<tr>
<td>fields_version.xsd</td>
<td>fields_version.dtd</td>
<td>schema for all field definition</td>
</tr>
<tr>
<td>datatypes_version.xsd</td>
<td>datatypes_version.dtd</td>
<td>schema for all data type definition</td>
</tr>
</tbody>
</table>

The single schema contains all HL7 v2.xml definitions. This file is logically broken up into a schema containing all message declarations, a schema containing all segment declarations, a schema containing all field declarations, and a schema containing all data type declarations.

Message instances are validated against the single XML schema "hl7_version.xsd" or against the DTD "hl7_version.dtd".

3.2 Informative Appendix

3.2.1 Extracting Extracting Subsets Of The Normative HL7 Database

3.2.1.1 Messages and their segments

3.2.1.1.1 HL7 Database tables used
The following HL7 Database tables are used in the creation of the message schemas. (Only those fields being queried are shown. The field names and their descriptions are taken verbatim from the HL7 Database.)

### EventMessageTypeSegments

**Primary Key**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>* event_code</td>
<td>Text-3</td>
<td>Event-Code</td>
</tr>
<tr>
<td>* hl7_version</td>
<td>Text-8</td>
<td>version number</td>
</tr>
<tr>
<td>* message_type</td>
<td>Text-3</td>
<td>Type of this Message</td>
</tr>
<tr>
<td>* lfd_nr</td>
<td>Integer</td>
<td>consecutive increasing number used for 1:n relation</td>
</tr>
<tr>
<td>seg_code</td>
<td>Text-3</td>
<td>Segment-Code</td>
</tr>
<tr>
<td>repetitional</td>
<td>Yes/No</td>
<td>Repetitional</td>
</tr>
<tr>
<td>optional</td>
<td>Yes/No</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### MsgStructIDSegments

**Primary Key**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>* message_structure</td>
<td>Text-7</td>
<td>Message Structure ID</td>
</tr>
<tr>
<td>* hl7_version</td>
<td>Text-8</td>
<td>version number</td>
</tr>
<tr>
<td>* lfd_nr*</td>
<td>Integer</td>
<td>consecutive increasing number used for 1:n relation</td>
</tr>
<tr>
<td>seg_code</td>
<td>Text-3</td>
<td>Segment-Code</td>
</tr>
<tr>
<td>repetitional</td>
<td>Yes/No</td>
<td>Repetitional</td>
</tr>
<tr>
<td>optional</td>
<td>Yes/No</td>
<td>Optional</td>
</tr>
</tbody>
</table>

*The field names and their descriptions are taken verbatim from the HL7 Database.

#### 3.2.1.1.2 SQL query

The following union query is used to gather together message structures from tables EventMessageTypeSegments and MsgStructIDSegments:

```sql
SELECT [message_type] + "_" + [event_code] AS MsgStruct, lfd_nr, seg_code, repetitional, optional
FROM EventMessageTypeSegments
WHERE hl7_version = "version"
UNION
SELECT message_structure, lfd_nr, seg_code, repetitional, optional
FROM MsgStructIDSegments
WHERE hl7_version = "version";
```

This creates a new table MsgStructUnion to hold the results of the union query. The structure of MsgStructUnion is shown here:

### MsgStructUnion

**Primary Key**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>* MsgStruct</td>
<td>Text-7</td>
<td>Message Structure ID</td>
</tr>
<tr>
<td>* lfd_nr</td>
<td>Integer</td>
<td>consecutive increasing number used for 1:n relation</td>
</tr>
<tr>
<td>seg_code</td>
<td>Text-3</td>
<td>Segment-Code</td>
</tr>
<tr>
<td>repetitional</td>
<td>Yes/No</td>
<td>Repetitional</td>
</tr>
<tr>
<td>optional</td>
<td>Yes/No</td>
<td>Optional</td>
</tr>
</tbody>
</table>
The results of the union query are copied into MsgStructUnion.

