

v2.xml

XML Encoding Rules For HL7 v2 Messages

Control/Query Technical Committee XML Special Interest Group

Editor:	Kai U. Heitmann, M. D. University of Cologne, Germany
Co-Chairs	Paul V. Biron (paul.v.biron@kp.org) Kaiser Permanente Kai U. Heitmann, MD (kai.heitmann@medizin.uni-koeln.de) University of Cologne, Germany Mike Henderson (mike@easterninformatics.com) Eastern Informatics Doug Pratt (doug.pratt@smed.com) Siemens Medical Solutions Health Services Larry Reis (71045.14@compuserve.com) Helus, inc. Mark Shafarman (mark.shafarman@oracle.com) Oracle

Contents

v2.xml XML Encoding Rules For HL7 v2 Messages	1
Contents	1
Preface	3
Acknowledgements	3
Remarks for 1 st Membership Ballot	4
1 Introduction	4
1.1 Background	4
1.2 Benefits From Using XML as an Alternative v2 Interchange Format	5
1.3 Overview of HL7 Database	5
1.4 Scope for HL7 Version 2	6
1.5 Version 2 Message Definitions	6
1.5.1 Version 2 Hierarchical Message Structure	6
1.5.2 Abstract Message Syntax Definitions	7
2 Specification	8
2.1 Introduction to the XML Representation	8
2.2 A First Example	8
2.3 Messages Identifications and Trigger Events	9
2.3.1 Message Structure IDs	9
2.4 Segments	10
2.4.1 Groups of Segments	10
2.4.2 Lists	11
2.5 Fields	12

2.6	Data Types.....	13
2.6.1	Primitive Data Types.....	13
2.6.2	Composite Data Types.....	13
2.6.3	Wildcard.....	15
2.6.4	CM Data Types.....	15
2.7	Processing Rules for v2.xml Messages.....	15
2.7.1	XML Application Processing Rules.....	15
2.7.2	Inter-version Backward Compatibility.....	15
2.7.3	Message Fragmentation and Continuation.....	15
2.7.4	Batch Messages.....	16
2.7.5	Message Delimiters.....	16
2.7.6	Null Values, Empty Values.....	16
2.7.7	Repetition of Fields.....	17
2.7.8	Escape Character Sequences Used in v2 Data Types.....	17
2.7.8.1	Text Formatting Escape Sequences.....	17
2.7.8.2	Structural Escape Sequences.....	18
2.7.8.3	Character References.....	18
2.7.8.4	Character Set Switching.....	18
2.7.8.5	Escaping XML Markup.....	18
2.7.9	Message Building Rules.....	18
2.7.10	Special Characters in Schemas/DTDs.....	19
2.8	Translating Between Standard Encoding and XML Encoding.....	19
2.9	Outstanding Problems.....	19
2.9.1	Database.....	19
2.9.2	Choices of Segments.....	20
2.9.3	Conformance With HL7 Version 3 XML Representation.....	20
3	Appendix.....	20
3.1	Normative Appendix.....	20
3.1.1	Design Considerations.....	20
3.1.1.1	XML Schema Optimization.....	20
3.1.1.1.1	Message Length.....	20
3.1.1.1.2	Structural Complexity.....	21
3.1.1.1.3	Localization Issues.....	21
3.1.1.1.4	"Looseness" of a DTD, Conformance Testing of Other Business Rules.....	22
3.1.1.1.5	Automation Considerations.....	22
3.1.1.2	List of Messages With Equal Message Structures.....	22
3.1.1.3	List of Schemas and DTDs.....	23
3.2	Informative Appendix.....	23
3.2.1	Localization of messages.....	23
3.2.1.1	DTD Localization.....	24
3.2.1.2	Schema Localization.....	24
3.2.2	Extracting Subsets Of The HL7 Database.....	25
3.2.2.1	Messages and Their Segments.....	25
3.2.2.1.1	HL7 Database Tables Used.....	25
3.2.2.1.2	SQL Query.....	26
3.2.2.1.3	Select Those Message Structures Reflected in Table 0354.....	27
3.2.2.2	Segments, Fields, and Data Types.....	27
3.2.2.2.1	HL7 Database Tables Used.....	27
3.2.2.2.2	SQL Query.....	27
3.2.2.3	Data Types and Their Data Type Components.....	28
3.2.2.3.1	HL7 Database Tables Used.....	28
3.2.2.3.2	SQL Query.....	28
3.2.2.4	Summary of Additional Files.....	29
3.2.3	Algorithms.....	29
3.2.4	Examples.....	29
3.2.4.1	Schema and DTD Fragments.....	29

3.2.4.2	V2 and v2.xml Example Messages	32
3.2.4.2.1	Short Example.....	32
3.2.4.2.2	Long Example.....	35
3.3	References	41

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*. It also specifies an interchange format and management rules, the *encoding rules* for HL7 message instances (see Figure1). The objective of this document is to present alternate 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.

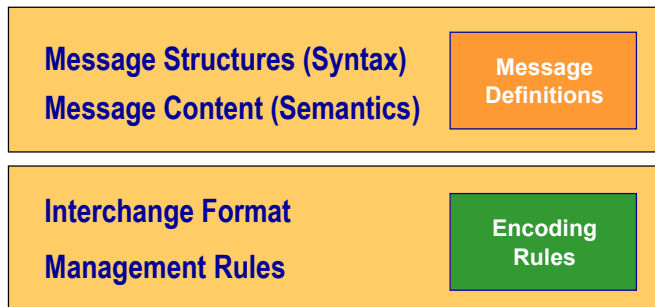


Figure1: The standard specification specifies message definitions and encoding rules.

It is not the intent of this document to replace the standard sequence oriented encoding rules, that use “vertical bars” and other delimiters (so called “vertical bar encoding”), but rather to provide an alternative way of encoding. Furthermore, message definitions given in the Version 2.3.1 and 2.4 standard are also untouched. However, if you are going to use XML for version 2.x messages, this HL7 normative document describes how to do that. This document does **not** modify the message definitions, only the way they are encoded.

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 (DTDs) 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. I wish to thank 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. Valuable discussions came from James Case, Ivan Emelin, Susan Abernathy, Peter Rontey, Nick Radov, John Firl, Jennifer Puyenbroek, Chuck Meyer, Tim Barry, Jacob Valenta, Eliot Muir, Grahame Grieve, Koo Weng On. Special thanks for his support to Tom de Jong.

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 Membership Ballot

For the 1st Membership Level Ballot, some remarks have to be made concerning this specification.

General Knowledge

This specification assumes general knowledge of XML technology on the part of readers. Readers unfamiliar with XML may gain the requisite knowledge from the following standards:

- XML 1.0, 2nd Edition [rfXML]
- XML Schema [rfXMLSchema]

Accompanying Material

- In addition to this specification, a set of DTDs and XML Schemas, hereafter called “schemas” in general, is provided. They are the work product of this specification. Please refer to section 3.1.1.3 for further details.
- The use of XML schema ([rfXMLSchema], a W3C recommendation since May 2001) is recommended by HL7 for all normative specifications. DTDs are not part of the normative specification, but rather added as an informative appendix in order to support vendors with migration from DTDs to XML schemas.
- The class of message instances validated by the normative schemas distributed as part of this specification equals the class of message instances validated by the informative DTDs.

