HL7 Recommendation:
Using XML as a Supplementary Messaging Syntax for HL7 Version 2.3.1

HL7 XML Special Interest Group
Informative Document

Editors:
Robert H. Dolin, MD (Robert.H.Dolin@kp.org**)**
Paul V. Biron, MLIS (Paul.V.Biron@kp.org**)**

I. OBJECTIVES

Chapter 2 of the HL7 2.3.1 specifies standard encoding rules for HL7 message instances. The objective of this report is to present XML encoding rules for HL7 Version 2.3.1 messages that could be used in environments where senders and receivers both understand XML. It is not the intent of this recommendation to replace the standard encoding rules, but if you are going to use XML, this is the way HL7 recommends that you do it.

Many XML encodings could serve as alternate messaging syntaxes for HL7 Version 2.3.1 messages. This document recommends one such encoding of HL7 Version 2.3.1 messages in XML. An algorithm is presented that will translate normative diagrams and tables within the HL7 Standard to an XML Document Type Definition (DTD).

The report also addresses the translation between standard encoded and XML encoded HL7 Version 2.3.1 messages.

II. INTRODUCTION

A. 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 [CEN, 1993]. A subsequent report extended the CEN study to look at SGML [Dolin, 1997]. 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 [XML, 1998] 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 [Dolin, 1998]. In 1999, Wes Rishel coordinated a 10-vendor HL7-XML interoperability demonstration at the annual HIMSS Conference. All vendors rated the demo a success.

Based on experience to date, XML shows promise as an interchange format for HL7 messages. Findings have shown that XML can serve as an implementable message specification for HL7 Version 2.3.1 messages and that the ability to explicitly represent an HL7 requirement in XML confers the ability to validate that requirement with an XML parser.

For more information on SGML and XML, see the references ([Clark], [Cover], [IBM], [Megginson, 1998], [Microsoft], [St.Laurent, 1999], [XML.com], [XML.org])

B. Introduction to the XML representation

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

C. Example Message

Here we see a simple message in the syntax of the standard encoding rules.

MSH|^~\&|LAB^foo^bar|767543|ADT|767543|19900314130405||ACK^|XX3657|P|2.3.1<CR>
MSA|AA|ZZ9380<CR>

Here we see the same message in the syntax of the recommended XML encoding rules.

```
<!DOCTYPE ACK SYSTEM "hl7_v231.dtd">
<ACK>
<MSH>
        <MSH.1>|</MSH.1>
        <MSH.2>^~\&amp;amp;</MSH.2>
        <MSH.3>
                <HD.1>LAB</HD.1>
                <HD.2>foo</HD.2>
                <HD.3>bar</HD.3>
        </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>19900314130405</MSH.7>
        <MSH.9><CM_MSG_TYPE.1>ACK</CM_MSG_TYPE.1></MSH.9>
        <MSH.10>XX3657</MSH.10>
        <MSH.11><PT.1>P</PT.1></MSH.11>
        <MSH.12><VID.1>2.3.1</VID.1></MSH.12>
</MSH>
<MSA>
        <MSA.1>AA</MSA.1>
        <MSA.2>ZZ9380</MSA.2>
</MSA>
</ACK>
```

As is always the case with XML when processed with a validating processor, the extra whitespace between elements (provided to make the message easier for people to read) can be removed in actual message instances, resulting is shorter messages in situations when overall message length is a factor.

A longer example message is included in Section "Long Example Message".

D. Overview of Normative HL7 Database

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

This database arose as the German HL7 user group undertook careful analysis of the Standard. They became aware that the chapters of the Standard had been developed by different groups, and that there had been no distinct rules or guidelines for the development of various parts of the Standard. They therefore defined a comprehensive database of the HL7 Standard (including Version 2.1 through Version 2.3.1) to allow consistency checks of items and to support the application of the Standard by the user.

Within the HL7 Database, all data added is checked for its consistency. Referential integrity among relations assures this consistency. The side effect of referential integrity is to modify the data from the standard documents because the standard is defined in the form of a document but not in the form of a database.

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

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

The XML representation of HL7 messages presented here is algorithmically derived directly from the HL7 Database. Therefore, ambiguities or errors in the Standard are reflected "as is" in the XML encoding. Fixing any such errors in the XML will require making appropriate modifications to the HL7 Database. (The issues we identified with the HL7 Database are summarized below in Section "Appendix - Issues Identified with the HL7 Database", and have all been submitted back to HL7 for consideration.)

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 [Oemig, 1996] [Oemig].

E. 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 representation presented in this report.

1. XML DTD Optimization

XML DTD 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.

a) Message Length

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

The greater the percentage of data characters (as opposed to markup characters) in an average message, the less important any additional overhead imposed by changing from the standard HL7 encoding rules to XML becomes. Data from the Duke HL7 production environment suggests that on average, data characters comprise about 70% of overall message length. (Data from Duke courtesy of Al Stone, and posted to the HL7 SGML/XML SIG List Server 1/15/98 and 1/16/98.) As a result, we anticipate that the XML encoding recommended here will result in messages that are approximately 40% to 100% longer, although this estimate has yet to be subjected to rigorous testing.

b) Structural Complexity

Krueger [Krueger] 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". We have listed potential components of this metric 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 from another.
- Message Debugging : Determine why an application is generating an HL7-invalid message.
- Message Filtering : Filtering might include sending only a subset of the message to a particular message receiver.
- Message Routing : Routing includes extracting from the message what is necessary to determine where to send it.
- Message Parsing : Parsing can include message validation and extraction of field values and data type components.

2. Localization

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

- Allow senders to introduce local variations into standard HL7 messages where necessary.

