



**ANSI/HL7 V2 XML, R2-2012**

**June 19, 2012**

HL7 Version 2: XML Encoding Rules, Release 2  
(revision of ANSI/HL7 V2 XML-2003 (R2010))

# HL7 Version 2 XML Encoding Rules, Release 2



**HL7 Implementable Technology Specifications Work Group Cochairs:**

Paul Knapp, Paul Knapp Consulting, inc.  
Dale Nelson, Lantana Consulting Group  
Andy Stechishin, CANA Software & Services, Ltd.

**Conformance & Guidance for Implementation/Testing Work Group Cochairs:**

Wendy Huang, Canada Health Infoway Inc.  
Frank Oemig, PhD, HL7 Germany  
Robert Snelick, National Institute of Standards & Technology

Copyright © 2012 Health Level Seven International ® ALL RIGHTS RESERVED. The reproduction of this material in any form is strictly forbidden without the written permission of the publisher. HL7 International and Health Level Seven are registered trademarks of Health Level Seven International. Reg. U.S. Pat & TM Off.

**IMPORTANT NOTES:**

HL7 licenses its standards and select IP free of charge. **If you did not acquire a free license from HL7 for this document**, you are not authorized to access or make any use of it. To obtain a free license, please visit <http://www.HL7.org/implement/standards/index.cfm>.

**If you are the individual that obtained the license for this HL7 Standard, specification or other freely licensed work (in each and every instance "Specified Material")**, the following describes the permitted uses of the Material.

**A. HL7 INDIVIDUAL, STUDENT AND HEALTH PROFESSIONAL MEMBERS**, who register and agree to the terms of HL7's license, are authorized, without additional charge, to read, and to use Specified Material to develop and sell products and services that implement, but do not directly incorporate, the Specified Material in whole or in part without paying license fees to HL7.

INDIVIDUAL, STUDENT AND HEALTH PROFESSIONAL MEMBERS wishing to incorporate additional items of Special Material in whole or part, into products and services, or to enjoy additional authorizations granted to HL7 ORGANIZATIONAL MEMBERS as noted below, must become ORGANIZATIONAL MEMBERS of HL7.

**B. HL7 ORGANIZATION MEMBERS**, who register and agree to the terms of HL7's License, are authorized, without additional charge, on a perpetual (except as provided for in the full license terms governing the Material), non-exclusive and worldwide basis, the right to (a) download, copy (for internal purposes only) and share this Material with your employees and consultants for study purposes, and (b) utilize the Material for the purpose of developing, making, having made, using, marketing, importing, offering to sell or license, and selling or licensing, and to otherwise distribute, Compliant Products, in all cases subject to the conditions set forth in this Agreement and any relevant patent and other intellectual property rights of third parties (which may include members of HL7). No other license, sublicense, or other rights of any kind are granted under this Agreement.

**C. NON-MEMBERS**, who register and agree to the terms of HL7's IP policy for Specified Material, are authorized, without additional charge, to read and use the Specified Material for evaluating whether to implement, or in implementing, the Specified Material, and to use Specified Material to develop and sell products and services that implement, but do not directly incorporate, the Specified Material in whole or in part.

NON-MEMBERS wishing to incorporate additional items of Specified Material in whole or part, into products and services, or to enjoy the additional authorizations granted to HL7 ORGANIZATIONAL MEMBERS, as noted above, must become ORGANIZATIONAL MEMBERS of HL7.

Please see <http://www.HL7.org/legal/ippolicy.cfm> for the full license terms governing the Material.

# HL7 Version 2.x: XML Encoding Syntax Release 2

## ITS WG + CGIT WG

Editor Rel.2:	Frank Oemig, PhD (frank@oemig.de) Agfa Healthcare GmbH, HL7 Germany
ITS Co-Chairs:	Paul Knapp (pknapp@pknapp.com) Knapp Consulting Inc.  Andy Stechishin (andy.stechishin@gmail.com) CANA Software and Services  Dale Nelson (dale.nelson@squaretrends.com) Squaretrends LLC
CGIT Co-Chairs:	Frank Oemig, PhD (frank@oemig.de) Agfa Healthcare GmbH, HL7 Germany  Robert Snelick (robert.snelick@nist.gov) National Institute of Standard & Technology  Wendy Huang (whuang@infoway-inforoute.ca) Canada Health Infoway  Ioana Singureanu (ioana.singureanu@gmail.com) Eversolve LLC
Sponsoring Committee:	Implementable Technology Specifications (ITS)
List Server:	its@lists.hl7.org

## v2.xml Contents

HL7 Version 2.x: XML Encoding Syntax Release 2 .....	3
v2.xml Contents .....	4
Preface .....	5
Acknowledgements .....	5
Remarks for this specification .....	6
1. Introduction .....	6
1.1. Background.....	6
1.2. Benefits from Using XML as an Alternative v2 Interchange Format .....	7
1.3. XML representation derivation from HL7 Database .....	7
1.4. Scope for HL7 Version 2 .....	8
1.5. Version 2 Message Definitions .....	8
1.5.1. Version 2 Hierarchical Message Structure Overview .....	8
1.5.2. Abstract Message Syntax Definitions .....	9
2. Specification .....	10
2.1. Introduction to the XML Representation .....	10
2.2. A First Example .....	10
2.3. Message Identification and Trigger Events .....	11
2.3.1. Message Structure IDs .....	11
2.4. Segments .....	12
2.4.1. Optional/Repeating Groups of Segments.....	12
2.4.2. Choice Groups of Segments .....	15
2.5. Fields .....	16
2.6. Data Types .....	18
2.6.1. Primitive Data Types .....	18
2.6.2. Composite Data Types.....	18
2.6.3. Wildcard .....	21
2.6.4. CM Data Types.....	21
2.7. Processing Rules for v2.xml Messages.....	21
2.7.1. XML Application Processing Rules .....	21
2.7.2. Inter-version Backward Compatibility .....	21
2.7.3. Message Fragmentation and Continuation.....	21
2.7.4. Batch Messages .....	21
2.7.5. Message Delimiters .....	22
2.7.6. Delete Indicators, Empty Values .....	22
2.7.7. Repetition of Segment Groups, Segments and Fields.....	23
2.7.8. Escape Character Sequences Used in v2 Data Types .....	23
2.7.9. Message Building Rules .....	24
2.7.10. Special Characters in Schemas .....	25
2.8. Translating Between Standard Encoding and XML Encoding .....	25
3. Appendix .....	26
3.1. Normative Appendix .....	26
3.1.1. List of Messages With Equal Message Structures .....	26
3.1.2. List of Schemas .....	26
3.1.3. Localization of messages.....	27
3.2. Informative Appendix.....	28
3.2.1. Design Considerations .....	28
3.2.2. Extracting Subsets of the HL7 Database .....	30
3.2.3. Options .....	35
3.2.4. Algorithms .....	35
3.2.5. Examples .....	35
3.3. References .....	48

## Preface

This document supersedes Release 1 and contains additional specifications to accommodate new features introduced beginning HL7 Version 2.3.1, e.g. the use of choices within message structures. As of the time of this writing the current version is v2.7. This document is valid for all v2.x versions which have passed ballot. Chapter 2 of the HL7 Version 2.3.1 and 2.7 [rfHL7v231, rfHL7v27] 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 Figure 1). The objective of this document is to present alternate encoding rules for HL7 Version 2.3.1 to 2.7 messages (and a mechanism for determining alternate encoding rules for subsequent HL7 2.x versions) based on the Extensible Markup Language XML [rfXML] that could be used in environments where senders and receivers both understand XML.

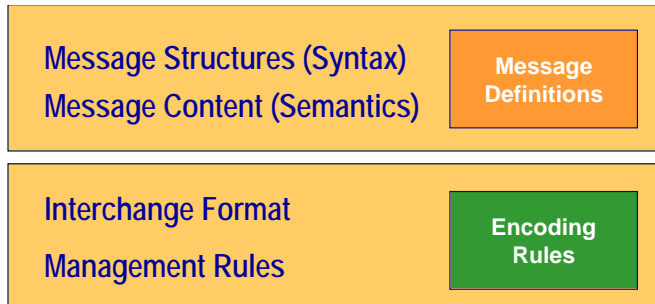


Figure 1: The standard specification specifies message definitions and encoding rules.

