HL7 Version 3-METL Description

January 26, 1999

DRAFT

Contents

Figures

Introduction

The HL7 Message Element Type Language (METL) is a technology-neutral way of describing a version 3 message. Its capabilities overlap that of the HMD. Indeed, it was conceived as an interim measure pending the availability of HMD software. It is quite likely that it will cease to exist when the HMD software is available.

In the meantime, however, it has been used to express message content for the HL7 Version 3 Messaging prototype to be demonstrated at the HIMSS trade show in February, 1999. A compiler has been written that accepts a message definition in METL and produces an XML DTD along with a "skeleton" sample XML message instance. The skeleton has instances of all the elements that the DTD will generate, given the message type the root. However the data that is present in each message is either "XXXMANDATORY" or "XXXOPTIONAL". It is frequently easier to build meaningful examples by editing the skeleton then by starting from scratch.

This is a first draft of the document. All caveats known to Man or conceivable by lawyers apply.

Introductions to Types and Components

Programming language-ish examples:

```
Type Point Contains
       X of type Real
       Y of Type Real

Type Line Contains
       Start of Type Point
       End of Type Point

Variable A of type Line
```

`A.Start` is a Point

`A.Start.X` is a Real.

A is a variable.

`X, Y, Start,` and `End` are *not variables*. They are **Components.** No memory is allocated anywhere for an X until the declaration for variable A.

In general, types can be atomic or contain Components, each of which has a name and a type.

We must be careful with this analogy, because *HL7 doesn't have variables.* HL7 has *types* and *instances.*

HMDs define **message types.** However, HMDs contain a lot more than simply the message type, and this additional information is also a normative part of the standard. The additional information includes semantics and business rules. More on this later.

Context

Figure 1, reproduced from MDF-98, helps to place this discussion in context. (As discussed below, the caption for the gray area will change from "Defining a Message Structure" to "Defining a Message Type" in MDF-99). XML is an ITS. There may be other ITSs. One of the primary benefits of XML is that the small boxes at the bottom labeled HL7 Message Creation and HL7 Message Parsing can rely on ubiquitous standard programs when the message instance is an XML document.

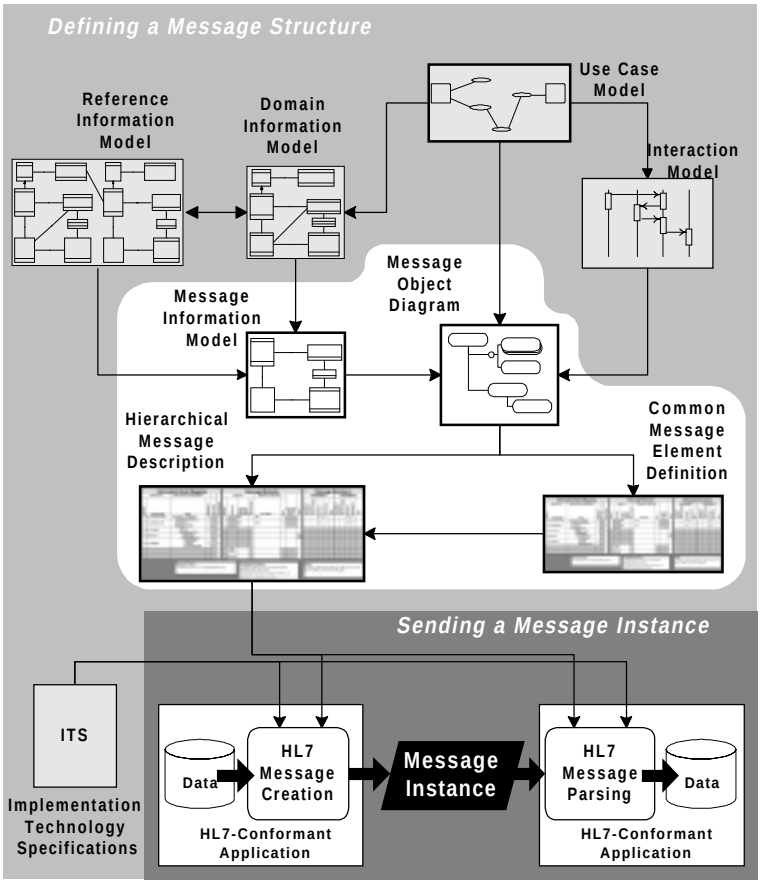

Figure 1. Version 3 process.

Definitions

An italicized term in the definitions is the subject of another definition. This section does not repeat the definition of terms from MDF-98 that have not changed, unless important to the exposition.

choice message element type	A *composite message element type* for which only one of the components will be sent in an instance.
common message element type	Certain message element types are defined in Common Message Element Definitions. These are defined separately from their use as the type of a Component in their HMD.
composite message element type	A message element type that contains other message elements (components). Each component message element has a *name* and a *type*. Each component of an element must have a different name, although many may be of the same type.
Hierarchical Message Definition (HMD)	A metaobject that defines message types. It also defines the linkages of the message types to Interactions and the linkages of certain message element types to attributes of the Reference Information Model.
instance	An utterance that conforms to a *type*. See *message instance, message element instance,* etc.
MDF-98	The Message Development Framework for Version 3, published by HL7 in January, 1998.

message	A message element that is the unit of information interchange among information systems conforming to HL7 Application Roles.
message message element type	A particular composite message element type which expresses an entire message.
message element	The basic unit of structure of a version 3 message. Message elements can contain other message elements. Message elements may contain other message elements.
message element instance	An actual set of data that is part of an actual message.
Message Element Instance Model	A graph of objects that represent a message instance. Each node represents a message element instance.
message element type	A specification for the values that a *message element* can take on in its instances.
Message Element Type Model	A graph of objects that represent a message type. Each node represents a message element type.
message instance	An actual message element instance corresponding to a *message type*.
message type	The type of a message. A message type is always a single message element type, although the type will contain many components.
pointy-bracket syntax	A syntax that generates a subset of all XML productions. Instances in the subset don't contain any meta-information.
primitive message element type	A *message element type* that does not contain other message elements
public message element type	When defining a message element in an HMD, the type that is assigned to a component may be defined within the HMD or it may be defined externally. A message element type that is defined externally to an HMD is an external message element type. Public message element types include *common message element types* and *primitive message element types.*
type	A description that describes the formulation or allowable values for numerous instances, which are not necessarily identical. For example the type "integer" describes a number of instances including 1, 2, 1.235 x 20^{27}, and –379. See *message type*, *message element type*, etc.