- Allow receivers to use well-formed XML processors or validating XML processors. Receivers using validating processors should not have to fall back to using a non-validating processor in those cases when the sender includes localized content in their messages.

3. "Looseness" of the DTD

XML is a formal grammar that can be used to encode HL7 business rules. When an XML processor validates that a message is valid per its DTD, it is also validating that a message is conformant to those HL7 rules that are explicitly represented in the XML DTD. Some HL7 rules are easy to explicitly represent within an XML DTD, such as the optionality and repetition of a field within a segment. Some HL7 rules are difficult to explicitly represent within an XML DTD, such as the maximum allowable field length. Representing such rules within a DTD, while possible, may conflict with other design considerations. Therefore, determining the "looseness" of the DTD, or the degree to which HL7 business rules are explicitly represented in the DTD, is itself a design consideration.

A very loose DTD might look something like this:

```
<!ELEMENT MESSAGE (SEGMENT+)>
<!ATTLIST MESSAGE  MESSAGE.ID CDATA #IMPLIED>

<!ELEMENT SEGMENT (FIELD+)>
<!ATTLIST SEGMENT  SEGMENT.ID CDATA #IMPLIED>

<!ELEMENT FIELD (DATATYPE)>
<ATTLIST FIELD  FIELD.ID CDATA #IMPLIED>

<!ELEMENT DATATYPE (#PCDATA)>
<!ATTLIST DATATYPE  DATATYPE.ID CDATA #IMPLIED)
```

You can carry HL7-valid messages in this construct, but you can also carry a lot of HL7-invalid messages. An XML processor can't do much to help validate that a message received is a valid HL7 message. The decision in the XML representation presented here is to capture as many HL7 business rules as reasonably possible. 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).

4. Conformance with HL7 Version 3 XML Representation

The XML messaging syntax in this document is not the same as that which is being proposed for Version 3.0 [HL7 V3/XML]. The Version 3 XML project is a moving target, and consensus on XML representation has not yet been reached. Given the different philosophies underlying Versions 2.3.1 and 3.0, there will probably be some differences.

III. ALGORITHMS

The current mapping from HL7 V2.3.1 into XML is a formal algorithm, driven from the HL7 Database described above. As such, ambiguities or errors in the Standard are reflected as is in the XML encoding. Fixing any such errors in the XML will require making appropriate modifications to the HL7 Database. (Prior to running the algorithms described in this section, we modified the HL7 Database as described below in Section "Appendix - Issues Identified with the HL7 Database".)

SQL queries are applied to the HL7 Database to extract tables containing definitions of messages, segments, fields, and data types. These tables are exported into ASCII delimited files. Perl scripts are applied to the ASCII delimited files to generate XML DTDs. The structure of the generated DTDs follows from the design considerations described above.

We provide two sets of DTDs:

- A single DTD (hl7_v231.dtd) that contains all HL7 V2.3.1 definitions. This file is logically broken up into a DTD containing all message declarations (messages.dtd), a DTD containing all segment declarations (segments.dtd), a DTD containing all field declarations (fields.dtd), and a DTD containing all data type declarations (datatypes.dtd).
- One DTD for each message structure. Each of these DTDs (e.g. "ACK.dtd", "MFN_M05.dtd"), imports the same datatype DTD referenced by hl7_v231.dtd, but is otherwise self contained with all message, segment, and field declarations necessary for a particular message structure.

Message instances can be validated against either "hl7_v231.dtd" or against the DTD for that particular message structure, by specifying the desired DTD in the DOCTYPE declaration of the message instance.

The following table summarizes the files that, along with this document, comprise the work products of this recommendation. All of these files are included in the zip file “hl7v231xml.FINAL.zip”, available on www.HL7.org.

HL7 Structure	ASCII delimited file	Perl script	DTD
Messages	messages.txt	messages.pl	messages.dtd; one DTD per message
Segments	fields.txt*	segments.pl	segments.dtd
Fields	fields.txt*	fields.pl	fields.dtd
Data types	datatypes.txt	datatypes.pl	datatypes.dtd

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

The structure of these ASCII delimited files is described below in "Extracting Subsets Of The Normative HL7 Database ". The algorithms instantiated in these Perl scripts are described below in "Algorithms For XML DTD Generation From The Extracted Subsets".

A. Extracting Subsets Of The Normative HL7 Database

1. Messages and their segments

While developing the analytic object model for the definition of a comprehensive HL7 Database, the German HL7 user group became aware that the relationship between message types, event types, and the structure of a message is problematic and not handled satisfactorily in the Standard. There is no distinct rule defining the relationship between events and messages. Sometimes the event type describes the structure and function of the message. In other cases, it defines the target file the data in the message has to be transmitted to.

As noted above, ambiguities or errors in the Standard are reflected "as is" in the XML encoding. Fixing any such errors in the XML will require making appropriate modifications to the HL7 Database. Our approach to representing messages and their contained segments is detailed here.

a) HL7 Database tables used

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

EventMessageTypeSegments : Table

Primary Key	Field Name	Data Type	Description
*	event_code	Text-3	Event-Code
*	hl7_version	Text-8	version number
*	message_type	Text-3	Type of this Message
*	lfd_nr	Integer	consecutive increasing number used for 1:n relation
	seg_code	Text-3	Segment-Code
	repetitional	Yes/No	Repetitional
	optional	Yes/No	Optional

MsgStructIDSegments : Table

Primary Key	Field Name	Data Type	Description
*	message_structure	Text-7	Message Structure ID
*	hl7_version	Text-8	version number
*	lfd_nr*	Integer	consecutive increasing number used for 1:n relation
	seg_code	Text-3	Segment-Code
	repetitional	Yes/No	Repetitional
	optional	Yes/No	Optional