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.x 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.

In principle, many XML encodings could serve as alternate messaging syntaxes for HL7 Version 2.x messages. This document describes the one suggested and standardized by HL7. It 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. Due to their greater expressiveness, schemas are the preferred way to describe a set of constraints on message instances. The outdated Document Type Definitions (DTDs) are not addressed any more. The algorithms used for this specification to derive the database excerpts and to create schemas are also presented in the informative appendix.

This document is the normative successor of the first release (2003) and 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.

This document assumes a basic understanding of HL7 version 2. However, some background information has been included to aid those without version 2 experience.

## Acknowledgements

This document is the second release of this specification to capture enhancements to the standard. As such, I wish to thank Kai Heitmann who has written the first release.

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, and 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, Andrew Hinchley, Dennis Janssen. Special thanks for his support to Tom de Jong.

Thanks also to all members of the ITS Work Group and the InM Work Group for their input during the development process.

## Remarks for 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]
- XML Namespace [rfXMLnamesapce]

### Accompanying Material

- In addition to this specification, a set of XML Schemas, hereafter called “schemas” in general, is provided. They are the work product of this specification. Please refer to section “3.1.2. List of Schemas” for further details.
- The use of XML schema ([rfXMLSchema], a W3C recommendation since May 2001) is recommended by HL7 for all normative specifications. The schemas are not part of the normative specification, but rather added as an informative appendix.
- Several example XML message instances are also part of the accompanying material.

### Subject to technical corrections

- The narrative segment group names described in section 2.4.1 and represented in the schema definitions are drawn from the v2.5 first membership ballot. Prior to v2.5, group names were neither present in the database nor in the specification. This specification makes use of these group names even for the schemas for v2.4 and v2.3.1. The group names are not yet finalized (balloted) by the date of this specification. There will be technical corrections to the schema definitions as soon as normative segment group names are finalized in the original standard work.
- Character set switching as described in chapter 2 of the v2.x standard cannot be addressed in XML. There will be a workaround solution for the v3 XML ITS that is not yet completely determined. The v2.xml ITS will use the same mechanism. This is considered to be a technical correction.

### Disclaimer

The reader is reminded that both examples and XML schema fragments presented within the document for illustrating purposes are informative and do not form a part of the normative content.

## 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 2<sup>nd</sup> 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. Another reason for a normative XML syntax is to support future Claims Attachment messages, which are currently using v2.4 encoding.

Enhancing v2.x even further with v2.6 and v2.7 new concepts have been introduced which require an enhancement of this specification.

This document stays with the original strategy for the representation of XML instances for backward compatibility.

## 1.2. Benefits from Using XML as an Alternative v2 Interchange Format

There are several benefits using XML as an interchange format.

The ability to explicitly represent an HL7 requirement in XML confers the ability to parse and validate messages with any 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 essential.

Frequently, a typical healthcare messaging application includes an in-house developed parser (message reader) and generator (message writer) to process traditional (“vertical bar” encoded) HL7 messages with an almost certain negative impact on development and maintenance costs. The only alternative to in house tool development which quite often is not implemented correctly and completely is to choose from among the limited but often expensive commercial tool sets. Increasing, the traditional encoding often contributes to the isolation of healthcare from the generic data interchange approaches used by other business areas. Adoption of across the board generic messaging encoding will become critical for cost and error reduction as healthcare and other areas of business increase their daily interactions. Using XML message parsers and generators will undoubtedly help to prepare healthcare for this growing challenge to increase data interchange commonality with other business areas.

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. XML representation derivation from HL7 Database

The XML representation of HL7 messages presented here is algorithmically derived directly from the HL7 Database (see below). This is done to prevent that work has to be done by hand, which often is susceptible to errors. Furthermore deriving the XML representation algorithmically allows generating schemas for future HL7 v2.x versions easily.

Underlying the HL7 2.x messaging Standards is a Microsoft Access database (the “HL7 Database”) that contains a copy of the official definitions of events, messages, segments, fields, data types, data type components, tables, and table values. The database is designed to have the same content and is used to accurately reflect on what is given in the paper based standard documents and, in addition, on what the membership voted on and including technical correction.

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 2.7 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 relational database.

As a consequence, the database is not an identical equivalent to the standard, but the differences are documented and reflected as technical corrections and new proposals.

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.

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]). Most of the problems have been solved with newer releases of the v2.x standard in the meantime. However, the database has been constructed to maintain all versions and perhaps derivations thereof in parallel.

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 technical corrections to the HL7 Database. There have been many 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.

It should be mentioned that the database itself or extracts of the database are not needed in order to implement or use the XML encoding of version 2 messages as described in this specification. The database and its excerpts are used for the schema creation process only. Implementers should be able to develop v2.xml interfaces having only the schemas and the printed version of both this specification and the HL7 standard. Implementers may also choose to hand-generate or adjust existing schemas to reflect localizations such as Z-segments.

## 1.4. Scope for HL7 Version 2

This specification presents XML encoding rules starting with HL7 Version 2.3.1 messages. Former versions of the HL7 Version 2 family of message standards are explicitly not covered, because a construct (MSH-9.3 – Message Structure) needed in this specification is not present in versions prior to v2.3.1. Therefore there is no XML encoding support for Versions prior to v2.3.1.

If a supplier claims conformance for V2 messages in XML the messages must be valid against schemas produced from the HL7 specification by the rules in the v2.xml specification.

## 1.5. Version 2 Message Definitions

### 1.5.1. Version 2 Hierarchical Message Structure Overview

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 XPN data type (extended person name), which has several components that are each comprised of several sub-components in order to express the various parts of a person’s name.

### 1.5.2. Abstract Message Syntax Definitions

Each message is documented in the standard using a 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. Optional/Repeating Groups of Segments”), because they are named. Such segment group names are uppercase (e. g. “PROCEDURE”, “INSURANCE”) and do not contain spaces or other special characters.

ADT^A01^ADT\_A01: ADT Message

Segments	Description	Status	Chapter
MSH	Message Header		2
[ { SFT } ]	Software Segment		3
{ UAC }	User Authentication		3
EVN	Event Type		3
PID	Patient Identification		3
[PD1]	Additional Demographics		3
[ { ARV } ]	Access Restrictions		15
[ { ROL } ]	Role		3
[ { NK1 } ]	Next of Kin /Associated Parties		3
PV1	Patient Visit		3
[ PV2 ]	Patient Visit - Additional Info.		3
[ { ARV } ]	Access Restriction		3
[ { ROL } ]	Role		15
[ { DB1 } ]	Disability Information		3
[ { OBX } ]	Observation/Result		7
[ { AL1 } ]	Allergy Information		3
[ { DG1 } ]	Diagnosis Information		6
[ DRG ]	Diagnosis Related Group		6
[ {	--- PROCEDURE begin		
PR1	Procedures		6
[ {ROL} ]	Role		12
} ]	--- PROCEDURE end		
[ GT1 ]	Guarantor		6
[ {	--- INSURANCE begin		
IN1	Insurance		6
[ IN2 ]	Insurance Additional Info.		6
[ {IN3} ]	Insurance Additional Info. - Cert.		6
[ {ROL} ]	Role		15
} ]	--- INSURANCE end		
[ { ACC } ]	Accident Information		6
[ { UBI } ]	Universal Bill Information		6

Figure 2: Abstract message syntax definition for message type ADT\_A01, drawn from chapter 3 in [rHL7v27] with additional group names interspersed, and adopted to the corresponding style containing group names (colored/shaded 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)
[ ]	0 .. 1
{ }	1 .. unbounded

{[ ]} = [{ ]}	0 .. unbounded
<, > and	1 .. 1 (complexType Choice)
no bracket or brace	1 .. 1

Table 1: Abstract Message Syntax Notations of Cardinality and their corresponding Schema 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. In addition, the XML schema annotation mechanism is used to provide the same information, as represented in the fixed attributes of field and data type definition (please refer to section “2.5. Fields” and “2.6. Data Types” for details).

