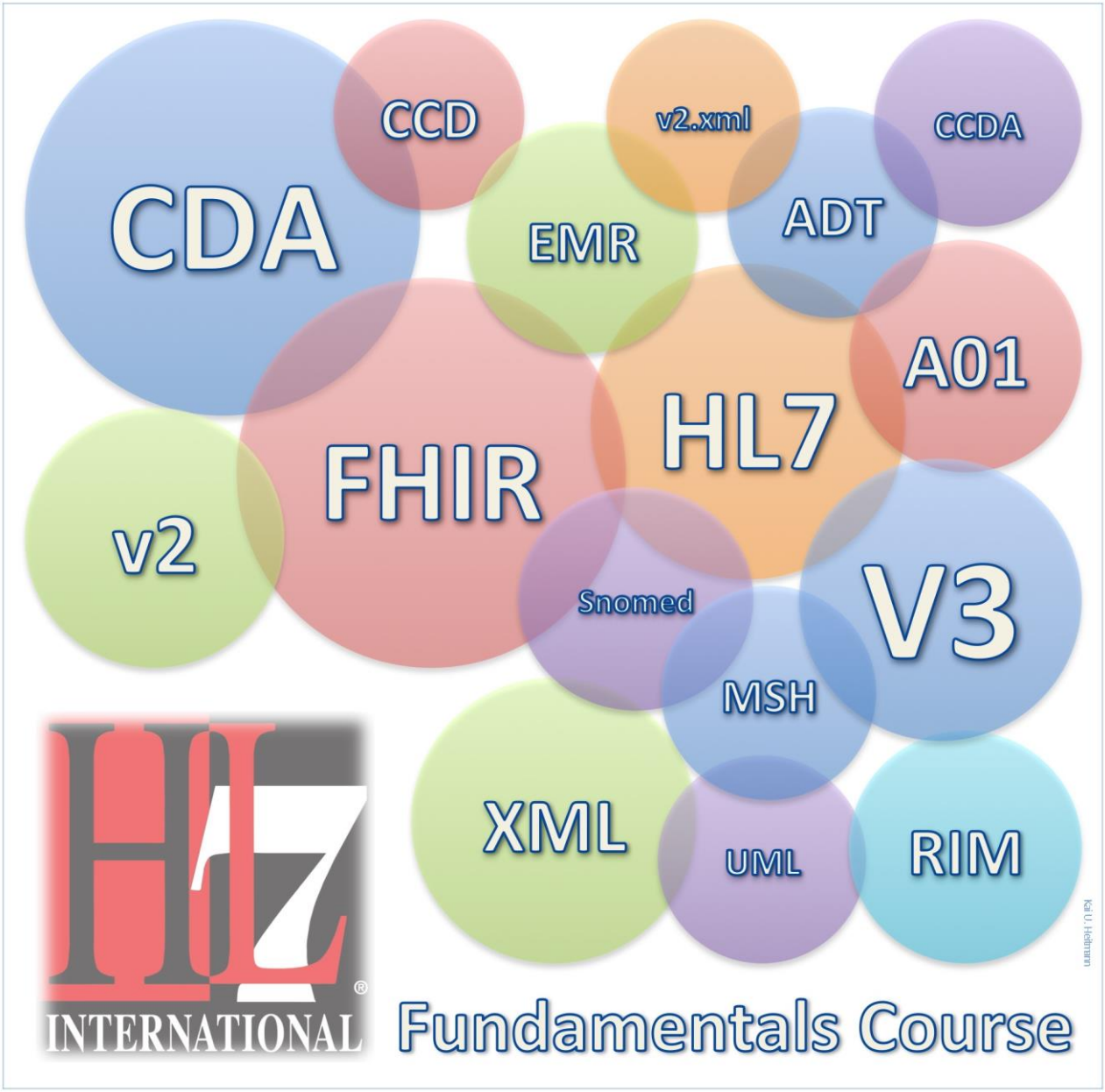

V2.x Module

XML Encoding V2.x Messages

Reading Material

Table of Contents

Unit Content and Learning Objectives

In this Unit you will learn how to format ("encode") HL7 V2.x in XML, rather than the traditional "Stick and Caret" or "Pipe and Hat" format.

1.Introduction

In the "Introduction to XML" Unit, you leant the advantages XML offers for systems integration and interoperability as well as why standards development organizations like HL7 International use those advantages.

Therefore a specification of the HL7 V2.x standard called v2.xml was developed. In this Unit, we will learn its characteristics, purpose and implementation.

Note: Throughout this Unit, we will refer to the traditional HL7 notation or encoding as "pipes and hats", "sticks and carets" or "vertical bar" as opposed to XML encoding for HL7 V2 messages.