3.2.1.1.3 Select from MsgStructUnion those message structures reflected in Table 0354

Chapter 2 of HL7 v2.x "Control/Query" describes the message header (MSH) segment. Field 9 "Message Type" (MSH.9) contains the message type, trigger event, and the message structure ID for the message. The third component of MSH.9 is the abstract message structure code defined by HL7 Table 0354 – Message structure. From MsgStructUnion those message structures contained in Table 0354 where extracted using this SQL query:

```
SELECT MsgStruct, lfd_nr, seg_code, repetitional, optional
FROM TableValues INNER JOIN MsgStructUnion ON TableValues.table_value = MsgStructUnion.MsgStruct
WHERE (table_id=354) AND (hl7_version = "version")
ORDER BY MsgStruct, lfd_nr;
```

This resulting table is exported to messages.txt. This file serves as input for the transformation algorithms described in section 0.

3.2.1.2 Segments, fields, and data types

3.2.1.2.1 HL7 Database tables used

The following HL7 Database tables are used in the creation of the segments and fields schema definitions. (Only those fields being queried are shown. The field names and their descriptions are taken verbatim from the HL7 Database.)

<table>
<thead>
<tr>
<th>SegmentDataElements</th>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Key</td>
<td>* seg_code</td>
<td>Text-3</td>
<td>Name of the Segment</td>
</tr>
<tr>
<td></td>
<td>* hl7_version</td>
<td>Text-8</td>
<td>version number</td>
</tr>
<tr>
<td></td>
<td>* lfd_nr</td>
<td>Integer</td>
<td>Position within the segment</td>
</tr>
<tr>
<td></td>
<td>data_item</td>
<td>Long Integer</td>
<td>Data Element ID</td>
</tr>
<tr>
<td></td>
<td>req_opt</td>
<td>Text-5</td>
<td>required/optional/backward compatibility</td>
</tr>
<tr>
<td></td>
<td>repetitional</td>
<td>Text-1</td>
<td>Repetitional</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DataElements</th>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Key</td>
<td>* data_item</td>
<td>Long Integer</td>
<td>ID of the Data Element</td>
</tr>
<tr>
<td></td>
<td>* hl7_version</td>
<td>Text-8</td>
<td>HL7-Version</td>
</tr>
<tr>
<td></td>
<td>description</td>
<td>Text-78</td>
<td>Field description according to the standard documentation</td>
</tr>
<tr>
<td></td>
<td>data_structure</td>
<td>Text-20</td>
<td>Name of the Data Structure</td>
</tr>
<tr>
<td></td>
<td>table_id</td>
<td>Long Integer</td>
<td>ID assigned table</td>
</tr>
</tbody>
</table>

3.2.1.2.2 SQL query

The following SQL query extracts data from tables SegmentDataElements and DataElements:

```
SELECT seg_code, lfd_nr, SegmentDataElements.data_item, description, data_structure, req_opt, repetitional, table_id
FROM DataElements INNER JOIN SegmentDataElements ON (DataElements.hl7_version = SegmentDataElements.hl7_version) AND (DataElements.data_item = SegmentDataElements.data_item)
WHERE SegmentDataElements.hl7_version = "version"
ORDER BY seg_code, lfd_nr;
```
This resulting table is exported to fields.txt. This file serves as input for the transformation algorithms described in section 0.

### 3.2.1.3 Data types and their data type components

#### 3.2.1.3.1 HL7 Database tables used

The following HL7 Database tables are used in the creation of data types schema definitions. (Only those fields being queried are shown. The field names and their descriptions are taken verbatim from the HL7 Database.)