### 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>^~\&amp;</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>
      <MSG.1>ACK</MSG.1>
      <MSG.3>ACK</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>
      <ELD.1>PID</ELD.1>
      <ELD.2>1</ELD.2>
      <ELD.3>16</ELD.3>
      <ELD.4>
        <CE.1>103</CE.1>
        <CE.2>Table value not found</CE.2>
        <CE.3>HL70357</CE.3>
      </ELD.4>
    </ERR.1>
  </ERR>
</ACK>

```

As is always the case with XML when processed with a validating processor, the extra white space between elements and line breaks (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. Message Identification 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.

All mentioned tables are defined in chapter 2 or 2C of the standard.

### 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 (see also section “3.1.2. List of 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, e. g., ADT^A04^ADT\_A01. This indicates that the ADT message with trigger event A04 has the message structure ID ADT\_A01 (i.e., it has the same sequence and cardinality of segments). All messages with that structure ID are structurally the same, though they differ in 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 section “3.1.1. List of Messages With Equal Message Structures”).

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

<u>ADT^A04^ADT_A01</u>	<u>ADT Message</u>
MSH	Message Header
EVN	Event Type
PID	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|M246|P|2.4|
```

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

The message structure ID is used 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. An 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 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. Optional/Repeating 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. Upon further consideration by both users and implementers alike (per the clarifying reasoning given below), it has become increasingly persuasive to deepen the XML element hierarchy by the addition of 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 2<sup>nd</sup> column in Figure 4 containing “--- 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 message instance 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. Introduction of the explicit segment group names marks a major difference between vertical bar and XML encoding. Furthermore, this allows elements to be accessed in a reasonable manner within an X-Path expression (see [rfXPath]). By this, an application can refer to specific XML items explicitly by name (e.g. ADT\_A01/PROCEDURE/PR1.3/CNE.1) or they can refer to them by position (e.g. ADT\_A01/PROCEDURE/PR1.3/\*[position()=1]). By taking the latter approach, one no longer has to take care what the name of the field, data element or data type is. See also section on 0 data types.

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

Please note: The narrative segment group names where this specification makes use of are neither in the paper version of v2.3.1 nor v2.4. They are drawn from the v2.5 specification.

ADT^A01^ADT\_A01: ADT Message

<u>Segments</u>	<u>Description</u>	<u>Status</u>	<u>Comment</u>	<u>Chapter</u>
MSH	Message Header			2
EVN	Event Type			3
PID	Patient Identification			3
...				
[{	--- PROCEDURE begin			
PR1	Procedures			6
[{ ROL }]	Role			12
}]	--- PROCEDURE end			
...				
[{	--- INSURANCE begin			
IN1	Insurance			6
[ IN2 ]	Insurance Add. Info.			6
[{ IN3 }]	Insurance Add. Info -Cert.			6
[{ ROL }]	Role			12
}]	--- 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. Consequentially, the v2.xml XML schema segment group naming convention has adopted the use of the owning message structure id as a *prefix* for the group name to insure uniqueness in regard to 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>
  <ROL>
  ...
  </ROL>
</ADT_A01.INSURANCE>
...
</ADT_A01>

```

The corresponding schema definition fragment for the ADT\_A01 message is shown below.

```

<!--
  MESSAGE ADT_A01
-->
<!-- .. groups used in message ADT_A01 -->
<xsd:complexType name="ADT_A01.PROCEDURE.CONTENT">
  <xsd:sequence>
    <xsd:element ref="PR1" minOccurs="1" maxOccurs="1" />
    <xsd:element ref="ROL" minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="ADT_A01.PROCEDURE" type="ADT_A01.PROCEDURE.CONTENT"/>
<xsd:complexType name="ADT_A01.INSURANCE.CONTENT">
  <xsd:sequence>
    <xsd:element ref="IN1" minOccurs="1" maxOccurs="1" />
    <xsd:element ref="IN2" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="IN3" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="ROL" minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="ADT_A01.INSURANCE" type="ADT_A01.INSURANCE.CONTENT"/>

<!-- .. message definition ADT_A01 -->
<xsd:complexType name="ADT_A01.CONTENT">
  <xsd:sequence>
    <xsd:element ref="MSH" minOccurs="1" maxOccurs="1" />
    <xsd:element ref="EVN" minOccurs="1" maxOccurs="1" />
    <xsd:element ref="PID" minOccurs="1" maxOccurs="1" />
    <xsd:element ref="PD1" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="ROL" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="NK1" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="PV1" minOccurs="1" maxOccurs="1" />
    <xsd:element ref="PV2" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="ROL" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="DB1" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="OBX" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="AL1" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="DG1" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="DRG" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="ADT_A01.PROCEDURE" minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
-->

```

```

<xsd:element ref="GT1" minOccurs="0" maxOccurs="unbounded" />
<xsd:element ref="ADT_A01.INSURANCE" minOccurs="0" maxOccurs="unbounded"
/>
<xsd:element ref="ACC" minOccurs="0" maxOccurs="1" />
<xsd:element ref="UB1" minOccurs="0" maxOccurs="1" />
<xsd:element ref="UB2" minOccurs="0" maxOccurs="1" />
<xsd:element ref="PDA" minOccurs="0" maxOccurs="1" />
</xsd:sequence>
</xsd:complexType>
<xsd:element name="ADT_A01" type="ADT_A01.CONTENT"/>

```

As an example, the corresponding schema definition fragment for the EVN segment is shown below. Please note that, consistent with the processing rule for v2 whereby receivers are to ignore fields not expected, the schemas will also allow additional elements at the end of a segment.

```

<!--
  SEGMENT EVN
-->
<xsd:complexType name="EVN.CONTENT">
  <xsd:sequence>
    <xsd:element ref="EVN.1" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="EVN.2" minOccurs="1" maxOccurs="1" />
    <xsd:element ref="EVN.3" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="EVN.4" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="EVN.5" minOccurs="0" maxOccurs="unbounded" />
    <xsd:element ref="EVN.6" minOccurs="0" maxOccurs="1" />
    <xsd:element ref="EVN.7" minOccurs="0" maxOccurs="1" />
    <xsd:any processContents="lax" namespace="##any"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="EVN" type="EVN.CONTENT"/>

<!--
  SEGMENT EVN
-->
<!ENTITY % EVN.CONTENT "(EVN.1?,EVN.2,EVN.3?,EVN.4?,EVN.5*,EVN.6?,EVN.7?)">
<!ELEMENT EVN %EVN.CONTENT;>

```

## 2.4.2. Choice Groups of Segments

Another way of grouping segments is by a choice, i.e. here a decision has to be made which set of segments should be conveyed in a message: This is indicated by angle brackets: < and >. The different options for a choice are then separated by a vertical bar.

Please note, that this vertical bar is independent from the vertical bar in the conventional encoding reflecting the standard field delimiter.

### CCI^I22^CCI I22: Collaborative Care Information

<u>Segments</u>	<u>Description</u>	<u>Status</u>	<u>Chapter</u>
MSH	Message Header		2
..	..		
[{	--- CLINICAL_HISTORY_DETAIL begin		
<	--- CLINICAL_HISTORY_OBJECT begin		
OBR	Observation		4
ODS	Dietary Order, Suppl., Prefer.		4
PR1	Procedure		6
RF1	Referral Information		11

<u>Segments</u>	<u>Description</u>	<u>Status</u>	<u>Chapter</u>
AL1	Allergy Information		3
IAM	Patient adverse reaction information		3
ACC	Accident Information		6
PDA	Patient Death and Autopsy		3
>	--- CLINICAL_HISTORY_OBJECT end		
[{OBX}]	Observation/Result		7
}}	--- CLINICAL_HISTORY_DETAIL end		
..			

As an example, the corresponding schema definition fragment for the CLINICAL\_HISTORY\_OBJECT choice group is shown below.