2.XML Encoding

HL7 v2.x defines the syntax and semantics of messages, specifying the interchange format and the processing rules.

The standardization and popularity of XML led to the development of the **HL7 v2.xml Encoding Rules** as an alternative for systems capable of understanding and processing XML documents.

This specification, known as the **HL7 v2.xml Specification**, does not alter the Version 2.x message definitions. It does not attempt to replace the ubiquitous bars encoding syntax, but offers an alternative to it.

Given its flexibility, several XML encodings could exist for HL7 messages. In June 2003, HL7 International standardized on one specification that consists of a translation of the existing message structures to XML.

The DTD and Schemas corresponding to each type of message are provided with this specification. We are going to focus on the Schemas, given their greater expressive power.

There are many benefits of using XML encoding for the messages:

- It is not necessary for each implementation to develop its own ad-hoc parser as was done for bar encoding, since XML parsers are standard and available for most platforms.
- Therefore, the cost of development and maintenance of parsers and message generators is reduced.
- It is nearer to the generic information interchange mechanisms, which many systems already support.
- It facilitates interchange with other systems that are not strictly in the healthcare domain but that have a very strong relation with it, like the payer community.
- It facilitates the transition to HL7 version 3.

In support of the standard, the Schemas and DTDs were generated in an algorithmic way from a database representation of the message definitions developed by HL7 Germany. This allows straightforward maintenance of the Schemas and DTDs for the different releases of HL7 v2.x. These Schemas and DTDs are available from HL7 along with the XML encoding specification.

Naturally, the Schemas and DTDs can be extended to support localization such as Z segments for a given implementation.

V2.3.1 messages and above are supported by V2.xml encoding.

An HL7 V2.x message is composed of segments terminated by a carriage return <CR>. Each segment is in turn composed of fields delimited by the field separator (usually the vertical bar "|"). Each field is made up of one or more components delimited by the component separator (usually the caret "^") and corresponds to a specific data type. A component may have one or more subcomponents delimited by the subcomponents separator (usually the ampersand "&").

The structure of an HL7 v2.x message is hierarchical and has well defined semantics.

We are going to show you how an instance of a message is encoded and how the XML Schema is defined as well.

A valid XML document can be generated using a Schema with an application and perhaps some XSL. If we have a valid XML document, it can beparsed and its information processed in a consistent manner.

Each HL7 v2.x message can now be an XML document.

Here is a summary of how each part of the message is represented in XML:

Part of the Message	XML
Message structure	Root element
Segment	Element
Field	Element
Component	Element
Field data type	Fixed attribute of the element corresponding to the field and the **<appinfo>** element of the Schema
Field name and abbreviation	Fixed attribute of the element corresponding to the field and the **<documentation>** element of the Schema
Reference tables	Fixed attribute of the element corresponding to the field

3. The Message

The "Admit/Visit Notification" (A01) message will be used to demonstrate the XML encoding.

The following table shows a V2.8 abstract message format of the "Admit/Visit Notification" ADT message (ADT^A01^ADT_A01):

Segments	Description	Chapter
MSH	Message Header	2
[{ SFT }]	Software Segment	2
[UAC]	User Authentication Credential	2
EVN	Event Type	3
PID	Patient Identification	3
[PD1]	Additional Demographics	3
[{ ARV }]	Access Restrictions	3
[{ ROL }]	Role	15
[{ NK1 }]	Next of Kin / Associated Parties	3
PV1	Patient Visit	3
[PV2]	Patient Visit - Additional Info.	3
[{ ARV }]	Access Restrictions	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	15
}]	--- PROCEDURE end	
[{ GT1 }]	Guarantor	6
[{	--- INSURANCE begin	
IN1	Insurance	6
[IN2]	Insurance Additional Info.	6
[{ IN3 }]	Insurance Additional Info - Cert.	6
[{ ROL }]	Role	15
[{ AUT }]	Authorization Record	11
[{ RF1 }]	Referral Information	11
}]	--- INSURANCE end	
[ACC]	Accident Information	6
[UB1]	Universal Bill Information	6
[UB2]	Universal Bill 92 Information	6
[PDA]	Patient Death and Autopsy	3