Outstanding Issues

- The suggested narrative segment group names as of section 2.4.1 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, instead of that many group names are constructed algorithmically (sequence) in this specification, but will be changed as soon as they are available as a technical addendum to this specification. Messages that are already complete concerning the narrative segment group names are ADT_A01 and ORU_O01. There will be a technical correction to this specification as soon as segment group names are finalized in the original standard work.
- CM data types are not consequently eliminated (some relicts) and thus not handled properly in this specification. Refer to section 2.6.4 for further information. There will be a technical correction to this specification as soon as they are finalized in the original standard work.
- There is no support of the choice indicator in segment definitions, because no choice indicators could be found in the standard documents (and nor in the database, see section 1.3).

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 several benefits using XML as an interchange format.

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. Vertical bar encoding has some severe limitations, for example there are a limited number of structured levels that could be expressed that way.

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. For the encoding part, trained personnel are much easier to find if using XML than experts familiar with vertical bar encoding rules. Of course explicit knowledge about the underlying semantic assumptions is still inevitable.

At former times 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. Meanwhile a limited number of (expensive) commercial tools are available for that purpose but its still far away from being a generic approach used in all areas of business, not in healthcare only. This will become even more important as the healthcare area and its messaging requirements gets more and more points of contact with other sectors of "daily business". Using XML and associated parsers and generators could help vendors (and users) to be well prepared for these upcoming challenges.

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 HL7 Database

Underlying the HL7 Standard is a 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. All data were drawn from the normative standard documents, largely algorithmically and to some minor amount handcrafted.

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. 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. There have been a lot of such fixes, both in the database and in the XML encoding since the last ballot cycle (committee level ballot). The procedures for deriving the schemas 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 initiated 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 Admission, Discharge and Transfer (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 or optional and repeatable or non-repeating. Each segment begins with a three-character literal value that identifies it within a message (segment identifier). For example, an 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. Field 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 include string of characters, number, code etc. Complex data types are comprised 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, whose comprises of several sub-components in order to express the various parts of a person’s name. There are many 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. Such segment group names are uppercase (e. g. "PROCEDURE", "INSURANCE").

<u>ADT^A01^ADT A01</u>	<u>ADT Message</u>	<u>Status</u>	<u>Comment</u>	<u>Chapter</u>
MSH	Message Header			2
EVN	Event Type			3
PID	Patient Identification			3
[PD1]	Additional Demographics			3
[{ ROL }]	Role			12
[{ NK1 }]	Next of Kin / Associated Parties			3
[PVI]	Patient Visit			3
[PV2]	Patient Visit - Additional Info.			3
[{ ROL }]	Role			12
[{ DB1 }]	Disability Information			3
[{ OBX }]	Observation/Result			7
[{ AL1 }]	Allergy Information			3
[{ DGI }]	Diagnosis Information			6
[DRG]	Diagnosis Related Group			6
[{	group PROCEDURE begin			
PR1	Procedures			6
[{ ROL }]	Role			12
}]	group PROCEDURE end			
[{ GT1 }]	Guarantor			6
[{	group INSURANCE begin			
IN1	Insurance			6
[IN2]	Insurance Add. Info.			6
[{ IN3 }]	Insurance Add. Info -Cert.			6
[{ ROL }]	Role			12
}]	group INSURANCE end			
[ACC]	Accident Information			6
[UB1]	Universal Bill Information			6
[UB2]	Universal Bill 92 Information			6
[PDA]	Patient Death and Autopsy			3

Figure 2: Abstract message syntax definition for message type ADT_A01, drawn from chapter 3 in [rfHL7v24] and adopted to the corresponding style containing group names (colored here for better readability)

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

HL7 Abstract Message Syntax	Equivalent Cardinality in XML Schema (minOccurs .. maxOccurs)	Equivalent XML DTD Occurrence Indicator
[]	0 .. 1	?
{ }	1 .. unbounded	+
{ [] } = [{ }]	0 .. unbounded	*
no bracket or brace	1 .. 1	no occurrence indicator (one exactly)

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 encoding rules specified 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|ZZ9380
ERR|PID^1^16^103^Table value not found^HL70357
```

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

```
<ACK
  xmlns="urn:hl7-org:v2xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:hl7-org:v2xml ACK.xsd">
  <MSH>
    <MSH.1|</MSH.1>
    <MSH.2>^~\&|</MSH.2>
    <MSH.3>
      <HD.1>LAB</HD.1>
    </MSH.3>
    <MSH.4>
      <HD.1>767543</HD.1>
    </MSH.4>
    <MSH.5>
      <HD.1>ADT</HD.1>
    </MSH.5>
    <MSH.6>
      <HD.1>767543</HD.1>
    </MSH.6>
    <MSH.7>
      <TS.1>199003141304-0500</TS.1>
    </MSH.7>
    <MSH.9>
      <CM_MSG.1>ACK</CM_MSG.1>
      <CM_MSG.3>ACK</CM_MSG.3>
    </MSH.9>
    <MSH.10>XX3657</MSH.10>
    <MSH.11>
      <PT.1>P</PT.1>
    </MSH.11>
    <MSH.12>
      <VID.1>2.4</VID.1>
    </MSH.12>
  </MSH>
  <MSA>
    <MSA.1>AR</MSA.1>
    <MSA.2>ZZ9380</MSA.2>
  </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>
      <CE.1>103</CE.1>
      <CE.2>Table value not found</CE.2>
      <CE.3>HL70357</CE.3>
    </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 role is played by the Message Type 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. Allowed values are defined in HL7 table 0076 – Message Type.
- The second component is the Trigger Event code with values like A01, O01, R01 etc. Refer to HL7 table 0003 – Trigger Event for a complete listing.
- The third component is the abstract message structure ID defined by HL7 Table 0354 – Message Structure.

2.3.1 Message Structure IDs

In the 2.x standard(s), many trigger events share the same abstract message syntax. This fact became standardized in v2.3.1 and was introduced in the form of the Message Structure component of MSH-9 (component 3).

The v2.xml schemas are based on the described message structure ID. 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 (i.e., they have the same sequence and cardinality of segments). 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, requires using the schema definition for the ADT_A01 message. The standard documents contain tables where the message structure IDs are listed (see also section 3.1.1.2 for information about messages with equal structures).

a) Part of the Abstract Message Syntax definition for an ADT A04 message

ADT^A04^ADT_A01

MSH
EVN
PID
...

ADT Message

Message Header
Event Type
Patient Identification
...

b) Example of a Message Header segment, with Message Type, Trigger Event, and the Message Structure ID repeated in MSH-9

MSH|^~\&|ADT1|MCM|LABADT|MCM|198808181126||**ADT^A04^ADT_A01**|M|P|2.4|

Figure 3: Example for Abstract Message Syntax (fragment) in (a) and Message Header Segment (b)

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

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

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

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

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. During discussions of the informative document it was found useful to deepen the XML element hierarchy by adding grouping elements.

Groups containing more than a single segment are thus 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 2nd column in Figure 4 containing "group *group_name* begin/end"). 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 have to appear also in an XML instance documents containing messages of that type, i.e. groups of segments are surrounded with their own tags.

There was no explicit way to express these groups in the traditional v2 "vertical bar" encoding of messages. Introducing explicit segment group names is thus an essential change between vertical bar encoding and XML encoding. Furthermore, this allows elements to be accessed in a reasonable manner within an X-Path expression (see [rfXPath]).¹

¹ By this, an application can refer to specific XML items explicitly by name (e.g. `ADT_A01/PID.5/XPN.3/FN.1`) or they can refer to them by position (e.g. `ADT_A01/PID.5/*[position()=3]/*[position()=1]`). By taking the latter approach, one no longer has to take care what the name of the data type is.

Segment group names are uppercase. In almost all cases the names convey the semantics carried by the group of segments itself, for example IN segments are bundled by the “INSURANCE” group, PV1 PV2 segments are bundles as the “VISIT” group etc.