```
<xsd:complexType name="CCI_I22.CLINICAL_HISTORY_OBJECT.CONTENT">
  <xsd:choice>
    <xsd:element ref="OBR" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="ODS" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="PR1" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="RF1" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="AL1" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="IAM" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="ACC" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="RMI" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="DB1" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="DGI" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="DRG" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="PDA" minOccurs="1" maxOccurs="1"/>
  </xsd:choice>
</xsd:complexType>
```

If more than a single segment is to be used, subgroups must be created.

## 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	C.LEN	DT	OPT	RP/#	TBL#	ITEM#	ELEMENT NAME
1				W			00099	Event Type Code
2			DTM	R			00100	Recorded Date/Time
3			DTM	O			00101	Date/Time Planned Event
4			CWE	O		0062	00102	Event Reason Code
5			XCN	O	Y	0188	00103	Operator ID
6			DTM	O			01278	Event Occurred
7			HD	O			01534	Event Facility

Figure 5: Segment attribute table of the EVN segment (fragment, drawn from chapter 3 in [rfHL7v27])

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 EVN encoding contains time stamp representation (TS) in EVN.2, EVN.3 and EVN.6 which are not primitive but



composite data types and which are expressed in a way described in detail in section "2.6.2. Composite Data Types".

EVN	A05	199901061000	199901101400	01	199901061000
-----	-----	--------------	--------------	----	--------------

```

...
<EVN>
  <EVN.1>A05</EVN.1>
  <EVN.2><TS.1>199901061000</TS.1></EVN.2>
  <EVN.3><TS.1>199901101400</TS.1></EVN.3>
  <EVN.4>01</EVN.4>
  <EVN.6><TS.1>199901061000</TS.1></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 explicit use of the "" is required to force a data delete action by the receiving application, see section "2.7.6. Delete Indicators, Empty Values"). 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> tag 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. In addition, the same information is provided as fixed attributes.

The example below shows the XML schema definition of the EVN.1 field element along with its annotations.

```

<!--
FIELD MSH.10
-->
<xsd:attributeGroup name="MSH.10.ATTRIBUTES">
  <xsd:attribute name="Item" type="xsd:string" fixed="10"/>
  <xsd:attribute name="Type" type="xsd:string" fixed="ST"/>
  <xsd:attribute name="LongName" type="xsd:string"
    fixed="Message Control ID"/>
  <xsd:attribute name="minLength" type="xsd:integer" fixed="1"/>
  <xsd:attribute name="maxLength" type="xsd:integer" fixed="199"/>
  <xsd:attribute name="truncation" type="xsd:string" fixed="="/>
</xsd:attributeGroup>
<xsd:complexType name="MSH.10.CONTENT">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">Message Control
ID</xsd:documentation>
    <xsd:documentation
xml:lang="de">Nachrichtenkontrollnummer</xsd:documentation>
  <xsd:appinfo>
    <hl7:Item>10</hl7:Item>
    <hl7:Type>ST</hl7:Type>
    <hl7:LongName>HL7Message Control ID</hl7:LongName>
  </xsd:appinfo>
</xsd:annotation>
  <xsd:simpleContent>
    <xsd:extension base="ST">
      <xsd:attributeGroup ref="MSH.10.ATTRIBUTES"/>
      <xsd:minLength value="1"/>
      <xsd:maxLength value="199"/>
    </xsd:extension>
  </xsd:simpleContent>

```

```
</xsd:complexType>
<xsd:element name="MSH.10" type="MSH.10.CONTENT"/>
```

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.

The constraints on minLength and maxLength allow for a schema-based validation. Nevertheless, providing this information as attributes in messages instances may be useful as well. Therefore, both options are valid. However, for backward compatibility reasons the new constraints should not be used with HL7 versions prior to 2.7, although this is a matter of negotiation between trading partners..

These rules also apply for data types.

## 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, date etc.

A field for which a primitive data type is defined simply contains the information without additional nesting or hierarchy. As an example the 4<sup>th</sup> 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).<sup>1</sup>

### 2.6.2. Composite Data Types

Complex data types are comprised of two or more components. As an example, consider the CNE data type (coded elements) which components are “identifier”, “text” and “name of coding 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<sup>2</sup>, but used for later reference in this specification).

Components: <Identifier (ST)> ^ <Text (ST)> ^ <Name of Coding System (ID)> ^ <Alternate Identifier (ST)> ^ <Alternate Text (ST)> ^ <Name of Alternate Coding System (ID)> ^ <Coding System Version ID (ST)> ^ <Alternate Coding System Version ID (ST)> ^ <Original Text (ST)> ^ <Second Alternate Identifier (ST)> ^ <Second Alternate Text (ST)> ^ <Name of Second Alternate Coding System (ID)> ^ <Second Alternate Coding System Version ID (ST)> ^ <Coding System OID (ST)> ^ <Value Set OID (ST)> ^ <Value Set Version ID (DTM)> ^ <Alternate Coding System OID (ST)> ^ <Alternate Value Set OID (ST)> ^ <Alternate Value Set Version ID (DTM)> ^ <Second Alternate Coding System OID (ST)> ^ <Second Alternate Value Set OID (ST)> ^ <Second Alternate Value Set Version ID (DTM)>

<sup>1</sup> This implies that the encoding of the v2.xml data types isn't as restrictive for as HL7 limits. As an example, for data types SI, NM, DT, DTM, more restrictive definitions for the schemas could be chosen. To keep schemas consistent, the restriction to data types in this sense isn't used in schema either.

<sup>2</sup> This table is newly introduced in version 2.5, chapter 2B.

HL7 Component Table - CNE - Coded No Exception Element

SEQ	LEN	C.LEN	DT	OPT	TBL#	COMPONENT NAME	COMMENTS	SEC.REF.
1		20=	ST	R		Identifier		<a href="#">2.A.75</a>
2		199#	ST	O		Text		<a href="#">2.A.75</a>
3	1..12		ID	O	0396	Name of Coding System		<a href="#">2.A.35</a>
4		20=	ST	O		Alternate Identifier		<a href="#">2.A.75</a>
5		199#	ST	O		Alternate Text		<a href="#">2.A.75</a>
6	1..12		ID	O	0396	Name of Alternate Coding System		<a href="#">2.A.35</a>
7		10=	ST	C		Coding System Version ID		<a href="#">2.A.75</a>
8		10=	ST	O		Alternate Coding System Version ID		<a href="#">2.A.75</a>
9		199#	ST	O		Original Text		<a href="#">2.A.75</a>
10		20=	ST	O		Second Alternate Identifier		<a href="#">2.A.75</a>
11		199#	ST	O		Second Alternate Text		<a href="#">2.A.75</a>
12	1..12		ID	O	0396	Name of Second Alternate Coding System		<a href="#">2.A.35</a>
13		10=	ST	C		Second Alternate Coding System Version ID		<a href="#">2.A.75</a>
14		199=	ST	C		Coding System OID		<a href="#">2.A.75</a>
15		199=	ST	O		Value Set OID		<a href="#">2.A.75</a>
16		8=	DTM	C		Value Set Version ID		<a href="#">2.A.22</a>
17		199=	ST	C		Alternate Coding System OID		<a href="#">2.A.75</a>
18		199=	ST	O		Alternate Value Set OID		<a href="#">2.A.75</a>
19		8=	DTM	C		Alternate Value Set Version ID		<a href="#">2.A.22</a>
20		199=	ST	C		Second Alternate Coding System OID		<a href="#">2.A.75</a>
21		199=	ST	O		Second Alternate Value Set OID		<a href="#">2.A.75</a>
22		8=	DTM	C		Second Alternate Value Set Version ID		<a href="#">2.A.22</a>

Figure 6: CNE data type components drawn from [rHL7v27] (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 CNE is defined as CNE.1, the second as CNE.2 and so on. This allows individual access to any of the components of a composite data type. The following example shows a CNE data type encoded traditionally (“vertical bar”) and as v2.xml fragment.