This is an example of an A01 message:

```
MSH|^~\&|NSI||LAB||20140327120759||ADT^A01^ADT_A01|NSI1|P|2.8||||AL<cr>
EVN||20011001000000<cr>
PID|1||55555^^^^HI|5555555^^^^DNI|TEST^AIDA||19780113000000|F|||
POTOSI 4032 108^^BUENOS AIRES^^1899<cr>
NK1|1|CAMUS^ALBERTO|PAD|RIVADAVIA 253|42539686<cr>
PV1|1|I|12^301^1211|R|||1436^PEREZ^JORGE^ALBERTO|1026^LOPEZ^NORBERTO|
998^GARCIA^ALEJANDRO|M|||A|4|A0|N|1026^LOPEZ^NORBERTO|OB|H0100240
```

```
||||||||||||||||ALV|||||||20140323095130|20140323102455<cr>
PV2||||SUB INTESTINAL|||||||||||||||||||^^223|||||||||||||N<cr>
IN1|1|INT^^HI|2^^^^HI~347^^^^NSI|PLAN DE SALUD<cr>
```

The Schemas and the structure of the messages are defined by the third component of the MSH-9 field, the **Message Structure ID**. Messages corresponding to different events may share a structure and will also share a Schema. Thus, an XML document of an ADT_A04 message will validate with the Schema of the ADT_A01 given their shared structure.

The Message Structure ID represents the root element of the XML document. The segments are its child elements with the three character segment identifiers being the name of those elements:

```
<ADT_A01>
  <MSH>
    ...
  </MSH>
  <EVN>
    ...
  </EVN>
  <PID>
    ...
  </PID>
  ...
</ADT_A01>
```

The message abstract definition identifies optional **[...]** and repetitive **{ ... }** segments. There is a parallel between this definition and the Schema:

Abstract HL7 syntax	Quantity in Schema	
	minOccurs	maxOccurs
[…]	0	1
{ … }	1	unbounded
{[…]} or [{ … }]	0	unbounded
Neither curly nor square brackets	1	1

The message ADT_A01 Schema is defined as follows:

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

  <xsd:element name="ADT_A01" type="ADT_A01.CONTENT"/>
  <xsd:complexType name="ADT_A01.CONTENT">
    <xsd:sequence>
      <xsd:element ref="MSH"/>
      <xsd:element ref="SFT" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="EVN"/>
      <xsd:element ref="PID"/>
      <xsd:element ref="PD1" minOccurs="0"/>
      <xsd:element ref="ROL" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="NK1" minOccurs="0" maxOccurs="unbounded"/>
```

```
        <xsd:element ref="PV1"/>
        <xsd:element ref="PV2" minOccurs="0"/>
        <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"/>
        <xsd:element ref="ADT_A01.PROCEDURE" minOccurs="0" max-
Occurs="unbounded"/>
        <xsd:element ref="GT1" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="ADT_A01.INSURANCE" minOccurs="0" max-
Occurs="unbounded"/>
        <xsd:element ref="ACC" minOccurs="0"/>
        <xsd:element ref="UB1" minOccurs="0"/>
        <xsd:element ref="UB2" minOccurs="0"/>
        <xsd:element ref="PDA" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="ADT_A01.PROCEDURE" type="ADT_A01.PROCEDURE.CONTENT"/>

    <xsd:complexType name="ADT_A01.PROCEDURE.CONTENT">
      <xsd:sequence>
        <xsd:element ref="PR1"/>
        <xsd:element ref="ROL" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="ADT_A01.INSURANCE" type="ADT_A01.INSURANCE.CONTENT"/>

    <xsd:complexType name="ADT_A01.INSURANCE.CONTENT">
      <xsd:sequence>
        <xsd:element ref="IN1"/>
        <xsd:element ref="IN2" minOccurs="0"/>
        <xsd:element ref="IN3" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="ROL" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>

  </xsd:schema>
```

We can see that the root element is indeed the one corresponding to the Message Structure ID:

```
  <xsd:element name="ADT_A01" type="ADT_A01.CONTENT"/>
```

Linking the Schema to the XML document, the root element will be:

```
<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">
```

4.Segment Groups

For segment groups, optional or repetitive, an element with the name of the group is used. This element includes the elements of the corresponding segments. This expresses a semantic impossible to represent in "hats and pipes" (^|) encoding. The names of the groups appear in capital letters and denote the semantics of the group.

As the quantity of the segments that makes up a group can differ from one message structure to another, the name of the structure is used as a prefix for the name of the group.