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

b) SQL query

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

SELECT [message_type] + "_" + [event_code] AS MsgStruct, lfd_nr, seg_code, repetitional, optional
FROM EventMessageTypeSegments
WHERE hl7_version = "2.3.1"

UNION

SELECT message_structure, lfd_nr, seg_code, repetitional, optional
FROM MsgStructIDSegments
WHERE hl7_version = "2.3.1";

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

MsgStructUnion : Table

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

The results of the union query are copied into MsgStructUnion.

c) Select from MsgStructUnion those message structures reflected in Table 0354

Chapter 2 of HL7 V2.3.1 "Control/Query" describes the message header (MSH) segment. Field 9 "Message Type" (MSH.9) contains the message type, trigger event, and the message structure ID for the message. The third component of MSH.9 is the abstract message structure code defined by HL7 Table 0354 - Message structure. We extract from MsgStructUnion those message structures contained in Table 0354 using this SQL query:

SELECT MsgStruct, lfd_nr, seg_code, repetitional, optional
FROM TableValues INNER JOIN MsgStructUnion ON TableValues.table_value = MsgStructUnion.MsgStruct
WHERE (table_id=354) AND (hl7_version="2.3.1")
ORDER BY MsgStruct, lfd_nr;

This resulting table is exported to **messages.txt**, a subset of which is shown here:

MsgStruct	lfd_nr	seg_code	repetitional	Optional
ACK	1	MSH	No	No
ACK	2	MSA	No	No
ACK	3	ERR	No	Yes
ADT_A02	1	MSH	No	No
ADT_A02	2	EVN	No	No
ADT_A02	3	PID	No	No
ADT_A02	4	[	No	No
ADT_A02	5	PD1	No	No
ADT_A02	6	]	No	No
ADT_A02	7	PV1	No	No
ADT_A02	8	[	No	No
ADT_A02	9	PV2	No	No
ADT_A02	10	]	No	No
ADT_A02	11	[	No	No
ADT_A02	12	{	No	No

ADT_A02	13	DB1	No	No
ADT_A02	14	}	No	No
ADT_A02	15	]	No	No
ADT_A02	16	[	No	No
ADT_A02	17	{	No	No
ADT_A02	18	OBX	No	No
ADT_A02	19	}	No	No
ADT_A02	20	]	No	No

2. Segments, fields, and data types

a) HL7 Database tables used

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

SegmentDataElements : Table

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

DataElements : Table

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

b) SQL query

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

SELECT seg_code, lfd_nr, SegmentDataElements.data_item, description, data_structure, req_opt, repetitional, table_id
FROM DataElements INNER JOIN SegmentDataElements ON (DataElements.hl7_version = SegmentDataElements.hl7_version) AND (DataElements.data_item = SegmentDataElements.data_item)
WHERE SegmentDataElements.hl7_version="2.3.1"
ORDER BY seg_code, lfd_nr;

This resulting table is exported to **fields.txt**, a subset of which is shown here:

seg_code	lfd_nr	data_item	description	data_structure	Req_opt	repetitional	table_id
ACC	1	00527	Accident Date/Time	TS	O		0000
ACC	2	00528	Accident Code	CE	O		0050

ACC	3	00529	Accident Location	ST	O		0000
ACC	4	00812	Auto Accident State	CE	O		0347
ACC	5	00813	Accident Job Related Indicator	ID	O		0136
ACC	6	00814	Accident Death Indicator	ID	O		0136

3. Data types and their data type components

a) HL7 Database tables used

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

DataStructures : Table

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

DataStructureComponents : Table

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

Components : Table

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

b) SQL query

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

SELECT DataStructures.data_structure, lfd_nr, DataStructures.description, DataStructureComponents.table_id, Components.description, Components.table_id, Components.data_type_code
FROM DataStructures LEFT JOIN (DataStructureComponents LEFT JOIN Components ON (DataStructureComponents.comp_nr = Components.comp_nr) AND (DataStructureComponents.hl7_version = Components.hl7_version)) ON (DataStructures.hl7_version = DataStructureComponents.hl7_version) AND (DataStructures.data_structure = DataStructureComponents.data_structure)

WHERE DataStructures.hl7_version="2.3.1"
ORDER BY DataStructures.data_structure, lfd_nr;

This resulting table is exported to **datatypes.txt**, a subset of which is shown here:

DataStructures.data_structure	DataStructureComponents.lfd_nr	DataStructures.description	DataStructureComponents.table_id	Components.description	Components.table_id	Components.data_type_code
AD	2	address	0000	other designation	0000	ST
AD	3	address	0000	city	0000	ST
AD	4	address	0000	state or province	0000	ST
AD	5	address	0000	zip or postal code	0000	ST
AD	6	address	0000	country	0000	ID
AD	7	address	0000	address type	0190	ID
AD	8	address	0000	other geographic designation	0000	ST
AD	1	address	0000	street address	0000	ST
ST		string data				
WILDCARD		jeder Datentyp ist einsetzbar				

B. Algorithms For XML DTD Generation From The Extracted Subsets

1. Messages contain segments

a) Introduction

This section describes the creation of **messages.dtd** from **messages.txt**. The algorithm is instantiated in **messages.pl**.

The Abstract Message Syntax in HL7 V2.3.1 specifies the arrangement of segments within a message, as shown in the following imaginary WRP message:

WRP	**Widget Report**
MSH	Message Header
MSA	Message Acknowledgment
{ WDN	Widget Description
WPN	Widget Portion
{ [WPD]}	Widget Portion Detail
}	

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

HL7 Abstract Message Syntax	Equivalent XML Occurrence Indicator
[ ]	? (zero or one)
{ }	+ (one or more)
{[ ]} = [{ }]	* (zero or more)
-no bracket or brace-	-no occurrence indicator- (one exactly)

Standard HL7 encoding rules flatten this out, such that one valid message might look something like:

MSH|...
MSA|...
WDN|...
WPN|...
WPD|....