Message Element Type/Message Element Instance Model

Conceptually, the dark gray portion of Figure 1 could be replaced with Figure 2. The HL7 Instance Object Model is a collection of objects (instances) drawn here as a graph. The nodes are instances of message elements; each is an instance of a message element type as defined in the HMD. The solid edges represent containment and the broken-line edges represent pointers.

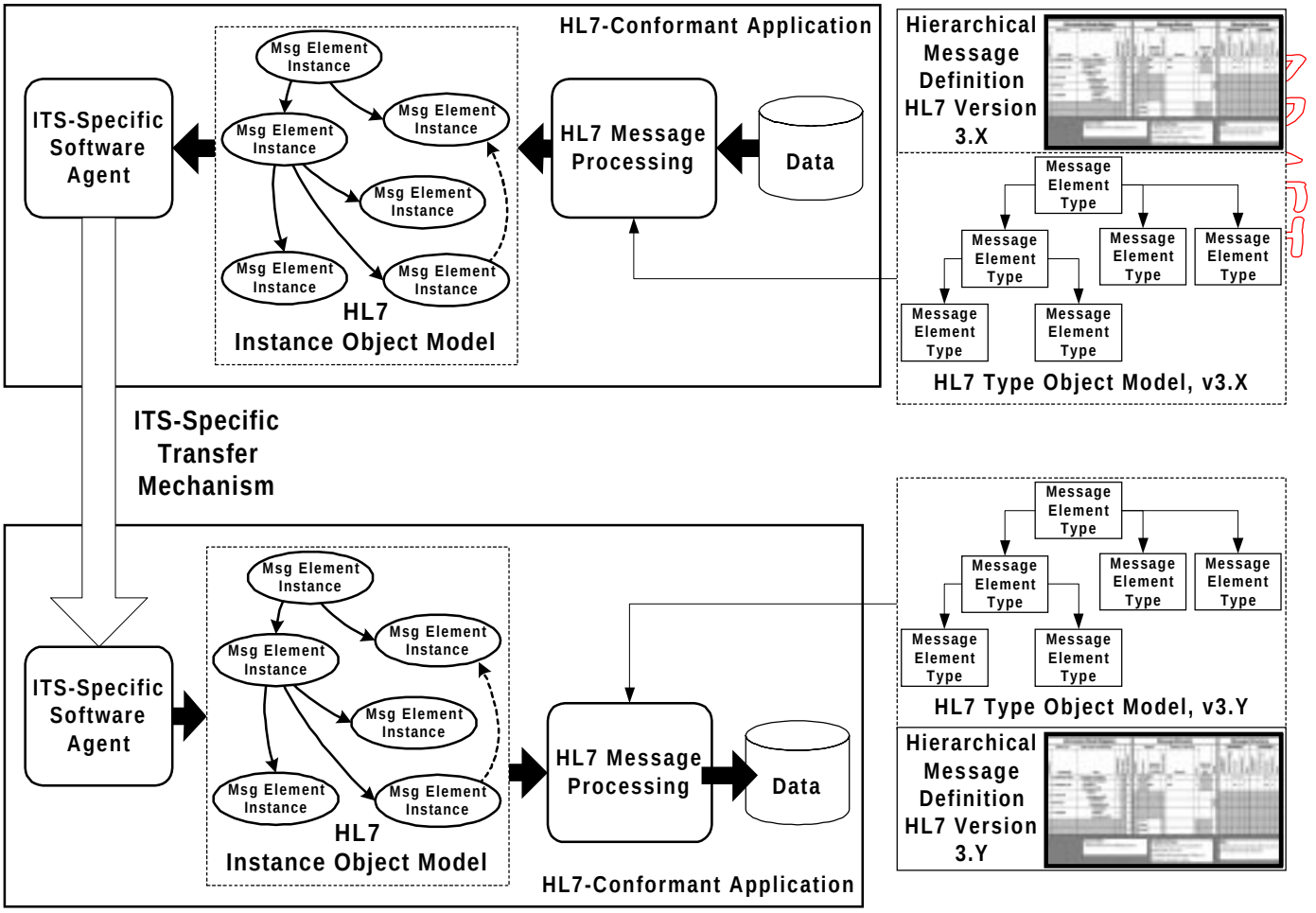

Figure 2. Message Element Type and Instance Object Models

The word "conceptually" is an important part of the preceding paragraph. HL7 has traditionally not seen fit to standardize the design of software. Current ideas conformance testing in version 3 envision the entire "HL7-Conformant Application" as a black box.

HL7 has, however, written recommendations for object models of messages. That is what SIGOBT had done and is doing.

We also conceptually think of the portion of the HMD that represents type information as the *message element type model*. We can further note that the type model that was used to create the message is that of the sender. The message element instance model that is built in the receiver's space may not correspond exactly to its message element type model. These conceptual definitions allow us to define the matching process as an algorithm that simultaneously steps between nodes of the Instance Object Model and makes matching steps among nodes of the Type Object Model.

(Personal note: I visualize this as a "tree dance". The tree dance is two tree-walks occurring together, with one foot walking the nodes of the type model and the other walking the nodes of the instance model.)

An important principal is that the logic to do the tree dance, and to process the data that is retrieved from the instance tree, is the same no matter what ITS was used to create the instance tree. In other words, the services provided by the message element instance nodes are defined without regard to the ITS.

One can ponder the relationship between the Message Element Instance object and a node in the XML Document Object Model. Our chosen use of XML permits the ITS-specific software agent to implement the Message Element

Instance object as a wrapper around *XMLNode* or some other important interface in the XML document object model.

The Message Element Type Model

Figure 3 is the message element model.