<u>ADT^A01^ADT A01</u>	<u>ADT Message</u>	<u>Status</u>	<u>Comment</u>	<u>Chapter</u>
MSH	Message Header			2
EVN	Event Type			3
PID	Patient Identification			3
...				
{ {	group PROCEDURE begin			
PR1	Procedures			6
{ { ROL }	Role			12
}}	group PROCEDURE end			
...				
{ {	group INSURANCE begin			
IN1	Insurance			6
[IN2]	Insurance Add. Info.			6
{ { IN3 }	Insurance Add. Info -Cert.			6
{ { ROL }	Role			12
}}	group INSURANCE end			
...				

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

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 these 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.

```

<ADT_A01>
  ...
  <ADT_A01.PROCEDURE>
    <PR1>
      ...
    </PR1>
    <ROL>
      ...
    </ROL>
  </ADT_A01.PROCEDURE>
  ...
  <ADT_A01.INSURANCE>
    <IN1>
      ...
    </IN1>
    <IN2>
      ...
    </IN2>
    <IN3>
      ...
    </IN3>
  </ADT_A01.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).

SEQ	LEN	DT	OPT	RP/#	TBL#	ITEM#	ELEMENT NAME
1	3	ID	B		0003	00099	Event Type Code
2	26	TS	R			00100	Recorded Date/Time
3	26	TS	O			00101	Date/Time Planned Event
4	3	IS	O		0062	00102	Event Reason Code
5	250	XCN	O	Y	0188	00103	Operator ID
6	26	TS	O			01278	Event Occurred
7	180	HD	O			01534	Event Facility

Figure 5: Segment attribute table of the EVN segment (fragment, drawn from chapter 3 in [rHL7v24])

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 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. Please note, that the correct EVN encoding is a little bit different from this simplified example.

```
EVN|A05|199901061000|199901101400|01||199901061000
```

```
...
<EVN>
  <EVN.1>A05</EVN.1>
  <EVN.2>199901061000</EVN.2>
  <EVN.3>199901101400</EVN.3>
  <EVN.4>01</EVN.4>
  <EVN.6>199901061000</EVN.6>
</EVN>
...
```

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 (unless needed to be expressed explicitly, see section 2.7.6). 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. In the XML schemas, the component’s item number, table reference, long name, and data type is provided by the <annotation> <appinfo> mechanism, in addition a <documentation> is given containing the long name of the field (also the language is defined by the standard xml:lang attribute) as specified in the standard.

The example below shows the XML schema definition of the EVN.1 field element along with its annotations. The second example shows the same definition in the DTD world. Please not that – in absence of a corresponding annotation mechanism in DTDs – the additional information mentioned above is provided as a comment only.

```

<!--
  FIELD EVN.1
-->
<xsd:element name="EVN.1" type="ID">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">Event Type Code</xsd:documentation>
    <xsd:appinfo>
      <hl7:Item>99</hl7:Item>
      <hl7:Type>ID</hl7:Type>
      <hl7:Table>3</hl7:Table>
      <hl7:LongName>Event Type Code</hl7:LongName>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>

```

```

<!--
  FIELD EVN.1
-->
<!ENTITY % EVN.1.CONTENT "%ID; ">
<!ELEMENT EVN.1 %EVN.1.CONTENT;>
<!-- Item='99' Type='ID' Table='3' LongName='Event Type Code' -->

```

If a receiver receives an XML instance that is validated against the schema, the receiving parser can make use of the information that is provided in the annotations appinfo (application information) and documentation (user information) element content of the underlying schema.

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 for example 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 4th field of the EVN segment (see Figure 5) is of type IS (a value drawn from an HL7 defined table), which is a primitive data type. The corresponding XML instance fragment looks like the following example:

```
<EVN.4>01</EVN.4>
```

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 the table shown in 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)>

```

Data structure	seq	component description	Table ID	Data type code
CE	1	identifier	-	ST
CE	2	text	-	ST
CE	3	name of coding system	396	IS
CE	4	alternate identifier	-	ST
CE	5	alternate text	-	ST
CE	6	name of alternate coding system	396	IS

Figure 6: CE data type components drawn from [rfHL7v24] (above) and a table representation of the same definition

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.

```
...|F-11380^CREATININE^I9^2148-5^CREATININE^LN|...
```

```
<CE.1>F-11380</CE.1>
<CE.2>CREATININE</CE.2>
<CE.3>I9</CE.3>
<CE.4>2148-5</CE.4>
<CE.5>CREATININE</CE.5>
<CE.6>LN</CE.6>
```

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.

Considering the following example where a field of type CE 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:

<p><u>Incorrect:</u></p> <pre><OBX.3> F-11380 </OBX.3></pre>	<p><u>Correct:</u></p> <pre><OBX.3> <CE.1>F-11380</CE.1> </OBX.3></pre>
--	---

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. Annotation mechanism is used to express the component's data type, long name, and table, as shown here for CE.1. Again, in DTDs (second example) this information is provided as comments only.

```
<!--
  COMPONENT CE.1
-->
<xsd:element name="CE.1" type="ST">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">identifier</xsd:documentation>
    <xsd:appinfo>
      <h17:Type>ST</h17:Type>
      <h17:LongName>identifier</h17:LongName>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
```

```
<!--  
    COMPONENT CE.1  
-->  
<!ENTITY % CE.1.CONTENT "%ST;">  
<!ELEMENT CE.1 %CE.1.CONTENT;>  
<!-- Type='CE_0002' Table='2' LongName='identifier' -->
```

2.6.3 Wildcard

In the HL7 Database, some few 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. It should be avoided to use wildcard datatypes however, because making use of them prevents sufficient conformance testing of the resulting message instances.

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 beginning with 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. A technical correction has been applied to HL7 v2.3.1 and v2.4, where these renamed data types are listed. These corrected names must be used when encoding CM data types with XML.

2.7 Processing Rules for v2.xml Messages

2.7.1 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 schema (i. e. DTD or XML schema) definitions (see section 3.1.1.3).

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 schema.

2.7.2 Inter-version Backward Compatibility

The vertical bar encoding provides a certain amount of backward compatibility between versions of the v2.x world. This makes ramifications possible in the sense of changing data types between versions (i. e. IS to CE data type), ramification by adding a field, inserting a segment or turning a repeating segment into a repeating group of segments. Because the XML encoding makes explicitly use of constructs touched by the changes mentioned above, an inter-version backward compatibility is not given. For example, if a data type for a field is changed from IS (primitive) to CE (composite), the CE composite data type introduces its own tags, in other words, the former IS field now has child elements drawn from the composite CE data type.

However, it should be easy to achieve XML transformations from an XML instance for one version to another using corresponding transformation rules or style sheets (which are not provided here).

2.7.3 Message Fragmentation and Continuation

Sometimes, implementation limitations require that large messages or segments be broken into manageable chunks. Message fragmentation and 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, i. e. several 100k bytes of information or more (about an average message length) at once.

2.7.4 Batch Messages

There are instances when it is convenient to transfer a batch of HL7 messages. Common examples would be a batch of financial posting detail transactions (DFTs) sent from an ancillary to a financial system.

Chapter 2 of the standard defines such a mechanism to wrap multiple valid HL7 messages by wrapping control segments in order to form a batch of messages. For that purpose specific file and batch header and trailer control segments FHS, FTS, BHS, BTS are defined.

In the XML encoding, it is also possible to wrap multiple messages with the corresponding control segments. The definitions could be found in the messages schema (*message.xsd*, or *messages.dtd* respectively).