WPD|...
WDN|...
WPN|...

In this recommendation, we've introduced grouping and list elements, similar to what is being proposed for HL7 Version 3 messages (irrespective of XML), to allow the wire format to reflect the inherent abstract hierarchy. A resulting DTD fragment for the above would be:

```
<!ELEMENT WRP (MSH,MSA,WRP.LST.2)>
<!ELEMENT WRP.LST.1 (WPD)+>
<!ELEMENT WRP.LST.2 (WRP.GRP.1)+>
<!ELEMENT WRP.GRP.1 (WDN,WPN,WRP.LST.1?)>
```

and a valid message might look something like:

```
<WRP>
        <MSH>...</MSH>
        <MSA>...</MSA>
        <WRP.LST.2>
                <WRP.GRP.1>
                        <WDN>...</WDN>
                        <WPN>...</WPN>
                        <WRP.LST.1>
                                <WPD>...</WPD>
                                <WPD>...</WPD>
                        </WRP.LST.1>
                </WRP.GRP.1>
                <WRP.GRP.1>
                        <WDN>...</WDN>
                        <WPN>...</WPN>
                </WRP.GRP.1>
        </WRP.LST.2>
</WRP>
```

b) Detailed Algorithm

The example WRP message shown here:

WRP	**Widget Report**
MSH	Message Header
MSA	Message Acknowledgment
{ WDN	Widget Description
WPN	Widget Portion
{ [WPD]}	Widget Portion Detail
}	

would look like this in **messages.txt**, as described above in "Messages and their segments":

MsgStruct	lfd_nr	seg_code	repetitional	optional
WRP	1	MSH	No	No
WRP	2	MSA	No	No
WRP	3	{	No	No
WRP	4	WDN	No	No
WRP	5	WPN	No	No
WRP	6	{	No	No

WRP	7	WPD	No	Yes
WRP	8	}	No	No
WRP	9	}	No	No

The algorithm walks through each row that has the same value of MsgStruct in order to build a linear message definition. During the walk through, if the value of repetitional = "Yes" then wrap the value of seg_code with braces ("{" and "}"), and if the value of optional = "Yes" then wrap the value of seg_code with brackets ("[" and "]"). The resulting linear definition of WRP becomes:

MSH MSA { WDN WPN { [WPD] } }

Next, add commas. Following a segment name, insert a comma before the next segment name, open bracket (“[“), or open brace (“{“):

MSH,MSA,{ WDN,WPN,{ [WPD]} }

Next, braces and brackets are converted to parenthesis and XML occurrence indicators.

- Convert open braces (“{“) and open brackets (“[“) to open parentheses (“(“).
- Convert close braces (“}”) to close parenthesis followed by a plus-sign (“)+”).
- Convert close brackets (“]”) to close parenthesis followed by a question mark (“)?”).

The result:

MSH,MSA,(WDN,WPN,((WPD)?)+)+

In some cases "?" and "+" XML occurrence indicators can be merged.

- ((xxx)?)+ = ((xxx)+)? = (xxx)*

The result:

MSH,MSA,(WDN,WPN,(WPD)*)+

Next, we deepen the XML element hierarchy by adding listing and grouping elements where there are "+" and "*" occurrence indicators and where there is more than one element in a list:

- **LIST**: Content model has 1 element, with a "+" occurrence indicator.
  - Balanced parentheses with an outer "+" or "*" become a new LIST element.
  - LIST element replaces what is contained in parenthesis and the occurrence indicator.
    - If occurrence indicator is "*", add a "?" occurrence indicator to the new LIST element.
  - Content model of LIST element is what is contained in parenthesis along with a "+" occurrence indicator.
- **GROUP**: Content model has >1 element, none of which have have "+" or "*" occurrence indicator.
  - If content model of a LIST element has > 1 element, create a GROUP element container.

<!ELEMENT WRP (MSH,MSA,(WDN,WPN,(WPD)*)+)>

becomes…

<!ELEMENT WRP (MSH,MSA,(WDN,WPN,WRP.LST.1?)+)>
<!ELEMENT WRP.LST.1 (WPD)+>

becomes…

<!ELEMENT WRP (MSH,MSA,WRP.LST.2)>
<!ELEMENT WRP.LST.1 (WPD)+>
<!ELEMENT WRP.LST.2 (WDN,WPN,WRP.LST.1?)+>

becomes…

<!ELEMENT WRP (MSH,MSA,WRP.LST.2)>
<!ELEMENT WRP.LST.1 (WPD)+>
<!ELEMENT WRP.LST.2 (WRP.GRP.1)+>
<!ELEMENT WRP.GRP.1 (WDN,WPN,WRP.LST.1?)>

2. Segments contain fields

a) Introduction

This section describes the creation of **segments.dtd** from **fields.txt**. The algorithm is instantiated in **segments.pl**.