```
<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">
  ...
  ...
    <ADT_A01.INSURANCE>
      <IN1>
        ...
      </IN1>
    </ADT_A01.INSURANCE>
  </ADT_A01>
```

The part of the Schema that defines the segment group will be:

```
<xsd:element ref="ADT_A01.INSURANCE" minOccurs="0" maxOccurs="unbounded"/>
<xsd:complexType name="ADT_A01.INSURANCE.CONTENT">
    <xsd:sequence>
      <xsd:element ref="IN1"/>
      <xsd:element ref="IN2" minOccurs="0"/>
      <xsd:element ref="IN3" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="ROL" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
```

5.Segments and Fields

Here is part of the V2.8 definition of the message header (MSH) segment:

SEQ	LEN	C.LEN	DT	OPT	RP/#	TBL#	ITEM #	ELEMENT NAME
1	1..1		ST	R			00001	Field Separator
2	4..5		ST	R			00002	Encoding Characters
3			HD	O		0361	00003	Sending Application
4			HD	O		0362	00004	Sending Facility
5			HD	O		0361	00005	Receiving Application
6			HD			0362	00006	Receiving Facility
7			DTM	R			00007	Date/Time of Message
8		40=	ST	O			00008	Security
9			MSG	R			00009	Message Type
10	1..199	=	ST	R			00010	Message Control ID
11			PT	R			00011	Processing ID
12			VID	R			00012	Version ID
13			NM	O			00013	Sequence Number
14		180=	ST	O			00014	Continuation Pointer
15	2..2		ID	O		0155	00015	Accept Acknowledgment Type
16	2..2		ID	O		0155	00016	Application Acknowledgment Type
17	3..3		ID	O		0399	00017	Country Code
18	5..15		ID	O	Y	0211	00692	Character Set
19			CWE	O			00693	Principal Language Of Message
20	3..13		ID	O		0356	01317	Alternate Character Set Handling Scheme
21			EI	O	Y		01598	Message Profile Identifier
22			XON	O			01823	Sending Responsible Organization
23			XON	O			01824	Receiving Responsible Organization
24			HD	O			01825	Sending Network Address
25			HD	O			01826	Receiving Network Address

Using bar encoding, fields are located within the segments by their position. They are identified by the segment identifier followed by this position. For example, the field Message Type is MSH-9.

In v2.xml the elements of the fields are named of a similar way, using a point (.) as separator, and they are child elements of the segment element:

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

XML does not have the limitation of field sequence inherent with bars encoding. It is not necessary to represent empty fields with a place holder, unless it is used explicitly to inform the receiver that the preexisting data is being erased, in which case the text node of the field element will include the special string ("").

The Schema that defines the MSH segment is:

```
<xsd:element name="MSH" type="MSH.CONTENT"/>
<xsd:complexType name="MSH.CONTENT">
 <xsd:sequence>
            <xsd:element ref="MSH.1" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="MSH.2" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="MSH.3" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="MSH.4" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="MSH.5" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="MSH.6" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="MSH.7" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="MSH.8" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="MSH.9" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="MSH.10" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="MSH.11" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="MSH.12" minOccurs="1" maxOccurs="1"/>
            <xsd:element ref="MSH.13" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="MSH.14" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="MSH.15" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="MSH.16" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="MSH.17" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="MSH.18" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="MSH.19" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="MSH.20" minOccurs="0" maxOccurs="1"/>
            <xsd:element ref="MSH.21" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:any processContents="lax" namespace="##other" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```

A field is a character string defined by an HL7 data type. Appendix A of the specification contains the data dictionary, offering an alphabetical listing of the fields, listings of recommended encoding and a cross reference of fields and segments.

In the Schema, the component numbers, reference tables, long name and data type are indicated in **<appinfo>** elements. The long name is also indicated In the **<documentation>** element of the **<annotation>** element with the language in the attribute **xml:lang**. This information is also included in fixed attributes.

This is a fragment of the Schema defining the field MSH-10, Message Control ID:

```
<xsd:element name="MSH.10" type="MSH.10.CONTENT"/>
<xsd:complexType name="MSH.10.CONTENT">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">Message Control ID</xsd:documentation>
    <xsd:appinfo>
      <hl7:Item>10</hl7:Item>
      <hl7:Type>ST</hl7:Type>
      <hl7:LongName>Message Control ID</hl7:LongName>
```

```
      </xsd:appinfo>
    </xsd:annotation>
    <xsd:simpleContent>
      <xsd:extension base="ST">
        <xsd:attributeGroup ref="MSH.10.ATTRIBUTES"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

  <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:attributeGroup>
```

An application parsing a v2.xml message, valid against its Schema, can use the information contained in the **<appinfo>** and **<documentation>** elements of the Schema to perform some necessary processing.

6.Components and Subcomponents

Components are defined as the elements composing a field. Depending on the data type of that field, it can have one or several components.

For components of simple or primitive data types, the value is the text node of the corresponding element:

```
<MSH.10>NSI1</MSH.10>
```

When a component is defined by a composite or complex data type, each of its parts is called a subcomponent.