2.7.5 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 MSH 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. They must still be sent, because MSH.1 and MSH.2 are required fields in the v2.x standard. Please note, that the special character “&” must be escaped in order to be included in an XML instance document (see also section 2.7.10)

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

2.7.6 Null Values, Empty Values

Null values of fields having a primitive data type or of components of composite data types should be expressed in the same way as in the vertical bar encoding (|"""). The following example carries a null value in the <XAD.6> data type component.

```
<NK1.4>
  <XAD.1>
    <SAD.1>123 INDUSTRY WAY</SAD.1>
  </XAD.1>
  <XAD.3>ISHPEMING</XAD.3>
  <XAD.4>MI</XAD.4>
  <XAD.5>49849</XAD.5>
  <XAD.6>""</XAD.6>
</NK1.4>
```

Explicit empty (missing) values are expressed by empty element content.

2.7.7 Repetition of Fields

Repetition of fields is handled by repeating the appropriate tags. This has to be done also with fields which are comprised of composite data types. The following examples demonstrate correct repetition in the XML encoding.

<p>example 1</p> <pre><ECR.3>PARAM1</ECR.3> <ECR.3>PARAM2</ECR.3></pre>	<p>example 2</p> <pre><NK1.6> <XTN.1>(900)545-1234</XTN.1> </NK1.6> <NK1.6> <XTN.1>(900)545-1200</XTN.1> </NK1.6></pre>
--	--

2.7.8 Escape Character Sequences Used in v2 Data Types

Chapter 2 of the v2.x standard specifies escape character sequences to be used in fields of certain types. When a field of data type TX, FT, or CF is being encoded, these escape characters 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.

For the XML encoding we must differentiate between data type associated escape characters (text formatting), structural escape sequences and character encoding / character set switching characters. They have to be handled differently when using v2.xml.

2.7.8.1 Text Formatting Escape Sequences

/H/ and /N/ are defined in chapter 2 of the standard as indicating begin and end highlight of text in a text field. In v2.xml these characters are replaced by corresponding XML elements that can easily be processed.

Escape characters as defined in the v2.x standard	Replacement to be used in v2.xml encoding	Meaning
\H\	<escape V="H">	start highlighting text
\N\	<escape V="N">	Normal text (stop highlighting)

Figure 7: Escape characters drawn from [rfHL7v24] and their replacement in the v2.xml encoding

An example, v2.x and below the corresponding v2.xml notation

A \H\special\N\ word
A <escape ="H">special<escape ="N"> word

There is also the possibility of specifying troff-commands in text fields. They are escaped accordingly. The following table just shows examples and is not complete. Please refer to chapter 2 of the v2.x standard.

Escape characters as defined in the v2.x standard	Replacement to be used in v2.xml encoding	Meaning
\.br\	<escape V="\.br">	begin new output line
\.spn\	<escape V="\.spn">	skip <i>n</i> vertical space
\.in±n\	<escape V="\.in±n">	indent <i>n</i> spaces
\.ti±n\	<escape V="\.ti±n">	temporarily indent <i>n</i> spaces

Figure 8: Escape characters (troff commands) drawn from [rfHL7v24] and their replacement in the v2.xml encoding

An example:

```
\.in+4\\\.ti-4\ 1. The cardiomediastinal silhouette...
```

is expressed in v2.xml as

```
<escape V=".in+4"><escape V=".ti-4"> 1. The cardiomediastinal silhouette...
```

2.7.8.2 Structural Escape Sequences

The escape character sequences `\F`, `\S`, `\T` and `\R` mentioned in chapter 2 of the v2.x standard used to indicate the literal field, component, subcomponent and repetition separators may not be used in the v2.xml encoding. They are superfluous because the XML does not make use of these structural separator characters.

2.7.8.3 Character References

The vertical bar character encoding mechanism using `/Xxxx/` as a character reference, and `/Zxxx/` to refer to a locally defined character reference is deprecated in the v2.xml encoding. Instead, the standard XML character reference mechanism `&#xxx` must be used.

An example:

```
\Xc9\ditions Lenard
```

is expressed in v2.xml as

```
&#xc9;ditions Lenard
```

For locally defined character references outside that scope, the private area of Unicode should be addressed.

2.7.8.4 Character Set Switching

Character set switching as described in chapter 2 of the v2.x standard cannot be addressed in XML. XML has only a single character set – UCS/Unicode. That implies, that within a single XML entity (i.e., file) the encoding must be the same. Switching character sets is thus not supported.

There will be workaround solution for the v3 XML ITS that is not yet completely determined. The v2.xml ITS will use the same mechanism.

2.7.8.5 Escaping XML Markup

It is the responsibility of sending applications to escape all characters occurring in data that may be interpreted by an XML processor as being markup characters. Depending on context, these include all characters listed in Figure 9.

Character	Pre-defined Entity	Character Reference
<	<	<
&	&	&
"	"	"
'	'	'

Figure 9: XML markup characters

2.7.9 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. As 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.

2.7.10 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;#38;” An example is “Critical Range for Ordinal & Continuous Obs” that becomes “Critical Range for Ordinal &#amp;#38; 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 “'”, e. g. “Contact’s Tel. Number” becomes “Contact's Tel. Number”.

The same rules apply to XML instance documents.

Please note, that spelling and capitalization of all tags in the XML encoding must be the same as defined in the HL7 database (see section 1.3). Please refer to the schemas, which reflect these rules.

2.8 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 2.7 need to be taken into account when performing the translations.

2.9 Outstanding Problems

2.9.1 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. There have been a lot

of such fixes, both in the database and in the XML encoding since the last ballot cycle (committee level ballot).

2.9.2 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:

```
<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 (although formally existing).

2.9.3 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 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.

3.1.1.1 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.

3.1.1.1.1 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, fields, components and subcomponents 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.) The XML encoding recommended here will result in messages that are approximately five to ten times longer, although this estimate has yet to be subjected to rigorous testing nor is officially published.

Message length is an issue for bandwidth requirements but also for long term archiving the original messages (as done e. g. by some healthcare providers). It should be mentioned that the use of compression is considered as a solution to deal with both bandwidth and archiving issues. It's a matter of fact, that using appropriate compression algorithms XML instances compress very well. This is for example because starting and ending tags are almost the same sequence of characters. However, describing compression methods is out of scope of this specification.

An example: the large messages shown in section 3.2.4.2 show 1,426 bytes for the vertical bar encoding and 6,442 bytes for the v2.xml encoding (4.5 times larger). After compression the v2.xml message is 1,714 bytes long. That is about 20% larger than the uncompressed vertical bar variant.

3.1.1.1.2 Structural Complexity

Krueger [rfKrueger] 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.

3.1.1.1.3 Localization Issues

The HL7 standard describes the responsibilities for parties sending and receiving HL7 messages (see also section 2.7.1 of this specification). These responsibilities enable also 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. Please refer to section XXX of the informative appendix for further information.

3.1.1.1.4 "Looseness" of a DTD, Conformance Testing of Other Business Rules

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, it is also validating that a message is conformant to those HL7 rules that are explicitly represented in the XML schema.

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 and define more constraints than 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 or other business rules to be conformant to).

3.1.1.1.5 Automation Considerations

The XML representation of HL7 messages presented here is algorithmically derived directly from the HL7 Database. The algorithms used for this specification to derive the database excerpts and to create schemas are presented in the informative appendix.

The automatic creation process was considered in order to avoid handcrafting of the schemas, which would have involved a certain danger of introducing errors. Furthermore, necessary refinement of definitions during the development process could be achieved much easier.

3.1.1.2 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 with the message structure IDs.

Version	Chapter	HL7 Table
v2.3.1	2.24.1.9	Message structure table 0354
v2.4	2.17.3	Message structure table 0354

3.1.1.3 List of Schemas and DTDs

This specification provides two sets of schema definition files: *normative* XML Schema definitions (xsd) –, and *informative (non-normative)* Document Type Definitions (dtd), as shown by the following table. There is a set for each HL7 version supported by the v2.xml specification (v2.3.1 and v2.4).

Please note that the use of XML schemas is recommended by HL7 for all normative specifications. DTDs are not part of the normative specification, but rather added as an informative appendix in order to support vendors with migration from DTDs to XML schemas. The class of message instances validated by the normative schemas distributed as part of this specification equals the class of message instances validated by the informative DTDs.

Schema (in directory xsd) <i>normative</i>	DTD (in directory dtd) <i>informative</i>	Description
<i>MessageStructureID</i> .xsd	<i>MessageStructureID</i> .dtd	A set of many schemas each containing the schema definitions for a specific message structure specified by <i>MessageStructureID</i> , for example ADT_A01.* contains the definitions for the ADT A01 message structure, ADT_A02.* for ADT A02 and so forth. The schemas import the segments definitions.
messages.xsd	messages.dtd	schema containing all message definitions to be used in batch messages (refer to section 2.7.4)
segments.xsd	segments.dtd	schema for all segment definitions, imports fields definitions
fields.xsd	fields.dtd	schema for all field definitions, imports data type definitions
datatypes.xsd	datatypes.dtd	schema for all data type definitions for v2

An XML instance of a specific message should refer to the corresponding schema. The following examples show a DTD and a schema reference within a v2.xml XML message instance fragment. In both cases <ADT_A01> is the root element of the instance.