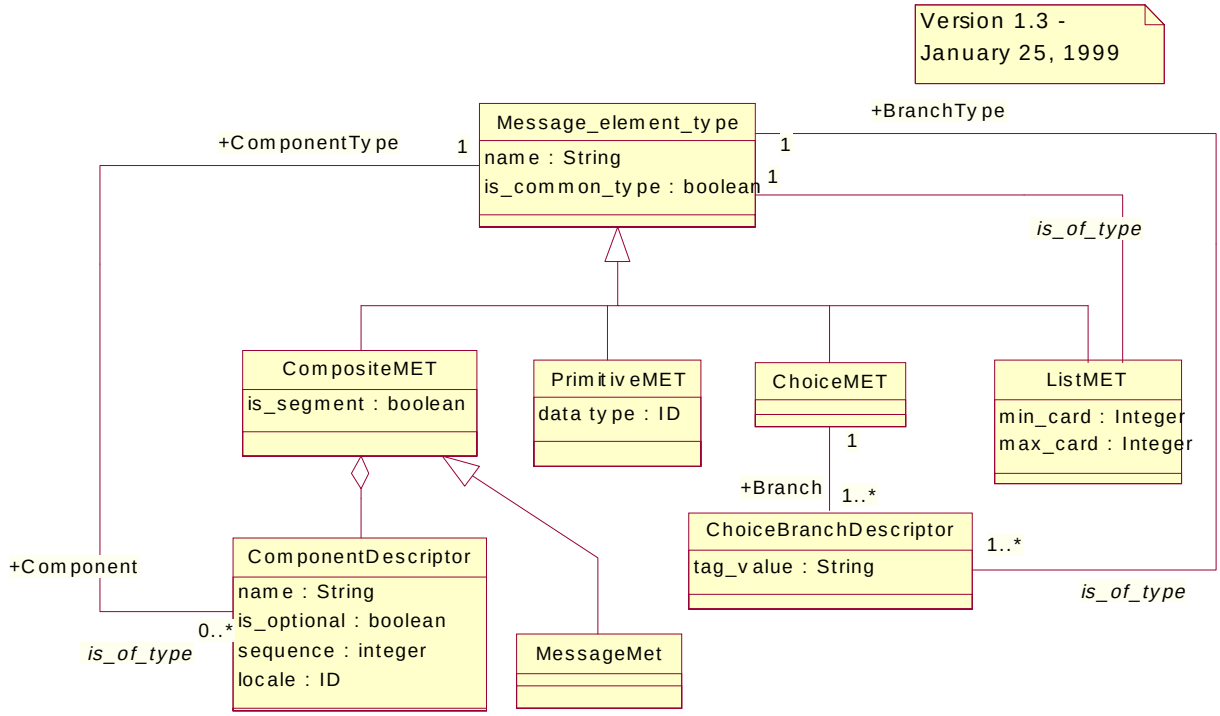

Figure 3. Message Element Type Metamodel.

The Message Element Type Language

The message element type language (METL) is an experiment at expressing HL7 version 3 message definitions using a linear language rather than the HMD. At this point the METL does not begin to express all the notions that could be expressed in the HMD. Some of the other HMD notions have been expressed as comments in the METL source for the purpose of creating the demo.

Figure 4 is a toy example of a message definition in METL. It purports to be a query for patient information. They system issuing the query must send primary_name_type_cd and primary_prsnm; in addition it may send a Stakeholder_id or a list of phone numbers (phone_number_list).

Remember from the Message Element Type model that all message element types are one of these metatypes.

- **composite**
- **list**
- **choice**
- **primitive**.

There should be a fifth metaype: **message**.

Each statement of the METL defines an instance of a type. Element Type metamodel, previous described in this document.

The following is true for every message element type:

- it has a **long name**
- it has a **short name** (an abbreviation of the long name that is used in building the XML)
- it *defines* a **type.** The type may be used in other message element type definitions; if so it is used by referring to the short name.
- it may be declared to be **public**. This means that it is intended to be shared broadly among different message definitions.

There are statements in the METL to define message elements of each of the metatypes, plus a few others.

The 'MESSAGE' Message Element Type

Line 4 in the example defines a message (in this case the only message defined in this source file.)

```
MESSAGE TYPE QRYNam_Patient_Reg   [QRgNamv3P00]  CONTAINS {
    Message_header         [MSGH]       MANDATORY OF TYPE MSGH
    QryPatient_person_Nam  [QPTNAM]     MANDATORY OF TYPE QPTNAM
}
```

The long name is QRYNam_Patient_Reg, and the short name is QRgNamv3P00. As with any type definition, the short name is the name of the type it defines.

The message comprises two components. Each has a name, a short name, and is "of a type". Note that in this example the short name and the type name are the same. This is very common in HL7. There are not likely to be two different components of a message, both of type MSGH (message header) so it doesn't make much sense to waste brainpower trying to find an original name. As we will soon see, this is a common occurrence, but it does not happen all the time.

Figure 4. Toy example of MTL.

```
# Query by primary_prsnm message QRgNamv3P00

MESSAGE TYPE QRYNam_Patient_Reg   [QRgNamv3P00]  CONTAINS {
    Message_header          [MSGH]       MANDATORY OF TYPE MSGH
    QryPatient_person_Nam  [QPTNAM]     MANDATORY OF TYPE QPTNAM
    }
    COMPOSITE TYPE QryPatient_person_Nam  [QPTNAM]  CONTAINS {
        primary_name_type_cd [primrNamType] MANDATORY OF TYPE CE
        primary_prsnm        [primrPrsnm]   MANDATORY OF TYPE PN
        Stakeholder_id       [StkID]        OPTIONAL  OF TYPE StkID
        phone_number_list    [PhonNmbr_L]   OPTIONAL  OF TYPE PhonNmbr_L
    }
        COMPOSITE TYPE Stakeholder_ID [StkID] CONTAINS {
            id                   [id]     MANDATORY OF TYPE ST
            identifier_type_cd   [idType] MANDATORY OF TYPE ID # Values for demo:
                                                               #      UPIN
                                                               #      MRN
        }
#-------------------------------------------------------------------------
#                       HL7 PUBLIC TYPE DEFINITIONS
#-------------------------------------------------------------------------
PUBLIC COMPOSITE TYPE Message_header   [MSGH]  CONTAINS {
    sending_application   [sndApp]     OPTIONAL  OF TYPE ID
    receiving_application [rcvgApp]    OPTIONAL  OF TYPE ID
    date_time_message     [msgDt]      OPTIONAL  OF TYPE DTM
    message_type          [msgTyp]     MANDATORY OF TYPE MSGT
}
    PUBLIC COMPOSITE TYPE   Message_type    [MSGT] CONTAINS {
        message_id       [msgID]       MANDATORY OF TYPE ID
        interaction_id   [intrId]      OPTIONAL  OF TYPE ID
    }
PUBLIC COMPOSITE TYPE Person_name [PN] CONTAINS {
    family_name            [fmn] MANDATORY OF TYPE ST
    given_name             [gvn] OPTIONAL  OF TYPE ST
    middle_initial_or_name [mdn] OPTIONAL  OF TYPE ST
    suffix                 [sfx] OPTIONAL  OF TYPE ST # e.g., JR or III
}
PUBLIC COMPOSITE TYPE   Coded_element [CE]  CONTAINS {
    identifier              [cd]    MANDATORY OF TYPE ST
    identifier_text         [tx]    OPTIONAL  OF TYPE ST
    name_of_coding_system   [cs]    MANDATORY OF TYPE ST
    alt_identifier          [acd]   OPTIONAL  OF TYPE ST
    alt_identifier_text     [atx]   OPTIONAL  OF TYPE ST
    alt_name_coding_system  [acs]   OPTIONAL  OF TYPE ST
}
PUBLIC LIST TYPE Patient_Phone_Number_List [PhonNmbr_L] INCLUDES 1..N
         OF TYPE XTN_C

    CHOICE TYPE Electronic_contact_address [XTN_C] SELECTS
        {
        E: email_address     [emailAddr]     OF TYPE ST
        T: Extended_telecomm_id [ExtndTelId] OF TYPE XTN
        }
PUBLIC COMPOSITE TYPE Extended_telecomm_address [XTN] CONTAINS {
    telecom_use_code [tlcmnUse]           MANDATORY OF TYPE CE # Table 201
    telecomm_equipment_type [tlcmnEqpTyp] MANDATORY OF TYPE CE # Table 202
    country_code     [cntryCode]    OPTIONAL  OF TYPE NM
    area_city_code   [areaCityCode] OPTIONAL  OF TYPE NM
    phone_number     [phonNmbr]     MANDATORY OF TYPE NM
    extension        [xtnsn]        OPTIONAL  OF TYPE NM
    any_text         [anytxt]       OPTIONAL  OF TYPE TX
}
PUBLIC PRIMITIVE TYPE text_for_humans_to_read [TX]
PUBLIC PRIMITIVE TYPE string                  [ST]
PUBLIC PRIMITIVE TYPE ID                      [ID]
PUBLIC PRIMITIVE TYPE numeric                 [NM]
PUBLIC PRIMITIVE TYPE date_time               [DTM]
```