A child element is used for each individual subcomponent, naming them with the identifier of the data type and its position within the same, separated by a point (**.**).

In position 9 of the MSH segment is the field Message Type, whose data type is **MSG**. Its definition in the standard is:

```
MSH-9 Message Type (MSG) 00009

Components: <Message Code (ID)> ^ <Trigger Event (ID)> ^ <Message
Structure (ID)>
```

Its representation in XML encoding will then be:

```
<MSH.9>
    <MSG.1>ADT</MSG.1>
    <MSG.2>A01</MSG.2>
    <MSG.3>ADT_A01</MSG.3>
</MSH.9>
```

Empty components can be omitted. In the event that a single component has a value and the others are omitted, the element of the data type must be included anyway.

As with the fields, the **<appinfo>** and **<documentation>** elements in the Schema and fixed attributes are used to indicate the data type, long name, reference table, etc.

This is a fragment of the Schema of the composite data type MSG:

```
<xsd:complexType name="MSG">
  <xsd:sequence>
    <xsd:element ref="MSG.1" minOccurs="0" maxOccurs="1" />
  <xsd:element ref="MSG.2" minOccurs="0" maxOccurs="1" />
  <xsd:element ref="MSG.3" minOccurs="0" maxOccurs="1" />
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="MSG.1" type="MSG.1.CONTENT" />

<xsd:complexType name="MSG.1.CONTENT">
  <xsd:annotation>
```

```
    <xsd:documentation xml:lang="en">Message Code</xsd:documentation>
    <xsd:appinfo>
      <hl7:Type>ID</hl7:Type>
      <hl7:Table>HL70076</hl7:Table>
      <hl7:LongName>Message Code</hl7:LongName>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:simpleContent>
    <xsd:extension base="ID">
      <xsd:attributeGroup ref="MSG.1.ATTRIBUTES" />
   </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:attributeGroup name="MSG.1.ATTRIBUTES">
  <xsd:attribute name="Type" type="xsd:string" fixed="ID" />
  <xsd:attribute name="Table" type="xsd:string" fixed="HL70076" />
  <xsd:attribute name="LongName" type="xsd:string" fixed="Message Code" />
</xsd:attributeGroup>
```

7.HL7 v2.xml Schemas

HL7 provides the Schemas for V2.xml messages. They are organized in the following manner:

There is a Schema for each message structure: **ADT_A01.xsd**, **ORM_O01.xsd**, **ORU_R01.xsd**, etc. Each one of them show the definition of the XML document root element as a complex type and the definition of the complex type that establishes the sequence of the elements corresponding to the segments and groups of segments of the message. In each of these elements, its type is indicated by means of a reference (attribute **ref**) to another complex type, as well as its quantity. If the attribute **minOccurs** is 0, the segment is not required. Furthermore, the reference to the Schema defining the segments is included.

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

  <xsd:include schemaLocation="segments.xsd"/>

  <xsd:element name="ADT_A01" type="ADT_A01.CONTENT"/>

  <xsd:complexType name="ADT_A01.CONTENT">
    <xsd:sequence>
      <xsd:element ref="MSH"/>
      <xsd:element ref="SFT" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="EVN"/>
      ...
```

The Schema **segments.xsd** contains the definitions of the complex types that correspond to each of the segments of all the messages. These complex types contain a sequence with the definition of the fields of each segment. Again, the type of each field is indicated by means of a reference to another complex type. Its quantity is indicated as well. As the last element of the sequence an **<any>** element is included to specify that additional fields at the end of segment are allowed.

In this Schema the reference to the Schema file defining the fields is included.

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

  <xsd:include schemaLocation="fields.xsd"/>
      ...

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

  <xsd:complexType name="MSH.CONTENT">
    <xsd:sequence>
      <xsd:element ref="MSH.1"/>
      <xsd:element ref="MSH.2"/>
      <xsd:element ref="MSH.3" minOccurs="0"/>
      ...
```

```
        <xsd:any namespace="##other" processContents="lax" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
        ...
```

The Schema **fields.xsd** contains the definitions of the complex types corresponding to each of the fields of all the segments. These contain, in attributes and in an **<appinfo>** element, the declaration of the field data type, the field long name, its position within the segment and the reference table, if any. The reference to the Schema defining the data types is also included.

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

            <xsd:include schemaLocation="datatypes.xsd"/>
                ...