Segments contain fields. Each field has an ordinal position within a segment. Optionality and repetition are specified for each field.

b) Detailed Algorithm

The example FOO segment would look like this in **fields.txt**, as described above in "Segments, fields, and data types":

seg_code	lfd_nr	data_item	description	data_structure	req_opt	repetitional	table_id
FOO	1	9876	Acknowledgment Code	ST	O	N	0000
FOO	2	9875	Message Control ID	ST	O	Y	0000
FOO	3	9874	Text Message	CE	O	Y	0933
FOO	4	9873	Expected Sequence Number	CX	O	N	0000

The algorithm walks through each row that has the same value of seg_code in order to build a linear segment definition. In the process, XML occurrence indicators are derived from the values of req_opt and repetitional. Within the HL7 Database, values for req_opt include: B, (B) R, C, O, R, X, Y, and NULL; and values for repetitional include: N, Y, and NULL. Each of these values is defined in Chapter 2 of the HL7 V2.3.1 Standard. The XML occurrence indicator is derived from the values of req_opt and repetitional as shown in the following table:

req_opt	repetitional	Mapped XML Occurrence Indicator
(B) R, R	N, NULL	-none-
B,C,O,X,Y, NULL	N, NULL	?
(B) R, R	Y	+
B,C,O,X,Y, NULL	Y	*

resulting in:

<!ELEMENT FOO (FOO.1?, FOO.2*, FOO.3*, FOO.4?)>

Next, the XML element hierarchy is deepened by adding listing elements (similar to the process described above) where there are "+" and "*" occurrence indicators, resulting in:

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

3. Fields contain data types

a) Introduction

This section describes the creation of **fields.dtd** from **fields.txt**. The algorithm is instantiated in **fields.pl**.

Each field has an official field identifier, a long name, and a data type. Some fields specify an HL7 Table that contains enumerated values for use in the field.

b) Detailed Algorithm

The example FOO segment would look like this in **fields.txt**, as described above in "Segments, fields, and data types":

seg_code	lfd_nr	data_item	description	data_structure	req_opt	repetitional	table_id
FOO	1	9876	Acknowledgment Code	ST	O	N	0000
FOO	2	9875	Message Control ID	ST	O	Y	0000
FOO	3	9874	Text Message	CE	O	Y	0933
FOO	4	9873	Expected Sequence Number	CX	O	N	0000

The algorithm creates an XML element for each field. The content model of each field is a parameter entity reference to the field's data type. Fixed attributes are used to express the field identifier, data type, long name, and table, as shown here:

```
<!ELEMENT FOO.3 %CE;>
<!ATTLIST FOO.3
   Item CDATA #FIXED '9874'
   Table CDATA #FIXED '933'
   LongName CDATA #FIXED 'Text Message'
   Type CDATA #FIXED 'CE'>
```

Certain characters within the HL7 Database must be "escaped" before inclusion in a DTD. The ampersand is a reserved XML metacharacter. Where an ampersand occurs in the long name of a field, it is converted to an XML entity representation "&" (e.g. "Critical Range for Ordinal & Continuous Obs" becomes "Critical Range for Ordinal & Continuous Obs"). Because the DTD wraps the value of attribute LongName in single quotes, when a single quote occurs in the long name of a field, it is converted to an XML entity representation "'" (e.g. "Contact's Tel. Number" becomes "Contact's Tel. Number"). Because data types become XML element names (as described in the section that follows), internal plus-signs ("+") are not allowed. There is a data type in the HL7 Database (CE_TABS_136+262+263) that contains internal plus-signs ("+"). These are converted to underscores ("_") (e.g. "CE_TABS_136+262+263" becomes "CE_TABS_136_262_263").

4. Data types contain data type components

a) Introduction

This section describes the creation of **datatypes.dtd** from **datatypes.txt**. The algorithm is instantiated in **datatypes.pl**.

Some data types are composite. 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.

Some data types are primitive, in which case they have no components.

b) Detailed Algorithm

A subset of data types found in **datatypes.txt** is shown here, as described above in "Data types and their data type components":

DataStructures.data_structure	DataStructureComponents.lfd_nr	DataStructures.description	DataStructureComponents.table_id	Components.description	Components.table_id	Components.data_type_code
AD	2	address	0000	other designation	0000	ST
AD	3	address	0000	city	0000	ST
AD	4	address	0000	state or province	0000	ST
AD	5	address	0000	zip or postal code	0000	ST
AD	6	address	0000	country	0000	ID
AD	7	address	0000	address type	0190	ID
AD	8	address	0000	other geographic designation	0000	ST
AD	1	address	0000	street address	0000	ST
ST		string data				
WILDCARD		jeder Datentyp ist einsetzbar				

The algorithm walks through each row that has the same value of DataStructures.data_structure in order to build a linear data type definition.

Data types are modeled as parameter entities reflecting their data type components, as shown here:

<!ENTITY % AD "(AD.1?,AD.2?,AD.3?,AD.4?,AD.5?,AD.6?,AD.7?,AD.8?)">

All data type components are modeled as optional. Data types having a null value for DataStructureComponents.lfd_nr are primitive, as shown here:

<!ENTITY % ST "(#PCDATA)">

Data type components of composite data types are modeled similarly to fields. The content model of each component is a parameter entity reference to the component's data type. Fixed attributes are used to express the component data type, long name, and table, as shown here:

```
<!ELEMENT AD.1 %ST; >
<!ATTLIST AD.1
   LongName CDATA #FIXED 'street address'
   Type CDATA #FIXED 'ST'
   Table CDATA #FIXED '0'>
```

In the HL7 Database, data type components (as well as fields) specify an HL7 Table that contains enumerated values for use in the component. Per the HL7 Database, if the value of DataStructureComponents.table is not zero, it overrides the value in Components.table.

The data type "WILDCARD" is modeled as:

<!ENTITY % WILDCARD "(ANY)">

Certain characters within the HL7 Database must be "escaped" before inclusion in a DTD. The ampersand is a reserved XML metacharacter. Where an ampersand occurs in the long name of a component, it is converted to an XML entity representation "&" (e.g. "value1&value2&value3" becomes "value1&value2&value3"). Because data types become XML element names, internal plus-signs ("+") are not allowed. There is a data type in the HL7 Database (CE_TABS_136+262+263) that contains internal plus-signs ("+"). These are converted to underscores ("_") (e.g. "CE_TABS_136+262+263" becomes "CE_TABS_136_262_263").