In this example, the keyword MANDATORY appears in for each component. That means that it *must* appear in each message occurrence. It is also possible to sat OPTIONAL.

The COMPOSITE Message Element Type

The first component of the message is Message_header. It has the short name MSGH and it is also of type MSGH. Let's drill down and look at the subcomponents of this component. To do that, we find a message element type definition whose short name is MSGH. (Hint: look at line 23.)

```
PUBLIC COMPOSITE TYPE Message_header   [MSGH]  CONTAINS {
    sending_application   [sndApp]     OPTIONAL  OF TYPE ID
    receiving_application [rcvgApp]    OPTIONAL  OF TYPE ID
    date_time_message     [msgDt]      OPTIONAL  OF TYPE DTM
    message_type          [msgTyp]     MANDATORY OF TYPE MSGT
}
```

It is another composite, which itself has four components. The first component of MSGH is sending_application short name sndApp which is of type ID.

The PRIMITIVE Message Element Type

Let's drill down again, by finding the definition of the type ID. (Line 66).

```
PUBLIC PRIMITIVE TYPE ID [ID]
```

In this example the long name and the short name are the same: ID.

Here we find an element type that is of a different metatype: primitive. METL doesn't tell us anything about the syntax of an ID, except that there is no more type drilldown.

This drilldown is building a hierarchy of components within components. It is common to represent a hierarchy in outline form. Here's what we have so far (just using short names)

```
QRgNamv3P00 contains {
        MSGH OF TYPE MSGH, which contains (
                sndApp OF TYPE ID
```

If we finish the components of MSGH, it looks like this

```
QRgNamv3P00 contains {
        MSGH OF TYPE MSGH, which contains (
                sndApp  OF TYPE ID
                rcvgApp OF TYPE ID
                msgDt   OF TYPE DTM
                msgTyp  OF TYPE MSGT contains {
                        msgID OF TYPE ID
                        intrId OF TYPE ID
                }
        }
        QPTNAM etc,
}
```

The LIST Message Element Type

If we drill down into the QPTNAM message element we find a component named Patient_Phone_Number_List [PhonNmbr_L] of type PhonNmbr_L. The definition of the PhonNmbr_L type (line 47) introduces a new metatype: a list.

```
PUBLIC LIST TYPE Patient_Phone_Number_List [PhonNmbr_L] INCLUDES 1..N
          OF TYPE XTN_C
```

Any element of type PhonNmbr_L will contain a repeating set of elements of type XTN_C, which will have at least one entry.

The MET Language allows the cardinality to be 0..N, 1..N, or expressions like 0..5 or 4..7. If the lower limit is other than 0 or 1, or if an integer is used for the upper limit, instead of N, this information is lost when the METL is

translated into a DTD. The DTD only has the ability to express (0..N) or (1..N). The other values are rare. Where they might exist, they constitute business rules, which might be expressed in an XML attribute.

The CHOICE Message Element Type

Drilling down further to the definition of XTN_C (line 50) we find an example of the last metatype: choice.

```
CHOICE TYPE Electronic_contact_address [XTN_C] SELECTS
    {
    E: email_address        [emailAddr]  OF TYPE ST
    T: Extended_telecomm_id [ExtndTelId] OF TYPE XTN
    }
```

This type definition says that some message instances will have an element of emailAddr of type ST, whereas others will have an element of named ExtndTelId of type XTN. The letters E and T are tag values. Although these are included in the XML representations of messages, they serve no real purpose. They will be required if we also have an ER7 representation.

A CHOICE message element type is similar to a composite, in that it lists one or more components that may occur. However, unlike a composite, only one of the listed components may occur. The keyword SELECTS is used instead of CONTAINS to emphasize the distinction.

The choice message element type is used systematically to describe the information structure associated with specialized instances in a message. It can also serve *ad hoc* to solve specific message design problems. For example, the current definition of Clinical_observation includes many attributes that are only sensible for specific values of value_type_cd. The responsible committee, however, has not chosen to model this with specializations of Clinical_observation. The message designer has the option of making these optional (even though they may really be required for specific values of value_type_cd) or constructing an ad hoc choice to correctly require certain attributes based on the value type.

METL Syntax

Lexical notes: any white space is allowed between any syntax elements. This includes newlines. Newlines are required, however, where they appear in the syntax.

"<include *filepath*>" appearing in column 1 (with the angle brackets) terminates processing of the current file and continues with the file at filepath.