```
<ADT_A01
  xmlns="urn:hl7-org:v2xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:hl7-org:v2xml ADT_A01.xsd">
  <MSH>...
```

```
<!DOCTYPE ADT_A01 SYSTEM "ADT_A01.dtd">
<ADT_A01>
  <MSH>...
```

3.2 Informative Appendix

3.2.1 Localization of messages

The HL7 Standard describes the responsibilities for parties sending and receiving HL7 messages (see also section 2.7.1 of this specification). These responsibilities enable also exchange of messages that contain localizations (or local variations or Z-segments, Z-fields etc.). As a consequence to these requirements, the v2.xml specification fulfills the appropriate design considerations (see section

3.1.1.1.3). Examples to follow show how to introduce local variations. This has to be done in concordance to the v2.x standard itself, i. e. where localizations are appropriate and allowed.

The mechanism shown here is a non-normative recommendation.

3.2.1.1 DTD Localization

The sender includes differences from the standard DTD in the internal subset of a message instance. This is enabled by the fact, that all content models in the v2.xml DTDs are crafted as parameter entities, which can then be redefined in the internal subset.

Consider a sequence of segments in a message FOO as ABC, DEF, GHI, JKL. The corresponding DTD fragment would look like this:

```
<!ENTITY % FOO.CONTENT "( ABC, DEF, GHI, JKL )">
<!ELEMENT FOO %FOO.CONTENT;>
```

Please note that the content model is designed as an entity, the corresponding element FOO refers to that entity rather than defining it directly. All content models in the v2.xml DTDs are crafted that way.

A corresponding instance message fragment would look like this:

```
<FOO>
  <ABC>...</ABC>
  <DEF>...</DEF>
  <GHI>...</GHI>
  <JKL>...</JKL>
</FOO>
```

Expanding the FOO message by adding a local Z segment (with own field definitions not mentioned in detail here), let's say the new content model should be ABC, DEF, ZZZ, GHI, JKL. To achieve this, the entity of the content model describing FOO can be changed in the internal subset like this:

```
<!DOCTYPE FOO SYSTEM "FOO.dtd" [
  ---this includes the original FOO content model as shown above
  <!ENTITY % FOO.CONTENT "( ABC, DEF, ZZZ, GHI, JKL )">
  ---this modifies (expands) the FOO content model entity
  <!ELEMENT FOO.LOCAL %FOO.CONTENT;>
  ---and this finally defines a new element containing the new definitions
  ...
  <!ELEMENT ZZZ (---local definitions here to come---)> ]>
```

Now, the FOO message is redefined by the local modification. 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 a HL7 standard DTD.

The corresponding local message instance would look like this:

```
<FOO.LOCAL>
  <ABC>...</ABC>
  <DEF>...</DEF>
  <ZZZ>...</ZZZ>
  <GHI>...</GHI>
  <JKL>...</JKL>
</FOO.LOCAL>
```

3.2.1.2 Schema Localization

The schemas provided in this specification can easily be localized by a similar approach described above for DTDs. This is done by redefinitions of the existing definitions.

Again, consider the original FOO message that might be represented as follows using XML schema:

```

<!-- .. message definition FOO -->
<xsd:element name="FOO">
  <xsd:complexType name="FOO.CONTENT">
    <xsd:sequence>
      <xsd:element ref="ABC" />
      <xsd:element ref="DEF" />
      <xsd:element ref="GHI" />
      <xsd:element ref="JKL" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

By redefinition of the FOO content model and restricting it to whatever the new localized content should be could be done by the XML redefine statement and defining an additional new (root) element FOO.LOCAL that contains the new definitions.

```

<!-- .. message re-definition FOO -->
<xsd:redefine schemaLocation="FOO.xsd">
  <xsd:complexType name="FOO.CONTENT">
    <xsd:complexContent>
      <xsd:restriction base="FOO.CONTENT">
        <xsd:sequence>
          <xsd:element ref="ABC" />
          <xsd:element ref="DEF" />
          <xsd:element ref="ZZZ" />
          <xsd:element ref="GHI" />
          <xsd:element ref="JKL" />
        </xsd:sequence >
      </xsd:restriction >
    </xsd:complexContent>
  </xsd:complexType>
</xsd:redefine>

<xsd:element name="FOO.LOCAL" type="FOO.CONTENT"/>
<xsd:element name="ZZZ" type="xsd:string"/>

```

Now, the FOO message is redefined by the local modification. A new root element is defined called FOO.LOCAL that serves as a localized version of the original FOO root element. The message instances shown in section 3.2.1.1 remain the same.

3.2.2 Extracting Subsets Of The HL7 Database

The following section describes the methods used for extracting information from the HL7 database that were necessary to generate the schemas.

3.2.2.1 Messages and Their Segments

3.2.2.1.1 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	Field Name	Data Type	Description
*	event_code	Text-3	Event-Code
*	hl7_version	Text-8	version number
*	message_type	Text-3	Type of this Message
*	lfd_nr	Integer	consecutive increasing number used for each field within the segment
	seg_code	Text-3	Segment-Code
	repetitional	Yes/No	Repetitional
	optional	Yes/No	Optional

MsgStructIDSegments			
Primary Key	Field Name	Data Type	Description
*	message_structure	Text-7	Message Structure ID
*	hl7_version	Text-8	version number
*	lfd_nr*	Integer	consecutive increasing number used for each field within the segment
	seg_code	Text-3	Segment-Code
	repetitional	Yes/No	Repetitional
	optional	Yes/No	Optional

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

3.2.2.1.2 SQL Query

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

```
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	Field Name	Data Type	Description
*	MsgStruct	Text-7	Message Structure ID
*	lfd_nr	Integer	consecutive increasing number used for 1:n relation
	seg_code	Text-3	Segment-Code
	repetitional	Yes/No	Repetitional
	optional	Yes/No	Optional

The results of the union query are copied into MsgStructUnion.

3.2.2.1.3 Select 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 were 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.

3.2.2.2 Segments, Fields, and Data Types

3.2.2.2.1 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.)

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

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

3.2.2.2.2 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 3.2.3.

3.2.2.3 Data Types and Their Data Type Components

3.2.2.3.1 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.)

DataStructures			
Primary Key	Field Name	Data Type	Description
*	data_structure	Text-20	logical data type
*	hl7_version	Text-8	version number
	Description	Text-80	Description

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

Components			
Primary Key	Field Name	Data Type	Description
*	comp_nr	Long Integer	Component Number (ID)
*	hl7_version	Text-8	Version of HL7
	description	Text-50	Description
	table_id	Long Integer	reference to an assigned Table
	data_type_code	Text-3	Data type

3.2.2.3.2 SQL Query

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

```
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 3.2.3).