<table>
<thead>
<tr>
<th>DataStructures</th>
<th>Primary Key</th>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>data_structure</td>
<td>Text-20</td>
<td>logical data type</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>hl7_version</td>
<td>Text-8</td>
<td>version number</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DataStructureComponents</th>
<th>Primary Key</th>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>data_structure</td>
<td>Text</td>
<td>logical data type</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>hl7_version</td>
<td>Text</td>
<td>version number</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>lfd_nr</td>
<td>Text</td>
<td>consecutive increasing number used for 1:n relation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>comp_nr</td>
<td>Text</td>
<td>identifying number of the component</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table_id</td>
<td>Long Integer</td>
<td>Number of assigned table if different from component (overwrites table number of component)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Components</th>
<th>Primary Key</th>
<th>Field Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>comp_nr</td>
<td>Long Integer</td>
<td>Component Number (ID)</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>hl7_version</td>
<td>Text-8</td>
<td>Version of HL7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>description</td>
<td>Text-50</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td></td>
<td>table_id</td>
<td>Long Integer</td>
<td>reference to an assigned Table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>data_type_code</td>
<td>Text-3</td>
<td>Data type</td>
<td></td>
</tr>
</tbody>
</table>

#### 3.2.1.3.2 SQL query

The following SQL query extracts data from tables DataStructures, DataStructureComponents, and Components:

```sql
SELECT DataStructures.data_structure, lfd_nr, DataStructures.description, DataStructureComponents.table_id, Components.description, Components.table_id, Components.data_type_code
FROM DataStructures LEFT JOIN (DataStructureComponents LEFT JOIN Components
ON (DataStructureComponents.comp_nr = Components.comp_nr) AND
(DataStructureComponents.hl7_version = Components.hl7_version))
ON (DataStructures.hl7_version = DataStructureComponents.hl7_version) AND
(DataStructures.data_structure = DataStructureComponents.data_structure)
WHERE DataStructures.hl7_version = "version"
ORDER BY DataStructures.data_structure, lfd_nr;
```

This resulting table is exported to datatypes.txt. This file serves as input for the transformation algorithms (see section 3.2.2).
3.2.1.4 Summary of Additional Files

The following table summarizes the files that, along with this document, comprise the work products of this recommendation.

<table>
<thead>
<tr>
<th>HL7 Structure</th>
<th>Database excerpt</th>
<th>Resulting schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messages</td>
<td>messages.txt</td>
<td>schema for all message definition</td>
</tr>
<tr>
<td>Segments</td>
<td>fields.txt*</td>
<td>schema for all segment definition</td>
</tr>
<tr>
<td>Fields</td>
<td>fields.txt</td>
<td>schema for all field definition</td>
</tr>
<tr>
<td>Data types</td>
<td>datatypes.txt</td>
<td>schema for all data type definition</td>
</tr>
</tbody>
</table>

* fields.txt is used in the generation of both segments and fields schema definitions.

3.2.2 Algorithms

The mapping from HL7 Version 2.x into the XML specification v2.xml is a formal algorithm, driven from the HL7 Database described above. As such, ambiguities or errors in the Standard are reflected as is in the XML encoding. Fixing any such errors in the XML will require making appropriate modifications to the HL7 Database.

As described above, HL7 Database extracts (ASCII files, tab delimited) are created containing definitions of messages, segments, fields, and data types.

Perl scripts are applied to the ASCII delimited files to generate XML schema definitions. The structure of the generated schemas follows from the design considerations described above. The algorithms instantiated in these Perl scripts are not described in detail here and are not part of this specification, but are publicly available on demand.

3.2.3 Examples

3.2.3.1 Schema and DTD Fragments

These are actual fragments of the real schemas provided as illustrations. There is not enough of the schemas included here to allow for validation of the example messages. Messages will validate against the complete schema.

```xml
<!-- Data type definitions -->
...  
<!-- PRIMITIVE DATATYPE ID -->
<simpleType name="ID">  
  <restriction base="string" />  
</simpleType>  
...
<!-- COMPOSITE DATATYPE CE -->
<complexType name="CE">  
  <sequence>  
    <element minOccurs="0" maxOccurs="1" ref="CE.1" />  
    <element minOccurs="0" maxOccurs="1" ref="CE.2" />  
    <element minOccurs="0" maxOccurs="1" ref="CE.3" />  
    <element minOccurs="0" maxOccurs="1" ref="CE.4" />  
    <element minOccurs="0" maxOccurs="1" ref="CE.5" />  
    <element minOccurs="0" maxOccurs="1" ref="CE.6" />  
  </sequence>  
</complexType>  
...  
<!-- COMPONENT CE.1 -->
<attributeGroup name="CE.1.ATTRIBUTES">  
  <attribute name="LongName" type="string" fixed="identifier (ID)" />  
  <attribute name="Type" type="string" fixed="ID" />  
  <attribute name="Table" type="string" fixed="0" />  
</attributeGroup>  
<element name="CE.1">  
```