Figure 5. MET Language Syntax.

```
<ROOT> ::=          (<MsgMet> | <CompMet> | <PrimMet> | <ListMet> | <ChoiceMet> )+
                     endfile

<MsgMet> ::=        ('MESSAGE' 'TYPE' <longName> '[' <shortName> ']' 'CONTAINS' '{'
                       ( <CompMetComp> newline )+ '}' newline

<CompMet> ::=       ['PUBLIC'] 'COMPOSITE' 'TYPE' <longName> '[' <shortName> ']'
                    'CONTAINS' '{' ( <CompMetComp> )+ '}' newline

<CompMetComp>::=    <longName> '[' <shortName> ']' (MANDATORY | OPTIONAL)
                    OF 'TYPE' <shortName> newline

<PrimMet> ::=       ['PUBLIC'] 'PRIMITIVE' 'TYPE' <longName> '[' <shortName> ']' newline

<ListMet> ::=       ['PUBLIC'] LIST  'TYPE' <longName> '[' <shortName> ']' 'INCLUDES'
                    <lowLimit> '..' <highLimit> OF 'TYPE' <shortName> newline

<ChoiceMet>::=      ['PUBLIC'] 'CHOICE' 'TYPE' <longName> '[' <shortName> ']' SELECTS
                     '{'  ( <ChoiceMetComp> )+ '}' newline

<ChoiceMetComp>::= <tagValue> ':' <longName> '[' <shortName> ']'
                    OF 'TYPE' <shortName> newline

<longName> ::=      pattern "[A-Za-z0-9_]+"
<shortName>::=      pattern "[A-Za-z0-9_]+"
<lowLimit> ::=      pattern "[0-9]+"
<highLimit>::=      ( 'N' |  pattern "[0-9]+")
<tagValue> ::=      pattern "[A-Za-z]"
```


Relationship to the RIM

Currently, the METL does not directly specify the relationship between message element types and RIM metaobjects. Many element types do not have a direct counterpart in the RIM. These include

- element types that implement data types, and
- element types that implement information structures associated with the HL7 protocol rather than with the functional data. (In this example, the MSGH message element is an example of a message element that is required for HL7, but does is not represented in the RIM.)

Where message element types do have a relationship to the RIM this is usually shown by using long names for types or components that correspond to class names, attributes or associations. Examples in the toy message type include Stakeholder_ID at lines 14-19 and the component names within QryPatient_person_Nam at lines 8-13.

An Ersatz Message Object Diagram

As part of the prototype we have been trying to find a notation, comparable to the Message Object Diagram, that can be represented in plain text. The following example, taken from the METL that was designed to send an XML document within a version 3 message, is the best we have found. The right column consists of "objects" in the sense of the message object diagram … specializations of classes based on distinctive semantics. The indentation of this column indicates the hierarchical relationship of the message elements.

The left column presents the rationale for allowing the corresponding object to appear. It may be "root", "substitute CMED", "specialize", "generalize", or the name of an association.

Special characters are used in the notation to denote recursion, to indicate the components of a choice and to describe the cardinality of portions of the hierarchy.

location	symbol	interpretation
left side of right column	no symbol	(1,1) cardinality
left side of right column	?	(0,1) cardinality
left side of right column	+	(1,n) cardinality
left side of right column	*	(0,n) cardinality
left side of right column	-	choice component
left side of left column	**	this is a recursion

Figure 6. Ersatz MOD notation.

```
root                    Patient
Substitute CMED            Patient_person
is_target_of               Target_participation
is_target_of                  +Service_intent_or_order
is_an_instance_of                ?Master_service
has_as_active_participant        +Active_participation
has_as_participant                  Stakeholder
specialize                             Person
substitute CMED                           Person_ID
is_fulfilled_by                  +Service_event
is_documented_by                    Clinical_document_header
**has_as_a_parent_document             ?Clinical_document_header
has_been_originated_by                 ?Originator
substitute CMED                           Provider_Stakeholder
is_related_to                             *Authentication
is_related_to                                Healthcare_document_authenticator
substitute CMED                                 Provider_Stakeholder
is_transcribed_by                         ?Transcriptionist
is_identified_as                             Person
is_performed_at                     Master_patient_service_location
**is_included_in                       ?Master_patient_service_location
is_assigned_to                      ?Patient_encounter
has_as_participant                     +Encounter_practitioner
is_participant_for                        Individual_healthcare_practitioner
has_as_active_participant        Active_participation
has_as_participant                  Stakeholder
specialize                             Person
substitute CMED                           Person_ID
```

Mapping the METL to the XML DTD

In the message style chosen for the HIMSS prototype, all message data is sent as #PCDATA. That is to say, it appears between a start-tag and an end-tag. (An alternative strategy would be to send some or all data as the value of attributes of elements.) Each message element that is of a primitive type (and can therefore contain data) has the XML content model (#PCDATA).

The structure contained by composite, list, or choice message element types is implemented as a set of XML element with other content models. For example, the DTD comparing to the toy message might have included the following element definitions. (These are actually too simple, as will be explained below.) The XML comments included in the contain the line numbers in Figure 4 of the corresponding METL.

```
<!--  4-->      <!ELEMENT QRgNamv3P00  (MSGH,  QPTNAM)>
<!-- 23-->      <!ELEMENT MSGH         (sndApp?, rcvgApp?, msgDt?, msgTyp)>
<!--  8-->      <!ELEMENT QPTNAM       (primrNamType, primrPrsnm, StkID?,
                                        PhonNmbr_L?)>
<!-- 66-->      <!ELEMENT sndApp       (#PCDATA)>
<!-- 66-->      <!ELEMENT rcvgApp      (#PCDATA)>
<!-- 68-->      <!ELEMENT msgDt        (#PCDATA)>
<!-- 29-->      <!ELEMENT msgTyp       (msgID, intrId?)>

<!-- 66-->      <!ELEMENT msgID        (#PCDATA)>
<!-- 66-->      <!ELEMENT intrId       (#PCDATA)>
```

Name Collisions and the Indiana Dot Notation

Furthermore, there is an issue with names. Currently all element names in a DTD are global, but attribute names in the RIM are implicitly qualified by class name and systematically repeated to indicate specific design patterns. In order to avoid XML ELEMENT name collisions, the DTD compiler creates names by joining the name of an element of a composite type with the name of its component. Compare the example below to lines 4-13 of the toy example. The message element QPTNAM is a component of the MESSAGE message element QRgNamv3P00, so it appears as QRgNamv3P00.QPTNAM. QPTNAM is also the name of a type that has, among others, a component named primrPrsnm. The type QPTNAM defines the content model of QRgNamv3P00.QPTNAM. The component names of the type are prepended with the type name. For example primrPrsnm becomes QPTNAM.primrNamType. See the partial example below

```
<!ELEMENT QRgNamv3P00         (QRgNamv3P00.MSGH, QRgNamv3P00.QPTNAM)>
<!ELEMENT QRgNamv3P00.QPTNAM (QPTNAM.primrNamType, QPTNAM.primrPrsnm,
                               QPTNAM.StkID?, QPTNAM.PhonNmbr_L?)>
```

For historical reasons, this convention is called the Indiana Dot Notation. (Mark Tucker and Gunther thought it up.)