C. Localization

The HL7 Standard describes the responsibilities for parties sending and receiving HL7 messages. These responsibilities enable exchange of messages that contain localizations (or local variations or z-segments). This section describes how the DTD is crafted such that:

- 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.

- 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 DTD within the internal subset of a transmitted message instance.

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

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

We state an equivalent content model like this:

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

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

```
<!DOCTYPE ORU_RO1 SYSTEM "hl7_v231.dtd" [
  <!ENTITY % FOO.CONTENT "(FOO.1?, FOO.2.LST?, FOO.3.LST?, FOO.4?, FOO.5?)">
  <!ELEMENT FOO.5 (#PCDATA)> ]>
```

Receivers using non-validating XML processors can ignore the entire DOCTYPE declaration.

Senders are not required to create or provide an explicit representation of the transformation from the localized DTD to an HL7 standard DTD. (Architectural forms do represent one way of expressing the transformation in a machine-processable format. It is anticipated that other representations will also be possible.)

IV. DTDs And Message Instances

As noted above, we provide two sets of DTDs:

- A single DTD (hl7_v231.dtd, shown below) that contains all HL7 V2.3.1 definitions. This file is logically broken up into a DTD containing all message declarations (messages.dtd), a DTD containing all segment declarations (segments.dtd), a DTD containing all field declarations (fields.dtd), and a DTD containing all data type declarations (datatypes.dtd).
- One DTD for each message structure. Each of these DTDs (e.g. "ACK.dtd", "MFN_M05.dtd"), imports the same datatype DTD referenced by hl7_v231.dtd, but is otherwise self contained with all message, segment, and field declarations necessary for a particular message structure.

Message instances can be validated against either "hl7_v231.dtd" or against the DTD for that particular message structure, by specifying the desired DTD in the DOCTYPE declaration of the message instance.

A. HL7 Version 2.3.1 DTD

This is the complete HL7 V2.3.1 DTD. It references the other modular DTDs described above, and assumes that all DTDs are present in the same directory.

```
<!ENTITY % HL7V231-datatypes PUBLIC
        "-//HL7//DTD HL7 V2.3.1 datatype definitions//EN"
        "datatypes.dtd" >
%HL7V231-datatypes;

<!ENTITY % HL7V231-fields PUBLIC
        "-//HL7//DTD HL7 V2.3.1 field definitions//EN"
        "fields.dtd" >
%HL7V231-fields;

<!ENTITY % HL7V231-segments PUBLIC
        "-//HL7//DTD HL7 V2.3.1 segment definitions//EN"
        "segments.dtd" >
%HL7V231-segments;

<!ENTITY % HL7V231-messages PUBLIC
        "-//HL7//DTD HL7 V2.3.1 message definitions//EN"
        "messages.dtd" >
%HL7V231-messages;
```

B. Example DTD Fragment

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

All content models and attribute lists are defined as parameter entities, for reasons described above in "Localization".

```
<!-- Data type definitions -->
<!ENTITY % CE "(CE.1?,CE.2?,CE.3?,CE.4?,CE.5?,CE.6?)">

<!ENTITY % CE.1.CONTENT "%ST;">
<!ELEMENT CE.1 %CE.1.CONTENT;>
<!ENTITY % CE.1.ATTRIBUTES
  "LongName CDATA #FIXED 'identifier'
  Type CDATA #FIXED 'ST'
  Table CDATA #FIXED '0'">
<!ATTLIST CE.1 %CE.1.ATTRIBUTES;>


<!-- Field definitions -->
<!ENTITY % MSA.6.CONTENT "%CE;">
<!ELEMENT MSA.6 %MSA.6.CONTENT;>
<!ENTITY % MSA.6.ATTRIBUTES
  "Item CDATA #FIXED '23'
  Table CDATA #FIXED '0'
  LongName CDATA #FIXED 'Error Condition'
  Type CDATA #FIXED 'CE'">
<!ATTLIST MSA.6  %MSA.6.ATTRIBUTES;>


<!-- Segment definitions -->
```

```
<!ENTITY % MSH.18.LST.CONTENT "(MSH.18)+">
<!ELEMENT MSH.18.LST %MSH.18.LST.CONTENT;>
<!ENTITY % MSH.CONTENT
"(MSH.1,MSH.2,MSH.3?,MSH.4?,MSH.5?,MSH.6?,MSH.7?,MSH.8?,MSH.9,MSH.10,MSH.11,MSH.12,
MSH.13?,MSH.14?,MSH.15?,MSH.16?,MSH.17?,MSH.18.LST?,MSH.19?,MSH.20?)">
<!ELEMENT MSH %MSH.CONTENT;>


<!-- Message definitions -->
<!ENTITY % ACK.CONTENT "(MSH,MSA,ERR?)">
<!ELEMENT ACK %ACK.CONTENT;>
```

C. Long Example Message

Here we see a sample message in the syntax of the standard encoding rules.

```
MSH|^~\&|LAB|767543|ADT|767543|19900314130405||ADT^A04|XX3657|P|2.3.1<CR>
EVN|A01|19980327101314|19980327095000|I||19980327095000<CR>
PID|1||123456789ABCDEF|123456789ABCDEF|PATIENT^BOB^S||19590520|M||
    612345 MAIN STREET^^ANYTOWN^CA^91234||714-555-1212|
    714-555-1212||||123456789ABCDEF|||U<CR>
PD1||||WELBY<CR>
PV1|1|0||NEW|||SPOCK<CR>
```