3.2.2.4 Summary of Additional Files

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

HL7 Structure	Database excerpt (tab delimited)	Resulting schema
Messages	messages.txt	schema for all message definition
Segments	fields.txt*	schema for all segment definition
Fields	fields.txt	schema for all field definition
Data types	datatypes.txt	schema for all data type definition

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

3.2.3 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 mentioned 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 will be publicly available on the HL7 website.

3.2.4 Examples

3.2.4.1 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.

```

<!-- Data type definitions -->
...
<!--
    PRIMITIVE DATATYPE ID
-->
<xsd:simpleType name="ID">
  <xsd:restriction base="xsd:string" />
</xsd:simpleType>
...
<!--
    COMPOSITE DATATYPE CE
-->
<xsd:complexType name="CE">
  <xsd:sequence>
    <xsd:element ref="CE.1" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="CE.2" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="CE.3" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="CE.4" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="CE.5" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="CE.6" minOccurs="0" maxOccurs="1" />
  </xsd:sequence>
</xsd:complexType>
...

```

```

<!--
  COMPONENT CE.1
-->
<xsd:element name="CE.1" type="ST">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">identifier</xsd:documentation>
    <xsd:appinfo>
      <hl7:Type>ST</hl7:Type>
      <hl7:LongName>identifier</hl7:LongName>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>

```

```

<!-- Field definitions -->
...
<!--
  FIELD MSH.19
-->
<xsd:element name="MSH.19" type="CE">
  <xsd:annotation>
    <xsd:documentation
      xml:lang="en">Principal Language Of Message</xsd:documentation>
    <xsd:appinfo>
      <hl7:Item>693</hl7:Item>
      <hl7:Type>CE</hl7:Type>
      <hl7:LongName>Principal Language Of Message</hl7:LongName>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>

```

```

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

```

```

<!-- Message definitions -->
...
<?xml version = "1.0" ?>
<!--
  v2.xml message definition ACK Version v24
-->
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="urn:hl7-org:v2xml"
  targetNamespace="urn:hl7-org:v2xml">

  <!-- import segment definitions for version v24 -->
  <xsd:import
    namespace="urn:hl7-org:v2xml" schemaLocation="segments.xsd"/>

  <!--
    MESSAGE ACK
  -->
  <!-- .. message definition ACK -->
  <xsd:element name="ACK">
    <xsd:complexType name="ACK.CONTENT">
      <xsd:sequence>
        <xsd:element ref="MSH" minOccurs="1" maxOccurs="1" />
        <xsd:element ref="MSA" minOccurs="1" maxOccurs="1" />
        <xsd:element ref="ERR" minOccurs="0" maxOccurs="1" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>

```

```

<!-- 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 "%ST;">
<!ELEMENT CE.1 %CE.1.CONTENT;>
<!-- Type='CE_0002' Table='2' LongName='identifier' -->

```

```

<!-- Field definitions -->
...
<!--
  FIELD MSH.19
-->
<!ENTITY % MSH.19.CONTENT "%CE;">
<!ELEMENT MSH.19 %MSH.19.CONTENT;>
<!-- Item='693' Type='CE' LongName='Principal Language Of Message' -->

```

```

<!-- 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*)" ">
<!ELEMENT MSH %MSH.CONTENT;>

```

```

<!-- Message definitions -->
...
<!--
    v2.xml message definition ACK Version v24
--><!-- import segment definitions -->
<!ENTITY % HL7v24-segments PUBLIC
    "-//HL7//DTD HL7 v24 segments definitions//EN
    segments.dtd">
    %HL7v24-segments;

<!--
    MESSAGE ACK
-->
<!-- .. message definition ACK -->
<!ENTITY % ACK.CONTENT "(MSH,MSA,ERR?)" ">
<!ELEMENT ACK %ACK.CONTENT;>

```

3.2.4.2 V2 and v2.xml Example Messages

3.2.4.2.1 Short Example

```

MSH|^~\&|GHH LAB|ELAB-3|GHH OE|BLDG4|200202150930||ORU^R01|CNTRL-3456|P|2.4
PID|||555-44-4444||EVERYWOMAN^EVE^E^^^L|JONES|196203520|F||
|153 FERNWOOD DR.^STATESVILLE^OH^35292|| (206) 3345232
|(206) 752-121|||AC555444444||67-A4335^OH^20030520
OBR|1|845439^GHH OE|1045813^GHH LAB|1554-5^GLUCOSE|||200202150730|||
|555-55-5555^PRIMARY^PATRICIA P^^^MD^LEVEL SEVEN HEALTHCARE, INC.|
|||F|||444-44-4444^HIPPOCRATES^HOWARD H^^^MD
OBX|1|SN|1554-5^GLUCOSE^POST 12H CFST:MCNC:PT:SER/PLAS:QN||^182|mg/dl|70-105|H||F

```

```

<ORU_R01
  xmlns="urn:hl7-org:v2xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:hl7-org:v2xml ORU_R01.xsd">
  <MSH>
    <MSH.1>|</MSH.1>
    <MSH.2>^~\&|</MSH.2>
    <MSH.3>
      <HD.1>GHH LAB</HD.1>
    </MSH.3>
    <MSH.4>
      <HD.1>ELAB-3</HD.1>
    </MSH.4>
    <MSH.5>
      <HD.1>GHH OE</HD.1>
    </MSH.5>
    <MSH.6>
      <HD.1>BLDG4</HD.1>
    </MSH.6>
    <MSH.7>
      <TS.1>200202150930</TS.1>
    </MSH.7>
    <MSH.9>

```

```

    <CM MSG.1>ORU</CM MSG.1>
    <CM MSG.2>R01</CM MSG.2>
  </MSH.9>
  <MSH.10>CNTRL-3456</MSH.10>
  <MSH.11>
    <PT.1>P</PT.1>
  </MSH.11>
  <MSH.12>
    <VID.1>2.4</VID.1>
  </MSH.12>
</MSH>
<ORU_R01.OBSERVATIONAL_REPORT>
  <ORU_R01.PATIENT>
    <PID>
      <PID.3>
        <CX.1>555-44-4444</CX.1>
      </PID.3>
      <PID.5>
        <XPN.1>
          <FN.1>EVERYWOMAN</FN.1>
        </XPN.1>
        <XPN.2>EVE</XPN.2>
        <XPN.3>E</XPN.3>
        <XPN.7>L</XPN.7>
      </PID.5>
      <PID.6>
        <XPN.1>
          <FN.1>JONES</FN.1>
        </XPN.1>
      </PID.6>
      <PID.7>
        <TS.1>196203520</TS.1>
      </PID.7>
      <PID.8>F</PID.8>
      <PID.11>
        <XAD.1>
          <SAD.1>153 FERNWOOD DR.</SAD.1>
        </XAD.1>
        <XAD.3>STATESVILLE</XAD.3>
        <XAD.4>OH</XAD.4>
        <XAD.5>35292</XAD.5>
      </PID.11>
      <PID.13>
        <XTN.1>(206) 3345232</XTN.1>
      </PID.13>
      <PID.14>
        <XTN.1>(206) 752-121</XTN.1>
      </PID.14>
      <PID.18>
        <CX.1>AC5554444444</CX.1>
      </PID.18>
      <PID.20>
        <DLN.1>67-A4335</DLN.1>
        <DLN.2>OH</DLN.2>
        <DLN.3>20030520</DLN.3>
      </PID.20>
    </PID>
  </ORU_R01.PATIENT>
  <ORU_R01.ORDER_OBSERVATION>
    <OBR>
      <OBR.1>1</OBR.1>
      <OBR.2>
        <EI.1>845439</EI.1>
    </OBR>
  </ORU_R01.ORDER_OBSERVATION>
</ORU_R01.OBSERVATIONAL_REPORT>