Currently, message element types that are declared PUBLIC receive different treatment with respect to the Indiana Dot Notation. To demonstrate this we drill down further in QPTNAM:

```
<!ELEMENT QRgNamv3P00          (QRgNamv3P00.MSGH, QRgNamv3P00.QPTNAM)>
<!ELEMENT QRgNamv3P00.QPTNAM  (QPTNAM.primrNamType, QPTNAM.primrPrsnm,
                                QPTNAM.StkID?, QPTNAM.PhonNmbr_L?)>

<!ELEMENT QPTNAM.primrNamType (cd, tx?, cs, acd?, atx?, acs?)>

<!ELEMENT cd                  (#PCDATA)>
<!ELEMENT tx                  (#PCDATA)>
<!ELEMENT cs                  (#PCDATA)>
<!ELEMENT acd                 (#PCDATA)>
<!ELEMENT atx                 (#PCDATA)>
<!ELEMENT acs                 (#PCDATA)>

<!ELEMENT QPTNAM.primrPrsnm   (fmn, gvn?, mdn?, sfx?)>
<!ELEMENT fmn                 (#PCDATA)>
<!ELEMENT gvn                 (#PCDATA)>
```

```
<!ELEMENT mdn                (#PCDATA)>
<!ELEMENT sfx                (#PCDATA)>
```


Because the message element type associated with the component QPTNAM.primrNamType is CE, the element has six subcomponents (see lines 39-46 in the METL). Each of these subcomponents must have its own XML ELEMENT definition.

If the Indiana dot notation had been used, the subcomponent names would have been qualified.

```
<!ELEMENT QPTNAM.primrNamType (CE.cd, CE.tx?, CE.cs, CE.acd?, CE.atx?,
                               CE.acs?)>

<!ELEMENT CE.cd              (#PCDATA)>
<!ELEMENT CE.tx              (#PCDATA)>
<!ELEMENT CE.cs              (#PCDATA)>
<!ELEMENT CE.acd             (#PCDATA)>
<!ELEMENT CE.atx             (#PCDATA)>
<!ELEMENT CE.acs             (#PCDATA)>
```

In a typical message, many elements will be of type CE. Skipping the Indiana dot notation was conceived as a way to reduce the size of message instances.

In the current HIMSS demo I used the PUBLIC declaration quite frequently. In retrospect, I would have done it far less, perhaps not at all.

Representing 'Message' Message Element Types

Each 'message' message element type is represented as an XML element which has a name that is the same as the short name of the message element type. The content model for the XML element is the components of the message element type. The keyword OPTIONAL is represented by a question mark in the content model.

Each of the components in the content model is then generated as a separate element.

This somewhat contrived METL …

```
MESSAGE TYPE QRYNam_Patient_Reg   [QRgNamv3P00]  CONTAINS {
    Message_header          [MSGH]       MANDATORY OF TYPE MSGH
    QryPatient_person_Nam   [QPTNAM]     OPIONAL   OF TYPE QPTNAM
    }
```

Would generate this DTD element.

```
<!ELEMENT QRgNamv3P00     (QRgNamv3P00.MSGH, QRgNamv3P00.QPTNAM?)>
```

Each ‘message’ message element type will be the root of a separate skeleton XML example message in the compiler output. However a single DTD will contain the elements for all the messages.

Representing Composite Message Element Types

When a component or list item is generated that is of a composite message element type, the XML element is very similar to that generated for the ‘message’ message element type.

The following message element type will generate an element in the DTD, as long as at least one element is generated that has a component of type QPTNAM.

```
COMPOSITE TYPE QryPatient_person_Nam  [QPTNAM]  CONTAINS {
        primary_name_type_cd [primrNamType] MANDATORY OF TYPE CE
        primary_prsnm        [primrPrsnm]   MANDATORY OF TYPE PN
        Stakeholder_id       [StkID]        OPTIONAL  OF TYPE StkID
        phone_number_list    [PhonNmbr_L]   OPTIONAL  OF TYPE PhonNmbr_L
    }
```

The corresponding element would be.

```
<!ELEMENT QRgNamv3P00.QPTNAM (QPTNAM.primrNamType, QPTNAM.primrPrsnm,
                              QPTNAM.StkID?, QPTNAM.PhonNmbr_L?)>
```

Each of the components would also be generated as elements.

Representing List Message Element Types

When a component or list item is generated that is of a list message element type, the compiler creates a special XML elementt, which has a content model that is the consists of a single element with the name <short name of message element type>.item. A '*' or '+' is used into the content model according to the declaration in the METL.

For example:

```
PUBLIC LIST TYPE Patient_Phone_Number_List [PhonNmbr_L] INCLUDES 1..N
        OF TYPE XTN_C
```

would generate:

```
<!ELEMENT QPTNAM.PhonNmbr_L (PhonNmbr_L.item+)>
```

An element is then generated for the "<short name of message element type>.item" entry, according to the type specified in the OF TYPE clause of the METL.

Representing Choice Message Element Types

A choice is generated in the same way as a composite, except that "|" is used to separate the entries in the content model. For example, this METL:

```
CHOICE TYPE Electronic_contact_address [XTN_C] SELECTS
    {
    E: email_address        [emailAddr]  OF TYPE ST
    T: Extended_telecomm_id [ExtndTelId] OF TYPE XTN
    }
```

would generate this XML element:

```
<!ELEMENT PhonNmbr_L.item (XTN_C.emailAddr| XTN_C.ExtndTelId)>
```

Each of the components is then generated. When the components are generated, the tag letters are generated as attributes of the component elements, as will be described below.

Representing Primitive Message Element Types

When a component is generated that is of a primitive data type, the element that is generated for it always has the content model:

For example, if a component is generate of type PN,

```
PUBLIC COMPOSITE TYPE Person_name [PN] CONTAINS {
    family_name            [fmn] MANDATORY OF TYPE ST
    given_name             [gvn] OPTIONAL  OF TYPE ST
    middle_initial_or_name [mdn] OPTIONAL  OF TYPE ST
    suffix                 [sfx] OPTIONAL  OF TYPE ST # e.g., JR or III
}
PUBLIC PRIMITIVE TYPE string                  [ST]
```

After the component model for the composite is generated, the following elements would be added to the DTD.

```
<!ELEMENT fmn             (#PCDATA)>
<!ELEMENT gvn             (#PCDATA)>
<!ELEMENT mdn             (#PCDATA)>
<!ELEMENT sfx             (#PCDATA)>
```


Use of XML Attributes

The current approach uses attributes in three ways:

- The **T** attribute conveys the data type associated with an XML ELEMENT
- The **HL7_name** attribute is a fixed attribute that conveys the long name associated with the element, qualified using the Indiana dot notation. Because it is a "fixed" attribute, it does not actually appear in the message instance. But a parser that uses the DTD will generate it in the parsed output data as if it had been sent.
- The **Choice** attribute is used to convey thetag associated with a choice message element type.