Here we see that sample message in the syntax of the XML encoding rules.

```
<!DOCTYPE ADT_A03 SYSTEM "hl7_v231.dtd">
<ADT_A03>
<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>19900314130405</MSH.7>
        <MSH.9>
                <CM_MSG_TYPE.1>ADT</CM_MSG_TYPE.1>
                <CM_MSG_TYPE.2>A04</CM_MSG_TYPE.2>
        </MSH.9>
        <MSH.10>XX3657</MSH.10>
        <MSH.11><PT.1>P</PT.1></MSH.11>
        <MSH.12><VID.1>2.3.1</VID.1></MSH.12>
</MSH>
<EVN>
        <EVN.1>A01</EVN.1>
        <EVN.2>19980327101314</EVN.2>
        <EVN.3>19980327095000</EVN.3>
        <EVN.4>I</EVN.4>
        <EVN.6>19980327095000</EVN.6>
</EVN>
<PID>
        <PID.1>1</PID.1>
        <PID.3.LST>
                <PID.3><CX.1>123456789ABCDEF</CX.1></PID.3>
```

```
        </PID.3.LST>
        <PID.4.LST>
                <PID.4><CX.1>123456789ABCDEF</CX.1></PID.4>
        </PID.4.LST>
        <PID.5.LST>
                <PID.5>
                        <XPN.1>PATIENT</XPN.1>
                        <XPN.2>BOB</XPN.2>
                        <XPN.3>S</XPN.3>
                </PID.5>
        </PID.5.LST>
        <PID.7>19590520</PID.7>
        <PID.8>M</PID.8>
        <PID.10.LST>
                <PID.10><CE.1>6</CE.1></PID.10>
        </PID.10.LST>
        <PID.11.LST>
                <PID.11>
                        <XAD.1>12345 MAIN STREET</XAD.1>
                        <XAD.3>ANYTOWN</XAD.3>
                        <XAD.4>CA</XAD.4>
                        <XAD.5>91234</XAD.5>
                </PID.11>
        </PID.11.LST>
        <PID.13.LST>
                <PID.13><XTN.1>714-555-1212</XTN.1></PID.13>
        </PID.13.LST>
        <PID.14.LST>
                <PID.14><XTN.1>714-555-1212</XTN.1></PID.14>
        </PID.14.LST>
        <PID.18><CX.1>123456789ABCDEF</CX.1></PID.18>
        <PID.21.LST>
                <PID.21><CX.1>U</CX.1></PID.21>
        </PID.21.LST>
</PID>
<PD1>
        <PD1.4.LST>
                <PD1.4><XCN.1>WELBY</XCN.1></PD1.4>
        </PD1.4.LST>
</PD1>
<PV1>
        <PV1.1>1</PV1.1>
        <PV1.2>0</PV1.2>
        <PV1.4>NEW</PV1.4>
        <PV1.7.LST>
                <PV1.7><XCN.1>SPOCK</XCN.1></PV1.7>
        </PV1.7.LST>
</PV1>
</ADT_A03>
```

As is always the case with XML when processed with a validating processor, the extra whitespace between elements (provided to make the message easier for people to read) can be removed in actual message instances, resulting is shorter messages in situations when overall message length is a factor.

D. Translating Between Standard Encoding And XML Encoding

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

Because of several important differences between the standard encoding and this XML encoding, translations between the two encodings is not straightforward although it is not hard. The following features of both encodings need to be taken into account when performing the translations:

- in the standard encoding, optional components (or sub-components) are represented as ||; in the XML encoding, optional components are simply omitted
- in the standard encoding, *lists* and *groups* are not explicitly encoded; in the XML encoding they are explicitly encoded

V. REFERENCES

1. [CEN, 1993] 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.

2. [Clark] James Clark's Home Page (and home of James Clark's SP, XP and Xpat validating SGML and XML parsers) (http://www.jclark.com/) (James Clark's processor can be run in SGML or XML mode. Please check the documentation for more information.)

3. [Cover] Robin Cover's SGML/XML Web Page, which is a comprehensive online database containing reference information and software pertaining to the SGML and XML. (http://www.oasis-open.org/cover/sgml-xml.html)

4. [Dolin, 1997] Dolin RH, Alschuler L, Bray T, Mattison JE. SGML as a message interchange format in healthcare. JAMIA Fall Symposium Supplement 1997: 635-9. (http://www.mcis.duke.edu/standards/HL7/committees/sgml/references/amia_f97.htm)

5. [Dolin, 1998] Dolin RH, Rishel W, Biron PV, Spinosa J, Mattison JE. SGML and XML as interchange formats for HL7 messages. JAMIA Fall Symposium Supplement 1998.

6. [HL7 V3/XML] HL7 Version 3 - XML Project Document. Draft, August, 1998 (http://www.mcis.duke.edu/standards/HL7/committees/control-query/V3XML980809.ZIP)

7. [IBM] IBM's XML Developers web site (http://www.ibm.com/developer/xml/)

8. [Krueger] Krueger G. A structured approach to HL7 application development. 1996. (http://www.gkrueger.com/other/hl7/)

9. [Megginson, 1998] David Megginson. Structuring XML Documents (Charles F. Goldfarb Series). 1998. Prentice Hall Computer Books; ISBN: 0136422993.

10. [Microsoft] Microsoft's XML web site. (http://msdn.microsoft.com/xml/default.asp)

11. [Oemig] Frank Oemig's Home Page (http://www.sr-solutions.de/oemig/)

12. [Oemig, 1996] Oemig F, Dudeck J. Problems in developing a comprehensive HL7 database. AMIA Fall Symposium 1996.