```

```

      <EI.2>GHH OE</EI.2>
    </OBR.2>
  <OBR.3>
    <EI.1>1045813</EI.1>
    <EI.2>GHH LAB</EI.2>
  </OBR.3>
  <OBR.4>
    <CE.1>1554-5</CE.1>
    <CE.2>GLUCOSE</CE.2>
  </OBR.4>
  <OBR.7>
    <TS.1>200202150730</TS.1>
  </OBR.7>
  <OBR.16>
    <XCN.1>555-55-5555</XCN.1>
    <XCN.2>
      <FN.1>PRIMARY</FN.1>
    </XCN.2>
    <XCN.3>PATRICIA P</XCN.3>
    <XCN.7>MD</XCN.7>
    <XCN.9>
      <HD.1>LEVEL SEVEN HEALTHCARE, INC.</HD.1>
    </XCN.9>
  </OBR.16>
  <OBR.25>F</OBR.25>
  <OBR.32>
    <CM_NDL.1>
      <CN.1>444-44-4444</CN.1>
      <CN.2>
        <FN.1>HIPPOCRATES</FN.1>
      </CN.2>
      <CN.3>HOWARD H</CN.3>
      <CN.7>MD</CN.7>
    </CM_NDL.1>
  </OBR.32>
</OBR>
<ORU_R01.OBSERVATION_RESULT>
  <OBX>
    <OBX.1>1</OBX.1>
    <OBX.2>SN</OBX.2>
    <OBX.3>
      <CE.1>1554-5</CE.1>
      <CE.2>GLUCOSE</CE.2>
      <CE.3>POST 12H CFST:MCNC:PT:SER/PLAS:QN</CE.3>
    </OBX.3>
    <OBX.5>182</OBX.5>
    <OBX.6>
      <CE.1>mg/dl</CE.1>
    </OBX.6>
    <OBX.7>70-105</OBX.7>
    <OBX.8>H</OBX.8>
    <OBX.11>F</OBX.11>
  </OBX>
</ORU_R01.OBSERVATION_RESULT>
</ORU_R01.ORDER_OBSERVATION>
</ORU_R01.OBSERVATIONAL_REPORT>
</ORU_R01>

```



```

<EVN.3>
  <TS.1>199901101400</TS.1>
</EVN.3>
<EVN.4>01</EVN.4>
<EVN.6>
  <TS.1>199901101410</TS.1>
</EVN.6>
</EVN>
<PID>
  <PID.3>
    <CX.1>191919</CX.1>
    <CX.4>
      <HD.2>GENHOS</HD.2>
    </CX.4>
    <CX.5>MR</CX.5>
  </PID.3>
  <PID.3>
    <CX.1>371-66-9256</CX.1>
    <CX.4>
      <HD.2>USSSA</HD.2>
    </CX.4>
    <CX.5>SS</CX.5>
  </PID.3>
  <PID.5>
    <XPN.1>
      <FN.1>MASSIE</FN.1>
    </XPN.1>
    <XPN.2>JAMES</XPN.2>
    <XPN.3>A</XPN.3>
  </PID.5>
  <PID.7>
    <TS.1>19560129</TS.1>
  </PID.7>
  <PID.8>M</PID.8>
  <PID.11>
    <XAD.1>
      <SAD.1>171 ZOBERLEIN</SAD.1>
    </XAD.1>
    <XAD.3>ISHPEMING</XAD.3>
    <XAD.4>MI</XAD.4>
    <XAD.5>49849</XAD.5>
    <XAD.6>""</XAD.6>
  </PID.11>
  <PID.13>
    <XTN.1>(900)485-5344</XTN.1>
  </PID.13>
  <PID.14>
    <XTN.1>(900)485-5344</XTN.1>
  </PID.14>
  <PID.16>
    <CE.1>S</CE.1>
    <CE.3>HL70002</CE.3>
  </PID.16>
  <PID.17>
    <CE.1>C</CE.1>
    <CE.3>HL70006</CE.3>
  </PID.17>
  <PID.18>
    <CX.1>1019992510199925</CX.1>
    <CX.4>
      <HD.2>GENHOS</HD.2>
    </CX.4>
    <CX.5>AN</CX.5>

```

```

    </PID.18>
  </PID>
  <NK1>
    <NK1.1>1</NK1.1>
    <NK1.2>
      <XPN.1>
        <FN.1>MASSIE</FN.1>
      </XPN.1>
      <XPN.2>ELLEN</XPN.2>
    </NK1.2>
    <NK1.3>
      <CE.1>SPOUSE</CE.1>
      <CE.3>HL70063</CE.3>
    </NK1.3>
    <NK1.4>
      <XAD.1>
        <SAD.1>171 ZOBERLEIN</SAD.1>
      </XAD.1>
      <XAD.3>ISHPEMING</XAD.3>
      <XAD.4>MI</XAD.4>
      <XAD.5>49849</XAD.5>
      <XAD.6>""</XAD.6>
    </NK1.4>
    <NK1.5>
      <XTN.1>(900)485-5344</XTN.1>
    </NK1.5>
    <NK1.6>
      <XTN.1>(900)545-1234</XTN.1>
    </NK1.6>
    <NK1.6>
      <XTN.1>(900)545-1200</XTN.1>
    </NK1.6>
    <NK1.7>
      <CE.1>EC1</CE.1>
      <CE.2>FIRST EMERGENCY CONTACT</CE.2>
      <CE.3>HL70131</CE.3>
    </NK1.7>
  </NK1>
  <NK1>
    <NK1.1>2</NK1.1>
    <NK1.2>
      <XPN.1>
        <FN.1>MASSIE</FN.1>
      </XPN.1>
      <XPN.2>MARYLOU</XPN.2>
    </NK1.2>
    <NK1.3>
      <CE.1>MOTHER</CE.1>
      <CE.3>HL70063</CE.3>
    </NK1.3>
    <NK1.4>
      <XAD.1>
        <SAD.1>300 ZOBERLEIN</SAD.1>
      </XAD.1>
      <XAD.3>ISHPEMING</XAD.3>
      <XAD.4>MI</XAD.4>
      <XAD.5>49849</XAD.5>
      <XAD.6>""</XAD.6>
    </NK1.4>
    <NK1.5>
      <XTN.1>(900)485-5344</XTN.1>
    </NK1.5>
    <NK1.6>