Copyright © 1999-2001, Health Level Seven, Inc. All rights reserved.
<complexType>
  <simpleContent>
    <extension base="ID">
      <attributeGroup ref="CE.1.ATTRIBUTES"/>
    </extension>
  </simpleContent>
</complexType>

<!-- Field definitions -->

<!-- FIELD MSH.19 -->
<attributeGroup name="MSH.19.ATTRIBUTES">
  <attribute name="Item" type="string" fixed="693" />
  <attribute name="Table" type="string" fixed="0" />
  <attribute name="LongName" type="string" fixed="Principal Language Of Message" />
  <attribute name="Type" type="string" fixed="CE" />
</attributeGroup>

<!-- Segment definitions -->

<!-- SEGMENT MSH -->
<element name="MSH">
  <complexType>
    <sequence>
      <element minOccurs="1" maxOccurs="1" ref="MSH.1" />
      <element minOccurs="1" maxOccurs="1" ref="MSH.2" />
      <element minOccurs="0" maxOccurs="1" ref="MSH.3" />
      <element minOccurs="0" maxOccurs="1" ref="MSH.4" />
      <element minOccurs="0" maxOccurs="1" ref="MSH.5" />
      <element minOccurs="0" maxOccurs="1" ref="MSH.6" />
      <element minOccurs="0" maxOccurs="1" ref="MSH.7" />
      <element minOccurs="0" maxOccurs="1" ref="MSH.8" />
      <element minOccurs="1" maxOccurs="1" ref="MSH.9" />
      <element minOccurs="1" maxOccurs="1" ref="MSH.10" />
      <element minOccurs="1" maxOccurs="1" ref="MSH.11" />
      <element minOccurs="1" maxOccurs="1" ref="MSH.12" />
      <element minOccurs="0" maxOccurs="1" ref="MSH.13" />
      <element minOccurs="0" maxOccurs="1" ref="MSH.14" />
      <element minOccurs="0" maxOccurs="1" ref="MSH.15" />
      <element minOccurs="0" maxOccurs="1" ref="MSH.16" />
      <element minOccurs="0" maxOccurs="1" ref="MSH.17" />
      <element minOccurs="0" maxOccurs="unbounded" ref="MSH.18" />
      <element minOccurs="1" maxOccurs="1" ref="MSH.19" />
      <element minOccurs="0" maxOccurs="1" ref="MSH.20" />
      <element minOccurs="0" maxOccurs="unbounded" ref="MSH.21" />
    </sequence>
  </complexType>
</element>

<!-- Message definitions -->

<!-- MESSAGE ACR -->
<!-- .. message definition ACR -->
<element name="ACK">
  <complexType>
    <sequence>
      <element minOccurs="1" maxOccurs="1" ref="MSH" />
      <element minOccurs="1" maxOccurs="1" ref="MSA" />
      <element minOccurs="0" maxOccurs="1" ref="ERR" />
    </sequence>
  </complexType>
</element>
<!-- Data type definitions -->
<!-- PRIMITIVE DATATYPE ID -->
<ENTITY % ID "(#PCDATA)"/>
<!-- COMPOSITE DATATYPE CE -->
<ENTITY % CE "(CE.1?,CE.2?,CE.3?,CE.4?,CE.5?,CE.6?)"/>
<!-- COMPONENT CE.1 -->
<ENTITY % CE.1 CONTENT "%ID;"/>
<ENTITY % CE.1 ATTRIBUTES
  "LongName CDATA #FIXED 'identifier (ID)'"
  "Type CDATA #FIXED 'ID'"
  "Table CDATA #FIXED '0'"
  "Type CDATA #FIXED 'CE'"
/>
<ATTLIST CE.1 %CE.1 ATTRIBUTES;>