            <xsd:element name="MSH.15" type="MSH.15.CONTENT"/>

            <xsd:complexType name="MSH.15.CONTENT">
              <xsd:annotation>
                <xsd:documentation xml:lang="en">Accept Acknowledgment
            Type</xsd:documentation>
                <xsd:appinfo>
                  <hl7:Item>15</hl7:Item>
                  <hl7:Type>ID</hl7:Type>
                  <hl7:Table>HL70155</hl7:Table>
                  <hl7:LongName>Accept Acknowledgment Type</hl7:LongName>
                </xsd:appinfo>
              </xsd:annotation>
              <xsd:simpleContent>
                <xsd:extension base="ID">
                  <xsd:attributeGroup ref="MSH.15.ATTRIBUTES"/>
                </xsd:extension>
              </xsd:simpleContent>
            </xsd:complexType>

            <xsd:attributeGroup name="MSH.15.ATTRIBUTES">
              <xsd:attribute name="Item" type="xsd:string" fixed="15"/>
              <xsd:attribute name="Type" type="xsd:string" fixed="ID"/>
              <xsd:attribute name="Table" type="xsd:string" fixed="HL70155"/>
              <xsd:attribute name="LongName" type="xsd:string" fixed="Accept
            Acknowledgment Type"/>
            </xsd:attributeGroup>
        ...
```

The Schema **datatypes.xsd** contains the definitions of all the data types. For simple types, it establishes which primitive type it is. For the complex ones, it defines the sequence of subcomponents, and establishes their definition:

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

```
                ...

<xsd:simpleType name="ID">
  <xsd:restriction base="xsd:string"/>
</xsd:simpleType>
                ...

<xsd:complexType name="HD">
  <xsd:sequence>
    <xsd:element ref="HD.1" minOccurs="0"/>
    <xsd:element ref="HD.2" minOccurs="0"/>
    <xsd:element ref="HD.3" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
                ...

<xsd:element name="HD.1" type="HD.1.CONTENT"/>

<xsd:complexType name="HD.1.CONTENT">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">Namespace ID</xsd:documentation>
    <xsd:appinfo>
      <hl7:Type>IS</hl7:Type>
      <hl7:Table>HL70300</hl7:Table>
      <hl7:LongName>Namespace ID</hl7:LongName>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:simpleContent>
    <xsd:extension base="IS">
      <xsd:attributeGroup ref="HD.1.ATTRIBUTES"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:attributeGroup name="HD.1.ATTRIBUTES">
  <xsd:attribute name="Type" type="xsd:string" fixed="IS"/>
  <xsd:attribute name="Table" type="xsd:string" fixed="HL70300"/>
  <xsd:attribute name="LongName" type="xsd:string" fixed="Namespace ID"/>
</xsd:attributeGroup>
                ...
```

There is also a Schema called **Messages.xsd** that contains the inclusions of the Schemas of all the message structures.

With this Schema organization the logical sequence for undertaking a V2.xml encoding process would be to start from the message structure Schema. From this, one would determine what segments to use and take their definition from the segments Schema. For each field of each segment, one should take the definition from the fields Schema. On the basis of this, one would determine the corresponding data type and take its definition from the data types Schema.

8. Processing Rules

For v2.xml encoding, original processing rules and enhanced acknowledgement processing remain unchanged.

The application generating the v2.xml messages must ensure they are valid for their respective Schemas, but this does not force HL7 transactions to be validated at the moment. This will be a decision for each implementation.

On the other side, the receiving application will only have to verify that the XML message document is well-formed, but is not forced to validate it against its Schema. Again, this decision will be left up to each particular implementation.

The empty elements can be omitted. For the event in which the content of the corresponding field is null or blank, but the sense is to replace a preexisting non-null value, the element with a text node with two double quotation marks, that is to say, the empty string will be included:

```
<XAD.6>""</XAD.6>
```

The repetition of fields, segments and groups of segments is accomplished by repeating the corresponding elements.

```
<NK1.6>
   <XTN.1>4567-1234</XTN.1>
</NK1.6>
<NK1.6>
   <XTN.1>4987-9876</XTN.1>
</NK1.6>
```

There are different Schemas for each of the HL7 v2.x versions from the 2.3.1 onwards. When there are changes in the data types, compatibility between versions is not supported in the Schemas since the data types determine what elements will be the children of the field elements. Nevertheless, it is easy to use XSL transformations, if needed.

V2.xml does not provide a specified mechanism for message fragmentation and continuation. It assumes that systems capable of handling XML documents can support messages of any size.

Batch transfer of several messages is supported in v2.xml, using the corresponding control segments. The Schema **batch.xsd** provides the corresponding definitions.

9.Special Characters

The delimiter characters necessary for bar encoding lose meaning in XML encoding. Nevertheless, the elements **<MSH.1>** and **<MSH.2>**, with the fields corresponding to these delimiters are required. These become very important should there be a need to perform translations between encodings. For the character **&**, the XML entity **&** must be used.

Escape character sequences used to format text, for structures and character sets, must be handled in a different way than bar encoding given the characteristics and restrictions of XML.

For the text formatting sequences, like **\H**, the following elements are used:

\H	**<escape V="H"/>**	highlighted text
\H	**<escape V="N"/>**	end of highlighted text
\.br	**<escape V=".br"/>**	new output line
\.in+n	**<escape V=".in+n"/>**	Indent ***n*** spaces

This is a comparison between bar encoding and in XML:

```
A \H\special\H\ text

A <escape V="H"/>special<escape V="N"/> text
```

Escape character sequences used to indicate the delimiter characters of bar encoding, **\F**, **\S**, **\T** and **\R** have no meaning in XML and are not used.

V2.xml encoding does not use **\Xxxx** and **\Zxxx** to represent special characters. Instead, the standard XML form for characters reference is used: **&#xxx;.**

The character set interchange, defined in Chapter 2 of the HL7 Version 2 standard, cannot be used since an XML document can only have a single character set or encoding.

When XML specific delimiters must be included in the content of some field, it is the responsibility of the generating application to replace them with the corresponding entities. Normally, the parser of the receiving application will be able to handle these entities.

10. Local Extensions

Rules regarding extra information inserted at the end of the segments are valid for v2.xml encoding. Because of this, Schemas specifically define that any extra element in the final part of each segment is allowed:

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

<xsd:complexType name="MSH.CONTENT">
  <xsd:sequence>
    <xsd:element ref="MSH.1" minOccurs="1" maxOccurs="1"/>
  <xsd:element ref="MSH.2" minOccurs="1" maxOccurs="1"/>
...
...
    <xsd:element ref="MSH.20" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="MSH.21" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:any processContents="lax" namespace="##other" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

The **<any>** element highlighted in yellow indicates that any element can appear in this place of the complex type that is being defined. The value **##other** in attribute **namespace** indicates that elements of any namespace different from that the parent element belongs to are acceptable. The value **lax** in the attribute **processContents** instructs the processing application to attempt to obtain the Schema for the required namespace and validate its elements, but not to produce an error if it cannot be obtained.

For localized implementations, copies of the message Schemas can be redefined to support specific messages. The root element can be renamed to indicate that it is the localized version. Thus, for example, the definitions of Z segments can be included. It is a good practice to use a different namespace for the local implementation's proper elements.

11. Example

Finally, here is the ADT_A01 example message encoded in XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<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>NSI</HD.1>
      </MSH.3>
      <MSH.5>
         <HD.1>LAB</HD.1>
      </MSH.5>
      <MSH.7>
         <TS.1>20140327120759</TS.1>
      </MSH.7>
      <MSH.9>
         <MSG.1>ADT</MSG.1>
         <MSG.2>A01</MSG.2>
         <MSG.3>ADT_A01</MSG.3>
      </MSH.9>
      <MSH.10>NSI1</MSH.10>
      <MSH.11>
         <PT.1>P</PT.1>
      </MSH.11>
      <MSH.12>
         <VID.1>2.8</VID.1>
      </MSH.12>
      <MSH.16>AL</MSH.16>
   </MSH>

   <EVN>
      <EVN.1>A01</EVN.1>
      <EVN.2>
         <TS.1>18000101000000</TS.1>
      </EVN.2>
   </EVN>

   <PID>
      <PID.1>1</PID.1>
      <PID.3>
         <CX.1>55555</CX.1>
         <CX.5>HI</CX.5>
      </PID.3>
      <PID.4>
         <CX.1>55555555<CX.1>
         <CX.5>DNI</CX.5>
```

```
    </PID.4>
    <PID.5>
      <XPN.1>
        <FN.1>TEST</FN.1>
      </XPN.1>
      <XPN.2>AIDA</XPN.2>
    </PID.5>
    <PID.7>
      <TS.1>19780113000000</TS.1>
    </PID.7>
    <PID.8>F</PID.8>
    <PID.11>
      <XAD.1>
        <SAD.1>POTOSI 4032 108</SAD.1>
      </XAD.1>
      <XAD.3>BUENOS AIRES</XAD.3>
      <XAD.5>1899</XAD.5>
    </PID.11>
  </PID>

  <NK1>
    <NK1.1>1</NK1.1>
    <NK1.2>
      <XPN.1>
        <FN.1>CAMUS</FN.1>
      </XPN.1>
      <XPN.2>ALBERTO</XPN.2>
    </NK1.2>
    <NK1.3>
      <CE.1>PAD</CE.1>
    </NK1.3>
    <NK1.4>
      <XAD.1>
        <SAD.1>RIVADAVIA 253</SAD.1>
      </XAD.1>
    </NK1.4>
    <NK1.5>
      <XTN.1>42539686</XTN.1>
    </NK1.5>
  </NK1>

  <PV1>
    <PV1.1>1</PV1.1>
    <PV1.2>I</PV1.2>
    <PV1.3>
      <PL.1>12</PL.1>
      <PL.2>301</PL.2>
      <PL.3>1211</PL.3>
    </PV1.3>
    <PV1.4>R</PV1.4>
    <PV1.7>
      <XCN.1>1436</XCN.1>
```

```
      <XCN.2>
        <FN.1>PEREZ</FN.1>
        <FN.2>JORGE</FN.2>
        <FN.3>ALBERTO</FN.3>
      </XCN.2>
    </PV1.7>
    <PV1.8>
      <XCN.1>1026</XCN.1>
      <XCN.2>
        <FN.1>LOPEZ</FN.1>
        <FN.2>NORBERTO</FN.2>
      </XCN.2>
    </PV1.8>
    <PV1.9>
      <XCN.1>998</XCN.1>
      <XCN.2>
        <FN.1>GARCIA</FN.1>
        <FN.2>ALEJANDRO</FN.2>
      </XCN.2>
    </PV1.9>
    <PV1.10>M</PV1.10>
    <PV1.13>A</PV1.13>
    <PV1.14>4</PV1.14>
    <PV1.15>A0</PV1.15>
    <PV1.16>N</PV1.16>
    <PV1.17>
      <XCN.1>1026</XCN.1>
      <XCN.2>
        <FN.1>LOPEZ</FN.1>
        <FN.2>NORBERTO</FN.2>
      </XCN.2>
    </PV1.17>
    <PV1.18>OB</PV1.18>
    <PV1.19>
      <CX.1>H0100240</CX.1>
    </PV1.19>
    <PV1.36>ALV</PV1.36>
    <PV1.44>
      <TS.1>20140323095130</TS.1>
    </PV1.44>
    <PV1.45>
      <TS.1>20140323102455</TS.1>
    </PV1.45>
  </PV1>

  <PV2>
    <PV2.4>
      <CE.1>SUB INTESTINAL</CE.1>
    </PV2.4>
    <PV2.23>
      <XON.3>223</XON.3>
    </PV2.23>
```

```
      <PV2.36>N</PV2.36>
    </PV2>

    <ADT_A01.INSURANCE>
      <IN1>
        <IN1.1>1</IN1.1>
        <IN1.2>
          <CE.1>INT</CE.1>
          <CE.3>HI</CE.3>
        </IN1.2>
        <IN1.3>
          <CX.1>2</CX.1>
          <CX.5>HI</CX.5>
        </IN1.3>
        <IN1.3>
          <CX.1>347</CX.1>
          <CX.5>NSI</CX.5>
        </IN1.3>
        <IN1.4>
          <XON.1>PLAN DE SALUD</XON.1>
        </IN1.4>
      </IN1>
    </ADT_A01.INSURANCE>
</ADT_A01>
```