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

```

<CNE.1>F-11380</CNE.1>
<CNE.2>CREATININE</CNE.2>
<CNE.3>I9</CNE.3>
<CNE.4>2148-5</CNE.4>
<CNE.5>CREATININE</CNE.5>
<CNE.6>LN</CNE.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 CWE carries information in the first component only (i. e. the identifier of a coded element), the correct v2.xml encoding is shown as in the following example with an OBX.3 field:

Incorrect:

```

<OBX.3>
  F-11380
</OBX.3>

```

Correct:

```

<OBX.3>
  <CWE.1>F-11380</CWE.1>
</OBX.3>

```

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 CNE.1. In addition, the same information is provided as fixed attributes.

```

<!--
  COMPONENT CNE.3
-->
<xsd:element name="CNE.3" type="CNE.3.CONTENT"/>
<xsd:complexType name="CNE.3.CONTENT">
  <xsd:annotation>
    <xsd:documentation
      xml:lang="en">Name of Coding System</xsd:documentation>
    <xsd:appinfo>
      <hl7:type>ID</hl7:type>
      <hl7:LongName>Name of Coding System</hl7:LongName>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:simpleContent>
    <xsd:extension base="ID">
      <xsd:attributeGroup ref="CNE.3.ATTRIBUTES"/>
      <xsd:minLength value="1"/>
      <xsd:maxLength value="12"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:attributeGroup name="CNE.3.ATTRIBUTES">
  <xsd:attribute name="Type" type="xsd:string" fixed="ID"/>
  <xsd:attribute name="LongName" type="xsd:string"
    fixed="Name of Coding System"/>
  <xsd:attribute name="minLength" type="xsd:integer" fixed="1"/>
  <xsd:attribute name="maxLength" type="xsd:integer" fixed="12"/>
</xsd:attributeGroup>

```

### 2.6.3. Wildcard

In the HL7 Standard, a 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 to be any *HL7 data type*.

### 2.6.4. CM Data Types

A special data type, the CM data type, was used up to and including HL7 v2.3.1 to express that the explicit data type of the content is undefined (i. e. a type=“anyType” in XML Schema definitions). Use of this data type was deprecated beginning with v2.4 in order to allow more restricted conformance testing (no new fields would use the CM data type).

In v2.5, new data types were created for (and applied to) all existing fields/components using the CM data type. An addendum, for XML encoding, was 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 messages between sender and receiver, additional assumptions are made in terms of “well-formed” and “valid” XML instance documents.

The **sender** of a v2.xml XML message is required to create *both well-formed and valid* message instances. The instances created should be valid against the corresponding XML schema definitions (see section “3.1.2. List of Schemas”). However, this does not necessarily imply validation of the transaction at run time. The decision to do so and incur associated overhead should be made on a site-by-site basis or on interface development status.

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. For example, there is no difficulty changing data types to new data types that are backward compatible (as an example, IS to CE data type), or converting a repeating/optional segment into a repeating/optional group of segments. This helps ensure inter-version compatibility. Because the XML encoding makes explicit use of constructs touched by the changes mentioned above, inter-version backward compatibility is not a 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 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, a

backload of persons, admissions (ADT message batch for initial patient backload), employees, and master files.

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 can be found in the messages schema (*batch.xsd*). For queries there is the need to define a QPD segment differently for one query to a different query. The only way to support batches of queries (e. g. for non time critical processing) or responses is to wrap the contents of the batch tags as CDATA. This approach has been used for the general definitions of batch message “payload”, regardless of containing query segments or not.

For further information on batch messages refer to Chapter 2 and 5 of the standard.

## 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, escape, and truncation (as of v2.7) 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 4<sup>th</sup> 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. Although the message delimiter characters are meaningless in the v2.xml encoding, they are represented as shown in the example fragment of the MSH segment. However, they can be useful when translating from vertical bar to XML representation and vice versa. 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 message instance (see also section “2.7.10. Special Characters in Schemas”).

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

## 2.7.6. Delete Indicators, Empty Values

Where a sending system can ascertain that a data field has been deleted, then the two double quote marks ("" ) will be used to define the state of that data field. An encoded field with a value of two double quote marks ("" ) would instruct the receiving system to delete the contents in the database field.

If the state of a blank or null data field cannot be determined, the sending system will send the empty value or omit the element at all. An encoded field with an empty value or a missing element would instruct the receiving system to bypass processing and does not affect an already existing value in the corresponding receiving database.

The occurrence of an empty element is treated as not existing to keep backward compatibility with ER7.

The following example carries a delete indication in the <XAD.6> data type component. Explicit empty (missing) values are expressed by empty (missing) element content. In the example, <XAD.2> is omitted (empty).

```
<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>

```

### 2.7.7. Repetition of Segment Groups, Segments and Fields

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

#### example 1

```

<DG1>...</DG1>
<DG1>...</DG1>

```

#### example 2

```

<ECR.3>PARAM1</ECR.3>
<ECR.3>PARAM2</ECR.3>

```

#### example 3

```

<NK1.6>
  <XTN.1>(900)545-1234</XTN.1>
</NK1.6>
<NK1.6>
  <XTN.1>(900)545-1200</XTN.1>
</NK1.6>

```

### 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.

#### 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 V="H" />special<escape V="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



<code>\.spn\</code>	<code>&lt;escape V="\.spn"/&gt;</code>	skip <i>n</i> vertical space
<code>\.in±n\</code>	<code>&lt;escape V="\.in±n"/&gt;</code>	indent <i>n</i> spaces
<code>\.ti±n\</code>	<code>&lt;escape V="\.ti±n"/&gt;</code>	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...
```

## 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.

## 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 for UTF-8 must be used. Even non-printable characters like form feed can be represented that way.

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

## 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. Each XML entity must have a single encoding. An XML document can be made up of several entities, which each may have different encodings, but switching character sets in a single entity is thus not supported.

## 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 the XML specification [rfXML]. For the receiving application, XML markup characters are normally handled by the XML parser.

### 2.7.9. Message Building Rules

The message building rules remain 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 or by the v2.x standard). If an element is marked as “required” in HL7, a value must be present. An empty tag is not enough.
- For groups of segments defined in the v2.xml specification, additional group elements are introduced. In the standard encoding (ER7), 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, the receiver



- may ignore segments, fields, components, subcomponents, and extra repetitions of a field that are present but were not expected,
- will treat segments that were expected but are not present as consisting entirely of fields that are not present,
- will 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 determined by the v2.xml schema.

Please note, that, in correspondence of what the processing rules for v2 say for additional stuff after a segment, the schemas also allow any elements following after the end of a segment.

### 2.7.10. Special Characters in Schemas

Certain characters within the HL7 Database must be “escaped” before inclusion in a schema. The ampersand is a reserved XML meta character.

Where an ampersand occurs in the long name of a field, it is converted to an XML entity representation “&” An example is “Critical Range for Ordinal & Continuous Obs” that becomes “Critical Range for Ordinal & Continuous Obs”.

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

The same rules apply to XML message instances.

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. XML representation derivation from HL7 Database”). 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 are not straightforward although it is not hard. The issues described in section “2.7. Processing Rules for v2.xml Messages” need to be taken into account when performing the translations.

## 3. Appendix

### 3.1. Normative Appendix

#### 3.1.1. List of Messages With Equal Message Structures

As previously mentioned, the v2.xml schemas are based on the message structure ID – a concept introduced in version 2.3.1. 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
v2.5	2	Message structure table 0354
v2.5.1	2.17.3	Message structure table 0354
v2.6	2.16.3	Message structure table 0354
v2.7	2.C.2.175	Message structure table 0354
v2.7.1	????	Message structure table 0354
v2.8	????	Message structure table 0354

#### 3.1.2. List of Schemas

This specification provides the set of constraint definition files:

- XML Schema definitions (xsd)

as shown by the following table. There is a set for each HL7 version supported by the v2.xml specification. In addition, HTML files are provided, one for each message structure, containing a short description of the message and links to the corresponding schemas (in directory *xsd*).