Here are two examples:

```
<!ATTLIST QPTNAM.primrNamType
HL7_name  CDATA   #FIXED "QryPatient_person_Nam.primary_name_type_cd"
          T       CDATA  "CE">

<!ATTLIST XTN_C.ExtndTelId
HL7_name  CDATA   #FIXED "Electronic_contact_address.Extended_telecomm_id"
          T       CDATA   "XTN"
          Choice  CDATA   "T">
```

A Naming Anomaly

Arguably, an XML element definition defines a type. The element so defined will be used in many message instances where the contents will vary. It is surprising, then, that the names of XML elements do not correspond with Message Element Types. Many XML element names are not the names of message element types: sndApp and rcvgApp, above, are examples of this. Many message element types are not the names of XML elements. CE, XTN, and ST are examples.

Most XML elements names are actually component names in METL. In fact, there are only three ways that the name of a message element type becomes the name of an XML element:

- it is the name of a 'message' message element type
- it is the name of a list message element type
- it happens to have the same name as a component that is in the message.

Because of the Indiana dot notation, however, message element names frequently appear as the prefix for XML element names.

For me, this was the hardest thing to grasp in the entire process: **except for the message itself, it is only component names that become the names of message elements.**

Parameter Entities

The primary reflection of a message element type in a DTD is in the content model of an element. Whenever more than one component in a message has the same type, each component will be generated with the same content model. The DTD compiler generates a parameter entity for such types, and invokes it wherever it is used. The entity name is DT_<short name of message element that defines the data type>.

For example, there are several elements in the toy message of type CE.

```
<!ENTITY % DT_CE          "cd, tx?, cs, acd?, atx?, acs?">
<!ELEMENT tlcmnUse        (%DT_CE;)>
<!ELEMENT tlcmnEqpTyp     (%DT_CE;)>
```