<!-- Segment definitions -->
<!-- SEGMENT MSH -->
<ENTITY % MSH CONTENT
  "(MSH.1,MSH.2,MSH.3?,MSH.4?,MSH.5?,MSH.6?,MSH.7,MSH.8?,MSH.9,MSH.10,
   MSH.11,MSH.12,MSH.13,MSH.14?,MSH.15?,MSH.16?,MSH.17,MSH.18*,MSH.19?,
   MSH.20?,MSH.21*)"
/>
<ENTITY % MSH CONTENT/>

<!-- Message definitions -->
<!-- MESSAGE ACK -->
<ENTITY % ACK CONTENT "(MSH,MSA,ERR?)"/>
<ENTITY % ACK CONTENT/>

3.2.3.2 V2 and v2.xml Messages

MSH:|&REGADT|MCH|IFENG||199112311501||ADT^A04^ADT_A01|000001|P|2.4|||
   EVN|A04|199901101500|199901101400|01|199901101410
   PID|191919^"GENHOS\"|MR^71-66-9256|^USSSA\"|SS
   [253763]|MASSIE^JAMES^A|19560129^|371 ZOBERLEIN|^ISHPEMING^MI^49849^"|171 ZOBERLEIN|^ISHPEMING^MI^49849|900485-5344|900485-5344|19990199925^GENHOS\"AN|371-66-9256|
   NKL1|MASSIE^ELLEN|SPOUSE|171 ZOBERLEIN|^ISHPEMING^MI^49849|900485-5344|900485-5345
   NKL2|MASSIE^MARYLOU|MOTHER|300 ZOBERLEIN|^ISHPEMING^MI^49849|900485-5344
   NK1|123 INDUSTRY WAY|^ISHPEMING^MI^49849|900485-5342|900485-1200|EC1^FIRST EMERGENCY CONTACT
   NK1|3
   OBX|ST|1010.1^BODY WEIGHT||62|kg|||F
   OBX|ST|1010.1^HEIGHT||190|cm|||F
   DG1|1|19|BIOPSY||
<XPN.1>MASSIE</XPN.1><XPN.2>ELLEN</XPN.2>
<PID.19>371-66-9256</PID.19>
</NK1>
<NK1.1>1</NK1.1>
<NK1.2>
<XPN.1>MASSIE</XPN.1>
<XPN.2>ELLEN</XPN.2>
</NK1.2>
<NK1.3>
<CE_0063.1>SPOUSE</CE_0063.1>
</NK1.3>
<NK1.4>
<XAD.1>171 ZOBERLEIN</XAD.1>
<XAD.3>ISHPEMING</XAD.3>
<XAD.4>MI</XAD.4>
<XAD.5>49849</XAD.5>
</NK1.4>
<NK1.5>
<XTN.1>(900) 485-5344</XTN.1>
</NK1.5>
<NK1.6>
<XTN.1>(900) 545-1234 ~(900) 545-1200</XTN.1>
</NK1.6>
<NK1.7>
<CE_0131.1>EC1</CE_0131.1>
<CE_0131.2>FIRST EMERGENCY CONTACT</CE_0131.2>
</NK1.7>
</NK1>
<NK1.1>2</NK1.1>
<NK1.2>
<XPN.1>MASSIE</XPN.1>
<XPN.2>MARYLOU</XPN.2>
</NK1.2>
<NK1.3>
<CE_0063.1>MOTHER</CE_0063.1>
</NK1.3>
<NK1.4>
<XAD.1>300 ZOBERLEIN</XAD.1>
<XAD.3>ISHPEMING</XAD.3>
<XAD.4>MI</XAD.4>
<XAD.5>49849</XAD.5>
</NK1.4>
<NK1.5>
<XTN.1>(900) 485-5344</XTN.1>
</NK1.5>
<NK1.6>
<XTN.1>(900) 545-1234 ~(900) 545-1200</XTN.1>
</NK1.6>
<NK1.7>
<CE_0131.1>EC2</CE_0131.1>
<CE_0131.2>SECOND EMERGENCY CONTACT</CE_0131.2>