Please note that the use of XML schemas is recommended by HL7 for all normative specifications. The use of XML schema ([rfXMLSchema], a W3C recommendation since May 2001) is recommended by HL7 for all normative specifications. The schemas 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.

It should be mentioned that DTDs can coexist in the same interface with schemas and not cause any issues. For example, the sending interface can implement XML messages using schemas and the receiving system using DTDs. However, schemas have a much greater expressiveness and should be preferred.

HTML descriptions (in subdirectory htm)	Description
<i>MessageStructureID</i> .html	A set of many files in HTML format containing a short description of the message and links to the corresponding schemas.
Schema (in subdirectory xsd)	Description
< <i>MessageStructureID</i> >.xsd	A set of many schemas each containing the schema definitions for a specific message structure specified by <i>MessageStructureID</i> , for example ADT_A01.xsd contains the definitions for the ADT A01 message structure, ADT_A02.xsd for ADT A02 and so forth. The schemas import the segments definitions.
Segments.xsd	schema for all segment definitions, imports fields definitions

fields.xsd	schema for all field definitions, imports data type definitions
datatypes.xsd	schema for all data type definitions for v2
batch.xsd	schema containing definition of batch messages (refer to section "2.7.4. Batch Messages")
Messages.xsd	schema containing all message definitions together

An XML instance of a specific message should refer to the corresponding schema. The following examples show 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>...
```

### 3.1.3. Localization of messages

The HL7 Standard describes the responsibilities for parties sending and receiving HL7 messages (see also section "2.7.1. XML Application Processing Rules" of this specification). These responsibilities enable exchange of messages that contain localizations (or local variations or Z-segments, Z-fields etc.). The v2.xml specification attempts to provide full support for local encodings (see section "Localization Issues"). 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.

#### Schema Localization

The schemas provided in this specification can be localized. This is done by redefinitions of the existing definitions.

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:

```
<!-- .. 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>
```

A redefinition containing the new localized content of the FOO content model can then be made on a copy of the schema definition.

```
<!-- .. message re-definition FOO (on a copy of the schema) -->
<xsd:element name="FOO">
```

```

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

```

Now, the FOO message is redefined by the local modification. The copy of the schema will be used instead of the original version. A new root element can be defined called FOO.LOCAL that serves as a localized version of the original FOO root element.

The corresponding local message instance would look like this:

```

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

```

Please note that it is good practice to intersperse local stuff by using a different namespace. It is therefore recommended to associate Z-stuff with another namespace.

## 3.2. Informative Appendix

### 3.2.1. 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.

#### 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.

#### 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. In some cases modeling components as XML attributes as opposed to elements could result 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 University Medical Center (DUMC) HL7 production environment suggests that on average, for standard 'vertical bar encoding', data characters comprise about 70% of overall message length. (Data from DUMC courtesy of Al Stone, and posted to the HL7 SGML/XML SIG

List Server 1998-01-15 and 1998-01-16) 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. Algorithms" 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.

Furthermore it should be mentioned that we've learned from early v2.xml implementers that performance could be gained (along with the use of less bandwidth) if large batch files are broken into many small batch files.

### 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. (actually, this is one of the most important "chewers" of development resources)
- 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.

### Localization Issues

The HL7 standard describes the responsibilities for parties sending and receiving HL7 messages (see also section "2.7.1. XML Application Processing Rules" of this specification). These responsibilities also enable exchange of messages that contain localizations (or referred to as 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 are crafted to fulfill these requirements. Please refer to section “3.1.3. Localization of messages” of the informative appendix for further information.

### 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 schema, such as the optionality and repetition of a field within a segment.

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 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 in terms of 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).

Some actions that are supported in vertical bar encoding, such as the forward-adoption of new data types cannot be handled by the XML encoding.

### 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 this 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.

For the second release, backward compatibility with the previous version of this specification should be guaranteed.

### 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.

### Messages and Their Segments

#### HL7 Database Tables Used

The first table is just introduced in order to obtain the correct identifier for a specific variation of a version because the database is capable of maintaining different versions in parallel:

<i>HL7Versions</i>			
Primary Key	Field Name	Data Type	Description
*	version_id	long integer	version number
	hl7_version	Text-8	HL7 version

	description	Text-80	Description
	date_release	Date	Release date of that version

The following HL7 Database table is 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.)

<b><i>HL7EventMessageTypes</i></b>			
<b>Primary Key</b>	<b>Field Name</b>	<b>Data Type</b>	<b>Description</b>
*	event_code	Text-3	Code of this event
*	version_id	long integer	version number
*	seq_no	Integer	consecutive increasing number used for 1:n relation
	message_typ_snd	Text-3	Standard Message Type (Sender)
	message_typ_return	Text-3	Standard Message Type (Recipient)
	message_structure_snd	Text-7	Message Structure (Sender)
	message_structure_return	Text-7	Message Structure (Recipient)
	Chapter	Text-10	Chapter in which this message is described

### SQL Query

The following two queries are used to gather together message structures from table *HL7EventMessageTypes*:

```
SELECT HL7EventMessageTypes.message_typ_snd AS messageType,
       HL7EventMessageTypes.event_code AS eventCode,
       HL7EventMessageTypes.message_structure_snd AS messageStructure,
       HL7EventMessageTypes.hl7_version AS HL7version,
       HL7EventMessageTypes.chapter AS chapter
       INTO MsgStructv2xml
FROM HL7EventMessageTypes
WHERE HL7EventMessageTypes.message_structure_snd <> "NUL" ;
```

### Select Message Structures

From database table *HL7MsgStructIDSegments* (see below) those message structures were extracted from *HL7MsgStructv2xml* using the SQL query shown below.

<b><i>HL7MsgStructIDSegments</i></b>			
<b>Primary Key</b>	<b>Field Name</b>	<b>Data Type</b>	<b>Description</b>
*	message_structure	Text-7	Message Structure ID
*	version_id	long integer	version number
*	seq_no	Integer	consecutive increasing number used for each field within the segment
	seg_code	Text-3	Segment-Code
	groupname	Text-10	String identifying the repetition of subsequent segments (logical embracement)

	repetitional	Yes/No	Repetitional
	Optional	Yes/No	Optional

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

```
SELECT HL7MsgStructIDSegments.message_structure,
       HL7MsgStructIDSegments.seq_no,
       HL7MsgStructIDSegments.seq_code,
       HL7MsgStructIDSegments.repetitional AS Ausdr1,
       HL7MsgStructIDSegments.optional,
       HL7MsgStructIDSegments.hl7_version
FROM   HL7MsgStructIDSegments
WHERE  (HL7MsgStructIDSegments.hl7_version=" version ")
ORDER BY HL7MsgStructIDSegments.message_structure,
         HL7MsgStructIDSegments.seq_no;
```

This resulting table is exported to `messages.txt`. This file serves as input for the transformation algorithms (see also section "3.2.3. Options").

## Segments and Fields

### 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.) They are used to generate *Segments.xsd*.

<i>HL7Segments</i>			
Primary Key	Field Name	Data Type	Description
*	seg_code	Text-3	Code for the Segment
*	version_id	long integer	version number
	Description	Text-50	The name of the segment
	Interpretation	Text-50	The German interpretation of the name
	Visible	Yes/no	Is this a segment being visible
	...		

<i>HL7SegmentDataElements</i>			
Primary Key	Field Name	Data Type	Description
*	seg_code	Text-3	Code of the Segment
*	version_id	long integer	version number
*	seq_no	Integer	Position within the segment
	data_item	Long Integer	Data Element ID
	req_opt	Text-5	required/optional/backward compatibility
	repetitional	Text-1	Repetitional
	Repetitions	long integer	Number of repetitions



## SQL Query

The first query selects all relevant segments:

```
SELECT HL7Segments.seg_code
FROM HL7Segments
WHERE HL7Segments.version_id = " version " AND visible = true
ORDER BY HL7Segments.seg_code;
```

The following SQL query extracts data from tables *HL7SegmentDataElements* and *HL7DataElements*:

```
SELECT HL7SegmentDataElements.seg_code, HL7SegmentDataElements.lfd_nr,
      HL7DataElements.data_item, HL7DataElements.description,
      HL7DataElements.data_structure, HL7SegmentDataElements.req_opt,
      HL7SegmentDataElements.repetitional, HL7DataElements.table_id,
      HL7DataElements.hl7_version
FROM HL7DataElements
      INNER JOIN HL7SegmentDataElements ON HL7DataElements.data_item =
      HL7SegmentDataElements.data_item
WHERE ((HL7DataElements.hl7_version=" version ") AND
      ((HL7SegmentDataElements.hl7_version)=[HL7DataElements].[hl7_version]) AND
      (HL7SegmentDataElements.req_opt <> 'W') AND
      (HL7DataElements.data_structure <> '-'))
ORDER BY HL7SegmentDataElements.seg_code, HL7SegmentDataElements.seq_no;
```

## Data Elements

### HL7 Database Tables Used

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

<i>HL7DataElements</i>			
Primary Key	Field Name	Data Type	Description
*	data_item	Long Integer	ID of the Data Element
*	version_id	Long Integer	Version number
	description	Text-78	Field description according to the standard documentation
	data_structure	Text-20	Name of the Data Structure
	Min_length	Long Integer	minimum length (new)
	Max_length	Long Integer	maximum length (new) - contains also the length of previous version
	Conf_length	Text-10	conformance length
	table_id	Long Integer	ID assigned table

## SQL Query

The following SQL query extracts data from tables *HL7SegmentDataElements* and *HL7DataElements*: to create *Fields.xsd*:

```
SELECT HL7SegmentDataElements.seg_code, HL7SegmentDataElements.lfd_nr,
      HL7DataElements.data_item, HL7DataElements.description,
      HL7DataElements.data_structure, HL7SegmentDataElements.req_opt,
```

```

        HL7SegmentDataElements.repetitional, HL7DataElements.table_id,
        HL7DataElements.hl7_version
FROM HL7DataElements
      INNER JOIN HL7SegmentDataElements ON HL7DataElements.data_item =
        HL7SegmentDataElements.data_item
WHERE ((HL7DataElements.hl7_version=" version ") AND
       ((HL7SegmentDataElements.hl7_version)=[HL7DataElements].[hl7_version]) AND
       (HL7SegmentDataElements.req_opt <> 'W') AND
       (HL7DataElements.data_structure <> '-'))
ORDER BY HL7SegmentDataElements.seq_code, HL7SegmentDataElements.seq_no;

```

## Data Types and Their Data Type Components

### 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.)

<i>HL7DataStructures</i>			
Primary Key	Field Name	Data Type	Description
*	data_structure	Text-20	logical data type
*	version_id	long integer	version number
	Description	Text-80	Description

<i>HL7DataStructureComponents</i>			
Primary Key	Field Name	Data Type	Description
*	data_structure	Text	logical data type
*	version_id	long integer	version number
*	seq_no	long integer	consecutive increasing number used for 1:n relation
	comp_nr	long integer	identifying number of the component
	table_id	long integer	Number of assigned table if different from component (overwrites table number of component)

<i>HL7Components</i>			
Primary Key	Field Name	Data Type	Description
*	comp_nr	Long Integer	Component Number (ID)
*	version_id	long integer	version number
	description	Text-50	Description
	table_id	Long Integer	reference to an assigned Table
	data_type_code	Text-3	Data type

### SQL Query

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

```
SELECT HL7DataStructures.data_structure, HL7DataStructureComponents.seq_no,
       HL7DataStructures.description, HL7DataStructureComponents.table_id,
       HL7Components.description, HL7Components.table_id,
       HL7Components.data_type_code, DataStructures.hl7_version
FROM HL7DataStructures LEFT JOIN (HL7DataStructureComponents LEFT JOIN
       HL7Components ON (HL7DataStructureComponents.version_id =
       HL7Components.version_id) AND (HL7DataStructureComponents.comp_nr =
       HL7Components.comp_nr)) ON (HL7DataStructures.version_id =
       HL7DataStructureComponents.version_id) AND
       (HL7DataStructures.data_structure =
       HL7DataStructureComponents.data_structure)
WHERE (HL7DataStructures.version_id=" version ")
ORDER BY HL7DataStructures.data_structure, HL7DataStructureComponents.seq_no;
```

### 3.2.3. Options

The database contains a German interpretation of the different elements as well. They can be added to the schemas in form of further annotations if needed:

```
<xsd:annotation>
  <xsd:documentation xml:lang="en">
    Principal Language Of Message</xsd:documentation>
  <xsd:documentation xml:lang="de">
    Sprache der Nachricht </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>
```

### 3.2.4. 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.

A VB program generates the XML schema definitions and additional HTML files for further information. The structure of the generated schemas follows from the design considerations described above. The algorithms instantiated in this VB program are not described in detail here and are not part of this specification, but will be available with the HL7 database and on the HL7 website.

### 3.2.5. Examples

#### Schema 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:attributeGroup name="CE.1.ATTRIBUTES">
    <xsd:attribute name="Type" type="xsd:string" fixed="ST"/>
    <xsd:attribute name="LongName" type="xsd:string"
      fixed="identifier (ST)"/>
  </xsd:attributeGroup>
  <xsd:complexType name="CE.1.CONTENT">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
        identifier (ST)</xsd:documentation>
      <xsd:appinfo>
        <hl7:Type>ST</hl7:Type>
        <hl7:LongName>identifier (ST)</hl7:LongName>
      </xsd:appinfo>
    </xsd:annotation>

    <xsd:simpleContent>
      <xsd:extension base="ST">
        <xsd:attributeGroup ref="CE.1.ATTRIBUTES"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:element name="CE.1" type="CE.1.CONTENT"/>

```

```

<!-- Field definitions -->

```

```

...
<!--
  FIELD MSH.19
-->
  <xsd:attributeGroup name="MSH.19.ATTRIBUTES">
    <xsd:attribute name="Item" type="xsd:string" fixed="693"/>
    <xsd:attribute name="Type" type="xsd:string" fixed="CE"/>
    <xsd:attribute name="LongName" type="xsd:string"
      fixed="Principal Language Of Message"/>
  </xsd:attributeGroup>
  <xsd:complexType name="MSH.19.CONTENT">
    <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:complexContent>
    <xsd:extension base="CE">
        <xsd:attributeGroup ref="MSH.19.ATTRIBUTES"/>
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
<xsd:element name="MSH.19" type="MSH.19.CONTENT"/>

```

```

<!-- Segment definitions -->

```

```

...
<!--
    SEGMENT FAC
-->
<xsd:complexType name="FAC.CONTENT">
    <xsd:sequence>
        <xsd:element ref="FAC.1" minOccurs="1" maxOccurs="1" />
        <xsd:element ref="FAC.2" minOccurs="0" maxOccurs="1" />
        <xsd:element ref="FAC.3" minOccurs="1" maxOccurs="unbounded" />
        <xsd:element ref="FAC.4" minOccurs="1" maxOccurs="1" />
        <xsd:element ref="FAC.5" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="FAC.6" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="FAC.7" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="FAC.8" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="FAC.9" minOccurs="1" maxOccurs="unbounded" />
        <xsd:element ref="FAC.10" minOccurs="0" maxOccurs="1" />
        <xsd:element ref="FAC.11" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="FAC.12" minOccurs="0" maxOccurs="1" />
        <xsd:any processContents="lax" namespace="##any"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="FAC" type="FAC.CONTENT"/>