Full DTD and XML Instance Example for the Toy Message

Figure 7. DTD for the Toy Message

```
<!--============================================================================
     HL7 Version 3 Prototype Message:  QRYNam_Patient_Reg [QRgNamv3P00]
    ============================================================================-->

<!ENTITY % DT_CE          "cd, tx?, cs, acd?, atx?, acs?">

<!ELEMENT QRgNamv3P00     (QRgNamv3P00.MSGH, QRgNamv3P00.QPTNAM)>

<!ELEMENT QRgNamv3P00.MSGH (sndApp?, rcvgApp?, msgDt?, msgTyp)>

<!ELEMENT QRgNamv3P00.QPTNAM (QPTNAM.primrNamType, QPTNAM.primrPrsnm,
                              QPTNAM.StkID?, QPTNAM.PhonNmbr_L?)>

<!ELEMENT QPTNAM.primrNamType (%DT_CE;)>

<!ELEMENT QPTNAM.primrPrsnm (fmn, gvn?, mdn?, sfx?)>

<!ELEMENT QPTNAM.StkID    (StkID.id, StkID.idType)>

<!ELEMENT QPTNAM.PhonNmbr_L (PhonNmbr_L.item+)>

<!ELEMENT StkID.id        (#PCDATA)>
<!ELEMENT StkID.idType    (#PCDATA)>

<!ELEMENT sndApp          (#PCDATA)>
<!ELEMENT rcvgApp         (#PCDATA)>
<!ELEMENT msgDt           (#PCDATA)>
<!ELEMENT msgTyp          (msgID, intrId?)>

<!ELEMENT msgID           (#PCDATA)>
<!ELEMENT intrId          (#PCDATA)>

<!ELEMENT fmn             (#PCDATA)>
<!ELEMENT gvn             (#PCDATA)>
<!ELEMENT mdn             (#PCDATA)>
<!ELEMENT sfx             (#PCDATA)>

<!ELEMENT cd              (#PCDATA)>
<!ELEMENT tx              (#PCDATA)>
<!ELEMENT cs              (#PCDATA)>
<!ELEMENT acd             (#PCDATA)>
<!ELEMENT atx             (#PCDATA)>
<!ELEMENT acs             (#PCDATA)>

<!ELEMENT PhonNmbr_L.item (XTN_C.emailAddr| XTN_C.ExtndTelId)>
<!ELEMENT XTN_C.emailAddr (#PCDATA)>
<!ELEMENT XTN_C.ExtndTelId (tlcmnUse, tlcmnEqpTyp, cntryCode?, areaCityCode?,
                            phonNmbr, xtnsn?, anytxt?)>

<!ELEMENT tlcmnUse        (%DT_CE;)>

<!ELEMENT tlcmnEqpTyp     (%DT_CE;)>

<!ELEMENT cntryCode       (#PCDATA)>
<!ELEMENT areaCityCode    (#PCDATA)>
```

```
<!ELEMENT phonNmbr        (#PCDATA)>
<!ELEMENT xtnsn           (#PCDATA)>
<!ELEMENT anytxt          (#PCDATA)>

<!ATTLIST PhonNmbr_L.item
          HL7_name  CDATA   #FIXED "Patient_Phone_Number_List.item"
          T         CDATA   "XTN_C">
<!ATTLIST QPTNAM.PhonNmbr_L
          HL7_name  CDATA   #FIXED "QryPatient_person_Nam.phone_number_list"
          T         CDATA   "PhonNmbr_L">
<!ATTLIST QPTNAM.StkID
          HL7_name  CDATA   #FIXED "QryPatient_person_Nam.Stakeholder_id"
          T         CDATA   "StkID">
<!ATTLIST QPTNAM.primrNamType
HL7_name  CDATA   #FIXED "QryPatient_person_Nam.primary_name_type_cd"
          T         CDATA   "CE">
<!ATTLIST QPTNAM.primrPrsnm
          HL7_name  CDATA   #FIXED "QryPatient_person_Nam.primary_prsnm"
          T         CDATA   "PN">
<!ATTLIST QRgNamv3P00
          HL7_name  CDATA   #FIXED "QRYNam_Patient_Reg"
          T         CDATA   "QRgNamv3P00">
<!ATTLIST QRgNamv3P00.MSGH
          HL7_name  CDATA   #FIXED "QRYNam_Patient_Reg.Message_header"
          T         CDATA   "MSGH">
<!ATTLIST QRgNamv3P00.QPTNAM
          HL7_name  CDATA   #FIXED "QRYNam_Patient_Reg.QryPatient_person_Nam"
          T         CDATA   "QPTNAM">
<!ATTLIST StkID.id
          HL7_name  CDATA   #FIXED "Stakeholder_ID.id"
          T         CDATA   "ST">
<!ATTLIST StkID.idType
          HL7_name  CDATA   #FIXED "Stakeholder_ID.identifier_type_cd"
          T         CDATA   "ID">
<!ATTLIST XTN_C.ExtndTelId
HL7_name  CDATA   #FIXED "Electronic_contact_address.Extended_telecomm_id"
          T         CDATA   "XTN"
          Choice    CDATA   "T">
<!ATTLIST XTN_C.emailAddr
          HL7_name  CDATA   #FIXED "Electronic_contact_address.email_address"
          T         CDATA   "ST"
          Choice    CDATA   "E">
<!ATTLIST acd
          HL7_name  CDATA   #FIXED "alt_identifier"
          T         CDATA   "ST">
<!ATTLIST acs
          HL7_name  CDATA   #FIXED "alt_name_coding_system"
          T         CDATA   "ST">
<!ATTLIST anytxt
          HL7_name  CDATA   #FIXED "any_text"
          T         CDATA   "TX">
<!ATTLIST areaCityCode
          HL7_name  CDATA   #FIXED "area_city_code"
          T         CDATA   "NM">
<!ATTLIST atx
          HL7_name  CDATA   #FIXED "alt_identifier_text"
          T         CDATA   "ST">
<!ATTLIST cd
          HL7_name  CDATA   #FIXED "identifier"
          T         CDATA   "ST">
<!ATTLIST cntryCode
          HL7_name  CDATA   #FIXED "country_code"
          T         CDATA   "NM">
```

```
<!ATTLIST cs
          HL7_name  CDATA   #FIXED "name_of_coding_system"
          T         CDATA   "ST">
<!ATTLIST fmn
          HL7_name  CDATA   #FIXED "family_name"
          T         CDATA   "ST">
<!ATTLIST gvn
          HL7_name  CDATA   #FIXED "given_name"
          T         CDATA   "ST">
<!ATTLIST intrId
          HL7_name  CDATA   #FIXED "interaction_id"
          T         CDATA   "ID">
<!ATTLIST mdn
          HL7_name  CDATA   #FIXED "middle_initial_or_name"
          T         CDATA   "ST">
<!ATTLIST msgDt
          HL7_name  CDATA   #FIXED "date_time_message"
          T         CDATA   "DTM">
<!ATTLIST msgID
          HL7_name  CDATA   #FIXED "message_id"
          T         CDATA   "ID">
<!ATTLIST msgTyp
          HL7_name  CDATA   #FIXED "message_type"
          T         CDATA   "MSGT">
<!ATTLIST phonNmbr
          HL7_name  CDATA   #FIXED "phone_number"
          T         CDATA   "NM">
<!ATTLIST rcvgApp
          HL7_name  CDATA   #FIXED "receiving_application"
          T         CDATA   "ID">
<!ATTLIST sfx
          HL7_name  CDATA   #FIXED "suffix"
          T         CDATA   "ST">
<!ATTLIST sndApp
          HL7_name  CDATA   #FIXED "sending_application"
          T         CDATA   "ID">
<!ATTLIST tlcmnEqpTyp
          HL7_name  CDATA   #FIXED "telecomm_equipment_type"
          T         CDATA   "CE">
<!ATTLIST tlcmnUse
          HL7_name  CDATA   #FIXED "telecom_use_code"
          T         CDATA   "CE">
<!ATTLIST tx
          HL7_name  CDATA   #FIXED "identifier_text"
          T         CDATA   "ST">
<!ATTLIST xtnsn
          HL7_name  CDATA   #FIXED "extension"
          T         CDATA   "NM">

<!--============== End HL7 Version 3 Prototype Message DTD ==============-->
```

Figure 8. XML Instance Example for the Toy Message.

```
<?xml version = "1.0">
<!DOCTYPE QRgNamv3P00 SYSTEM "ToyMessage.dtd">
<QRgNamv3P00 T="QRgNamv3P00">
    <QRgNamv3P00.MSGH T="MSGH">
        <sndApp T="ID">INTRANET001</sndApp>
        <rcvgApp T="ID">RGSIUIPI</rcvgApp>
        <msgDt T="DTM">199808231123</msgDt>
        <msgTyp T="MSGT">
             <msgID T="ID">QRgNamv3P00</msgID>
        </msgTyp>
    </QRgNamv3P00.MSGH>
    <QRgNamv3P00.QPTNAM T="QPTNAM">
        <QPTNAM.primrNamType T="CE">
            <cd T="ST">L</cd>
            <tx T="ST">Local</tx>
            <cs T="ST">HL79999</cs>
        </QPTNAM.primrNamType>
        <QPTNAM.primrPrsnm T="PN">
            <fmn T="ST">Newman</fmn>
        </QPTNAM.primrPrsnm>
        <QPTNAM.PhonNmbr_L T="PhonNmbr_L">
            <PhonNmbr_L.item T="XTN_C">
                <XTN_C.emailAddr T="ST" Choice="E">what@me.worry.com</XTN_C.emailAddr>
            </PhonNmbr_L.item>
            <PhonNmbr_L.item T="XTN_C">
                <XTN_C.ExtndTelId T="XTN" Choice="T">
                    <tlcmnUse T="CE">
                        <cd T="ST">PRN</cd>
                        <tx T="ST">Primary Residence</tx>
                        <cs T="ST">HL79999</cs>
                    </tlcmnUse>
                    <tlcmnEqpTyp T="CE">
                        <cd T="ST">PH</cd>
                        <tx T="ST">Telephone</tx>
                        <cs T="ST">HL79999</cs>
                    </tlcmnEqpTyp>
                       <areaCityCode T="NM">555</areaCityCode>
                       <phonNmbr T="NM">7776666</phonNmbr>
                </XTN_C.ExtndTelId>
            </PhonNmbr_L.item>
        </QPTNAM.PhonNmbr_L>
    </QRgNamv3P00.QPTNAM>
</QRgNamv3P00>
```