```

```

    <XTN.1>(900)545-1234</XTN.1>
  </NK1.6>
  <NK1.6>
    <XTN.1>(900)545-1200</XTN.1>
  </NK1.6>
  <NK1.7>
    <CE.1>EC2</CE.1>
    <CE.2>SECOND EMERGENCY CONTACT</CE.2>
    <CE.3>HL70131</CE.3>
  </NK1.7>
</NK1>
<NK1>
  <NK1.1>3</NK1.1>
</NK1>
<NK1>
  <NK1.1>4</NK1.1>
  <NK1.4>
    <XAD.1>
      <SAD.1>123 INDUSTRY WAY</SAD.1>
    </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>
  <NK1.7>
    <CE.1>EM</CE.1>
    <CE.2>EMPLOYER</CE.2>
    <CE.3>HL70131</CE.3>
  </NK1.7>
  <NK1.8>19940605</NK1.8>
  <NK1.10>PROGRAMMER</NK1.10>
  <NK1.13>
    <XON.1>ACME SOFTWARE COMPANY</XON.1>
  </NK1.13>
</NK1>
<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>
      <FN.1>ADDISON, JAMES</FN.1>
    </XCN.2>
  </PV1.7>
  <PV1.8>
    <XCN.1>0148</XCN.1>
    <XCN.2>
      <FN.1>ADDISON, JAMES</FN.1>
    </XCN.2>
  </PV1.8>
  <PV1.10>AMB</PV1.10>
  <PV1.17>
    <XCN.1>0148</XCN.1>
    <XCN.2>
      <FN.1>ADDISON, JAMES</FN.1>
    </XCN.2>
  </PV1.17>

```

```

<PV1.18>S</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><TS.1>199501101410</TS.1></PV1.44>
</PV1>
<PV2>
  <PV2.8><TS.1>199901101400</TS.1></PV2.8>
  <PV2.33><TS.1>199901101400</TS.1></PV2.33>
</PV2>
<ROL>
  <ROL.2>AD</ROL.2>
  <ROL.3>
    <CE.1>CP</CE.1>
    <CE.3>HL70443</CE.3>
  </ROL.3>
  <ROL.4>
    <XCN.1>0148</XCN.1>
    <XCN.2>
      <FN.1>ADDISON, JAMES</FN.1>
    </XCN.2>
  </ROL.4>
</ROL>
<OBX>
  <OBX.2>NM</OBX.2>
  <OBX.3>
    <CE.1>3141-9</CE.1>
    <CE.2>BODY WEIGHT</CE.2>
    <CE.3>LN</CE.3>
  </OBX.3>
  <OBX.5>62</OBX.5>
  <OBX.6>
    <CE.1>kg</CE.1>
  </OBX.6>
  <OBX.11>F</OBX.11>
</OBX>
<OBX>
  <OBX.2>NM</OBX.2>
  <OBX.3>
    <CE.1>3137-7</CE.1>
    <CE.2>HEIGHT</CE.2>
    <CE.3>LN</CE.3>
  </OBX.3>
  <OBX.5>190</OBX.5>
  <OBX.6>
    <CE.1>cm</CE.1>
  </OBX.6>
  <OBX.11>F</OBX.11>
</OBX>
<DG1>
  <DG1.1>1</DG1.1>
  <DG1.2>ACODE</DG1.2>
  <DG1.3>
    <CE.1>0815</CE.1>
    <CE.2>BIOPSY</CE.2>
    <CE.3>ACODE</CE.3>
  </DG1.3>
  <DG1.6>00</DG1.6>
</DG1>

```

```

<GT1>
  <GT1.1>1</GT1.1>
  <GT1.3>
    <XPN.1>
      <FN.1>MASSIE</FN.1>
    </XPN.1>
    <XPN.2>JAMES</XPN.2>
    <XPN.3>""</XPN.3>
    <XPN.4>""</XPN.4>
    <XPN.5>""</XPN.5>
    <XPN.6>""</XPN.6>
  </GT1.3>
  <GT1.5>
    <XAD.1>
      <SAD.1>171 ZOBERLEIN</SAD.1>
    </XAD.1>
    <XAD.3>ISHPEMING</XAD.3>
    <XAD.4>MI</XAD.4>
    <XAD.5>49849</XAD.5>
    <XAD.6>""</XAD.6>
  </GT1.5>
  <GT1.6>
    <XTN.1>(900) 485-5344</XTN.1>
  </GT1.6>
  <GT1.7>
    <XTN.1>(900) 485-5344</XTN.1>
  </GT1.7>
  <GT1.11>
    <CE.1>SE</CE.1>
    <CE.2>SELF</CE.2>
    <CE.3>HL70063</CE.3>
  </GT1.11>
  <GT1.12>371-66-925</GT1.12>
  <GT1.16>
    <XPN.1>
      <FN.1>MOOSES AUTO CLINIC</FN.1>
    </XPN.1>
  </GT1.16>
  <GT1.17>
    <XAD.1>
      <SAD.1>171 ZOBERLEIN</SAD.1>
    </XAD.1>
    <XAD.3>ISHPEMING</XAD.3>
    <XAD.4>MI</XAD.4>
    <XAD.5>49849</XAD.5>
    <XAD.6>""</XAD.6>
  </GT1.17>
  <GT1.18>
    <XTN.1>(900) 485-5344</XTN.1>
  </GT1.18>
</GT1>
<ADT_A01.INSURANCE>
  <IN1>
    <IN1.1>0</IN1.1>
    <IN1.2>
      <CE.1>0</CE.1>
      <CE.2>HL70072</CE.2>
    </IN1.2>
    <IN1.3>
      <CX.1>BC1</CX.1>
    </IN1.3>
    <IN1.4>
      <XON.1>BLUE CROSS</XON.1>
  </IN1>

```

```

</IN1.4>
<IN1.5>
  <XAD.1>
    <SAD.1>171 ZOBERLEIN</SAD.1>
  </XAD.1>
  <XAD.3>ISHPEMING</XAD.3>
  <XAD.4>M149849</XAD.4>
  <XAD.5>""</XAD.5>
</IN1.5>
<IN1.7>
  <XTN.1>(900)485-5344</XTN.1>
</IN1.7>
<IN1.8>90</IN1.8>
<IN1.14>
  <CM_AUI.1>50 OK</CM_AUI.1>
</IN1.14>
</IN1>
</ADT_A01.INSURANCE>
</ADT_A01>

```

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>
- [rfXML] Extensible Markup Language (XML) 1.0. 2nd Edition W3C Recommendation <http://www.w3.org/TR/REC-xml>
- [rfXMLSchema] XML Schema, W3C Recommendation May 2, 2001 <http://www.w3.org/XML/Schema>, <http://www.w3.org/TR/xmlschema-0/>, <http://www.w3.org/TR/xmlschema-1/>, <http://www.w3.org/TR/xmlschema-2/>
- [rfINFO] 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.
- [rfCEN1993] European Committee for Standardization / Technical Committee 251 – Medical Informatics. Investigation of Syntaxes for Existing Interchange Formats to be used in Healthcare. CEN/TC251. January 1993.
- [rfDolin1997] Dolin RH, Alschuler L, Bray T, Mattison JE. SGML as a message interchange format in healthcare. JAMIA Fall Symposium Supplement 1997: 635-9.
- [rfDolin1998] Dolin RH, Rishel W, Biron PV, Spinosa J, Mattison JE. SGML and XML as interchange formats for HL7 messages. JAMIA Fall Symposium Supplement 1998.
- [rfOemig1996] Oemig F, Dudeck J. Problems in developing a comprehensive HL7 database. AMIA Fall Symposium 1996.
- [rfOemig] Frank Oemig's Home Page <http://www.sr-solutions.de/oemig/>
- [Krueger] Krueger G. A structured approach to HL7 application development. 1996. <http://www.gkrueger.com/other/hl7/>

- [rfHL7v3ITS] HL7 Version 3 – XML ITS, see HL7 Version 3 (Draft)
<http://www.hl7.org>
- [rfXPATH] XML Path Language (XPath) Version 1.0, W3C Recommendation
<http://www.w3.org/TR/xpath>