```

```

<!-- Message definitions -->

```

```

...
<?xml version = "1.0" ?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="urn:hl7-org:v2xml"
    targetNamespace="urn:hl7-org:v2xml">

    <!-- include segment definitions for version v24 -->
    <xsd:include schemaLocation="segments.xsd"/>

<!--
    MESSAGE ACK
-->

    <!-- .. message definition 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 name="ACK" type="ACK.CONTENT" />

</xsd:schema>

```

## V2 and v2.xml Example Messages

### 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.^STATEVILLE^OH^35292|||(206)3345232
|(206)752-121|||AC555444444|67-A4335^OH^20030520
OBR|1|845439^GHH OE|1045813^GHH LAB|1554-5^GLUCOSE^LN|||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^LN||^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>^~\&amp;</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>
      <MSG.1>ORU</MSG.1>
      <MSG.2>R01</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>AC555444444</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>
            <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>
            <CE.3>LN</CE.3>
        </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>
        </OBR.16>
    </OBR>

```

```

        </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>
        <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>
        </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 POST 12H CFST</CE.2>
            <CE.3>LN</CE.3>
        </OBX.3>
        <OBX.5>
            <SN.2>182</SN.2>
        </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>

```

### Long Example

```

MSH|^~\&|REGADT|MCM|IFENG||199112311501||ADT^A04^ADT_A01|000001|P|2.4|||
EVN|A04|199901101500|199901101400|01||199901101410
PID|||191919^^GENHOS^MR~371-66-9256^^^USSA^SS
    |253763|MASSIE^JAMES^A||19560129|M||171 ZOBERLEIN^^ISHPEMING^MI^49849^""^|
    |(900)485-5344|(900)485-5344||S^^HL70002|C^^HL70006|10199925^^^GENHOS^AN
    |371-66-9256||
NK1|1|MASSIE^ELLEN|SPOUSE^^HL70063|171 ZOBERLEIN^^ISHPEMING^MI^49849^""^
    |(900)485-5344|(900)545-1234~(900)545-1200|EC1^FIRST EMERGENCY CONTACT^HL70131
NK1|2|MASSIE^MARYLOU|MOTHER^^HL70063|300 ZOBERLEIN^^ISHPEMING^MI^49849^""^
    |(900)485-5344|(900)545-1234~(900)545-1200|EC2^SECOND EMERGENCY CONTACT^HL70131
NK1|3
NK1|4||123 INDUSTRY WAY^^ISHPEMING^MI^49849^""^|(900)545-1200
    |EM^EMPLOYER^HL70131|19940605||PROGRAMMER||ACME SOFTWARE COMPANY
PV1||O|R|||0148^ADDISON,JAMES|0148^ADDISON,JAMES||AMB|||
    |0148^ADDISON,JAMES|S|1400|A|||GENHOS|||199501101410|
PV2|||199901101400|||199901101400
ROL|AD|CP^^HL70443|0148^ADDISON,JAMES

```



```

OBX|NM|3141-9^BODY WEIGHT^LN||62|kg|||F
OBX|NM|3137-7^HEIGHT^LN||190|cm|||F
DG1|1|19||R63.4^LOSS OF WEIGHT^I10||00|
GT1|1|MASSIE^JAMES^"^^"^^"^^"^^"^^|171 ZOBERLEIN^^ISHPEMING^MI^49849^"^^
|(900)485-5344|(900)485-5344|||SE^SELF^HL70063|371-66-925|||MOOSSES AUTO CLINIC
|171 ZOBERLEIN^^ISHPEMING^MI^49849^"^^|(900)485-5344|
IN1|0|0^HL70072|BC1|BLUE CROSS|171 ZOBERLEIN^^ISHPEMING^M149849^"^^|
|(900)485-5344|90|||50 OK|

```

```

<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>
    <MSH.1>|</MSH.1>
    <MSH.2>^~\&amp;</MSH.2>
    <MSH.3>
      <HD.1>REGADT</HD.1>
    </MSH.3>

    <MSH.4>
      <HD.1>MCM</HD.1>
    </MSH.4>
    <MSH.5>
      <HD.1>IFENG</HD.1>
    </MSH.5>
    <MSH.7>
      <TS.1>199112311501</TS.1>
    </MSH.7>
    <MSH.9>
      <MSG.1>ADT</MSG.1>
      <MSG.2>A04</MSG.2>
      <MSG.3>ADT_A01</MSG.3>
    </MSH.9>
    <MSH.10>000001</MSH.10>
    <MSH.11>
      <PT.1>P</PT.1>
    </MSH.11>
    <MSH.12>
      <VID.1>2.4</VID.1>
    </MSH.12>
  </MSH>

  <EVN>
    <EVN.1>A04</EVN.1>
    <EVN.2>
      <TS.1>199901101500</TS.1>
    </EVN.2>
    <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.1>GENHOS</HD.1>
      </CX.4>
    </PID.3>
  </PID>

```

```

        </CX.4>
        <CX.5>MR</CX.5>
    </PID.3>
    <PID.3>
        <CX.1>371-66-9256</CX.1>
        <CX.4>
            <HD.1>USSSA</HD.1>
        </CX.4>
        <CX.5>SS</CX.5>
    </PID.3>
    <PID.4>
        <CX.1>253763</CX.1>
    </PID.4>
    <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.1>GENHOS</HD.1>
        </CX.4>
        <CX.5>AN</CX.5>
    </PID.18>
    <PID.19>371-66-9256</PID.19>
</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>19</DG1.2>
  <DG1.3>
```

```

        <CE.1>R63.4</CE.1>
        <CE.2>LOSS OF WEIGHT</CE.2>
        <CE.3>I10</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>
</IN1.4>
<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>
  <AUI.1>50 OK</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>
- [rfHL7v25] HL7 Version 2.5, HL7 V2.5-2003, Final Standard, August, 2003  
<http://www.hl7.org>
- [rfHL7v251] HL7 Version 2.5.1, HL7 V2.5.1-2007, Final Standard, May, 2007  
<http://www.hl7.org>
- [rfHL7v26] HL7 Version 2.6, HL7 V2.6-2007, Final Standard, October, 2007  
<http://www.hl7.org>
- [rfHL7v27] HL7 Version 2.7, HL7 V2.7-2011, Final Standard, April, 2011  
<http://www.hl7.org>
- [rfHL7v2.xml] HL7 v2.xml, Rel.1, Final Standard, June, 2003  
<http://www.hl7.org>
- [rfXML] Extensible Markup Language (XML) 1.0. 2<sup>nd</sup> 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/>
- [rfXMLSchema1.1p1] W3C XML Schema Definition Language (XSD) 1.1, Part 1: Structures Working Draft 3, Dec. 2009  
<http://www.w3c.org/TR/2009/WD-xmlschema11-2-20091203>
- [rfXMLSchema1.1p2] W3C XML Schema Definition Language (XSD) 1.1, Part 2: Datatypes Working Draft 3, Dec. 2009  
<http://www.w3c.org/TR/2009/WD-xmlschema11-2-20091203>
- [rfXMLnamespace] XML Namespace, W3C Recommendation: Namespaces in XML.  
<http://www.w3.org/TR/REC-xml-names/>
- [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 <a href="http://www.oemig.de/HL7/hl7amia96.htm">http://www.oemig.de/HL7/hl7amia96.htm</a>
[rfOemig]	Frank Oemig's Home Page <a href="http://www.oemig.de/HL7/">http://www.oemig.de/HL7/</a>
[Krueger]	Krueger G. A structured approach to HL7 application development. 1996. <a href="http://www.gkrueger.com/other/hl7/">http://www.gkrueger.com/other/hl7/</a>
[rfHL7v3ITS]	HL7 Version 3 – XML ITS, see HL7 Version 3 (Draft) <a href="http://www.hl7.org">http://www.hl7.org</a>
[rfXPATh]	XML Path Language (XPath) Version 1.0, W3C Recommendation <a href="http://www.w3.org/TR/xpath">http://www.w3.org/TR/xpath</a>

(intentionally left blank)