This XML document version of the ADT_01 example message is obviously much longer than its equivalent in "vertical bar" encoding. However, the blank spaces and line feeds are not necessary; they have only been included here for ease of human reading. Furthermore, the text can be compressed and generally compressed XML documents are more near the size of the bar messages. While this might be considered a disadvantage for some implementations, it will have to be compared against the advantages provided by this encoding scheme.

Unit Summary and Conclusion

V2.xml represents a great alternative to traditional "pipes and hats" encoding. It offers advantages leading to a more standard integration with other systems outside the strict scope of healthcare and has more tools for its processing.

However, it is not without some disadvantage, mainly the relative size of the messages.

The translation between either encoding is clearly possible. The standard does not specify a particular way to perform this translation. Although it is not trivial, it is not very complex either. Which is the more suitable encoding method should be evaluated with each local implementation.

Additional Reading and Reference Material

HL7 Version 2

XML Encoding Syntax ANSI/HL7 V2 XML-2003
HL7 Version 2.3.1, ANSI/HL7 V2.3.1-1999,
HL7 Version 2.4, ANSI/HL7 V2.4-2000,
HL7 Version 2.5, HL7 V2.5-2002,

Note: You can download the HL7 V2.x standards free from the above URLs on the HL7 International Web site (www.HL7.org) after you create a free log-in account.

Extensible Markup Language (XML)

http://www.w3.org/TR/REC-xml

XML Schema

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/

XML Namespace

http://www.w3.org/TR/REC-xml-names/

XML Path Language (XPath)

http://www.w3.org/TR/xpath