</NK1.7>
</NK1>
<NK1.1>3</NK1.1>
<NK1>
  <NK1.4>
    <XAD.1>123 INDUSTRY WAY</XAD.1>
    <XAD.3>ISHPEMING</XAD.3>
    <XAD.4>MI</XAD.4>
    <XAD.5>49849</XAD.5>
    <XAD.6>**</XAD.6>
  </NK1.4>
  <NK1.6>
    <XTN.1>(900) 545-1200</XTN.1>
  </NK1.6>
  <CE.0131.1>EM</CE.0131.1>
  <CE.0131.2>EMPLOYER</CE.0131.2>
  <NK1.7>
    <XON.1>ACME SOFTWARE COMPANY</XON.1>
  </NK1.7>
  <CE.0131.1>P</CE.0131.1>
  <CE.0131.2>PROGRAMMER</CE.0131.2>
  <PN.1>19940605</PN.1>
  <CE.0131.1>EMPLOYER</CE.0131.1>
  <CE.0131.2>EMPLOYER</CE.0131.2>
  <PB.1>99501101410</PB.1>
  <CE.0131.1>GENHOS</CE.0131.1>
  <CE.0131.2>GENHOS</CE.0131.2>
  <PV1>
    <PV1.2>O</PV1.2>
    <PV1.3>
      <PL.1>O/R</PL.1>
    </PV1.3>
    <PV1.7>
      <XCN.1>0148</XCN.1>
      <XCN.2>ADDISON,JAMES</XCN.2>
    </PV1.7>
    <PV1.8>
      <XCN.1>0148</XCN.1>
      <XCN.2>ADDISON,JAMES</XCN.2>
    </PV1.8>
    <PV1.9>
      <XCN.1>0148</XCN.1>
      <XCN.2>ADDISON,JAMES</XCN.2>
    </PV1.9>
    <PV1.10>AMB</PV1.10>
    <PV1.17>
      <XCN.1>0148</XCN.1>
      <XCN.2>ADDISON,JAMES</XCN.2>
    </PV1.17>
    <PV1.18>8</PV1.18>
    <PV1.19>
      <CX.1>1400</CX.1>
    </PV1.19>
    <PV1.20>
      <FC.1>A</FC.1>
    </PV1.20>
    <PV1.39>GENHOS</PV1.39>
    <PV1.44>199501101410</PV1.44>
  </PV1>
  <PV2>
    <PV2.8>199901011400</PV2.8>
    <PV2.34>199901011400</PV2.34>
  </PV2>
  <OBX>
    <OBX.2>ST</OBX.2>
    <OBX.3>
      <CE.1>1010.1</CE.1>
      <CE.2>BODY WEIGHT</CE.2>
    </OBX.3>
    <OBX.5>62</OBX.5>
    <OBX.6>
      <CE.1>kg</CE.1>
    </OBX.6>
    <OBX.11>P</OBX.11>
  </OBX>
</NK1>
3.3 References

[rfHL7v231] HL7 Version 2.3.1, ANSI/HL7 V2.3.1-1999, approved as an ANSI standard April 14, 1999
http://www.hl7.org

[rfHL7v24] HL7 Version 2.4, ANSI/HL7 V2.4-2000, approved as an ANSI standard October 6, 2000
http://www.hl7.org


http://www.w3.org/XML
http://www.w3.org/2001/XMLSchema.html


[rfOemig] Frank Oemig’s Home Page
http://www.sr-solutions.de/oemig/

http://www.gkrueger.com/other/hl7/

[rfHL7v3ITS] HL7 Version 3 – XML ITS, see HL7 Version 3 (Draft)
http://www.hl7.org