13. [St.Laurent, 1999] Simon St.Laurent, Ethan Cerami. Building XML Applications. 1999. McGraw-Hill; ISBN: 0071341161.

14. [XML.com] XML.com web site (http://www.xml.com/)

15. [XML.org] XML.org web site (http://www.xml.org/)

16. [XML, 1998] Extensible Markup Language (XML) 1.0. W3C Recommendation 10-February-1998. (http://www.w3.org/TR/1998/REC-xml-19980210.html)

VI. APPENDIX - ISSUES IDENTIFIED WITH THE HL7 DATABASE

The material in this appendix is not a part of the balloted HL7 Recommendation, but is included here for informative purposes only.

In the course of building the XML DTDs described in this report, we made some assumptions about and some minor changes to the HL7 Database. We also found a couple of discrepancies between the HL7 Database and the Version 2.3.1 documents. We describe here exactly how we managed the identified issues. Additionally, everything listed here has been submitted back to HL7 for consideration.

III. ALGORITHMS
A. EXTRACTING SUBSETS OF THE NORMATIVE HL7 DATABASE
1. Messages and their segments

HL7 Table 0354's list of message structures is incomplete. The value "ACK" is missing, as are values from Chapter 4 Order Entry. The value for "ORM_O01" contains a typo in that there is a double underscore, whereas there should only be a single underscore. We added the following entries to table TableValues.

Table.ID	Version	Value
0354	2.3.1	ACK
0354	2.3.1	ORM_O01 (fix underscore)
0354	2.3.1	OMS_O01
0354	2.3.1	OMN_O01
0354	2.3.1	OMD_O01
0354	2.3.1	ORS_O02
0354	2.3.1	ORN_O02
0354	2.3.1	ORD_O02
0354	2.3.1	ORR_O02
0354	2.3.1	RDO_O01
0354	2.3.1	RRO_O02

In Table MsgStructIDSegments, one of the segment names is in lower case. We changed it to upper case:

In table MsgStructIDSegments, change this:

Msg.Struct.	Ver.	No.	Segm	repetition code	Usage	repetitional	optional
ORM_O01	2.3.1	48	ail		O	No	No

to this:

Msg.Struct.	Ver.	No.	Segm	repetition code	Usage	repetitional	optional
ORM_O01	2.3.1	48	AIL		O	No	No

III. ALGORITHMS
B. ALGORITHMS FOR XML DTD GENERATION FROM THE EXTRACTED SUBSETS
1. Messages contain segments

Application of the algorithm described in III.B.1 results in an ambiguous XML content model for message REF_I12. This comes about because of the following Abstract Message Syntax within the REF_I12 definition:

[PV1[PV2]] [PV1[PV2]]

which corresponds to an XML content model of:

(PV1,PV2?)?,(PV1,PV2?)?

With this structure, both the 1st and 2nd occurrences of "PV1" are possible. In order to fix this, we modified table MsgStructIDSegments from this:

Message_structure	lfd_nr	seg_code
REF_I12	89	[
REF_I12	90	PV1
REF_I12	91	[
REF_I12	92	PV2
REF_I12	93	]
REF_I12	94	]
REF_I12	95	[
REF_I12	96	PV1
REF_I12	97	[
REF_I12	98	PV2
REF_I12	99	]
REF_I12	100	]

to this:

Message_structure	lfd_nr	seg_code
REF_I12	89	[
REF_I12	90	PV1
REF_I12	91	[
REF_I12	92	PV2
REF_I12	93	]
REF_I12	94	[
REF_I12	95	PV1
REF_I12	96	[
REF_I12	97	PV2
REF_I12	98	]
REF_I12	99	]
REF_I12	100	]

so that the Abstract Message Syntax becomes this:

[PV1[PV2] [PV1[PV2]]]

which corresponds to an XML content model of:

(PV1,PV2?,(PV1,PV2?)?)?

which is unambiguous (and requires no change in actual message instances).

III. ALGORITHMS
B. ALGORITHMS FOR XML DTD GENERATION FROM THE EXTRACTED SUBSETS
4. Data types contain data type components

HL7 Version 2.3.1, Chapter 2 shows the TQ data type as a composite:

Data Type Category/ Data type	Data Type Name	HL7 Section Reference	Notes/Format
TQ	Timing/quantity	2.8.4.3	For timing/quantity specifications for orders, see Chapter 4, Section 4.4. <quantity (CQ)> ^ <interval (*)> ^ <duration (*)> ^ <start date/time (TS)> ^ <end date/time (TS)> ^ <priority (ST)> ^ <condition (ST)> ^ <text (TX)> ^ <conjunction (ID)> ^ <order sequencing (*)> ^ <occurrence duration (CE)> ^ <total occurrences (NM)>

while the HL7 Database represents the TQ data type as a primitive. We are noting this finding, and took no action on it. As a result, the XML representation treats TQ as a primitive data type.

III. ALGORITHMS
B. ALGORITHMS FOR XML DTD GENERATION FROM THE EXTRACTED SUBSETS
4. Data types contain data type components

HL7 Version 2.3.1, Chapter 2 shows MSH.9 (Message Type) to be a CM data type, whose first component is an ID data type:

Message type (CM): <message type (ID)> ^ <trigger event (ID)> ^ <message structure (ID)>
Table Components in the HL7 Database represents the first component of MSH.9 as a "??" data type. We changed "??" to "ID" in the HL7 Database.

In table Components, changed this:

Comp.	Vers.	Description	Table-ID	Data Type
223	2.3.1	message type	0076	??

to this:

Comp.	Vers.	Description	Table-ID	Data Type
223	2.3.1	message type	0076	ID