Clinical Element Datatypes

Joey Coyle
Yan Heras
Tom Oniki
Stan Huff

November 14, 2008
## Contents

1 Datatypes for Clinical Elements ..... 15

1 Introduction ..... 17
  1.1 Datatypes ..... 17
  1.2 HL7 attributes and methods ..... 18

2 ANY ..... 21
  2.1 Any ..... 21
  2.2 Properties ..... 22
    2.2.1 nullFlavor ..... 22
  2.3 HL7 Comparison ..... 23
    2.3.1 dataType ..... 23
    2.3.2 nullFlavor ..... 24

3 CWE ..... 25
  3.1 Coded With Exceptions ..... 25
  3.2 Properties ..... 26
    3.2.1 code ..... 26
    3.2.2 originalText ..... 26
    3.2.3 translation ..... 26
  3.3 Constraint Properties ..... 27
    3.3.1 domain ..... 27
  3.4 HL7 Comparison ..... 28
    3.4.1 code ..... 29
    3.4.2 displayName ..... 30
    3.4.3 codeSystem ..... 30
    3.4.4 codeSystemName ..... 30
    3.4.5 codeSystemVersion ..... 30
    3.4.6 valueSet ..... 30
    3.4.7 valueSetVersion ..... 30
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.8</td>
<td>originalText ................................... 30</td>
</tr>
<tr>
<td>3.4.9</td>
<td>codingRationale ................................ 31</td>
</tr>
<tr>
<td>3.4.10</td>
<td>translation ..................................... 31</td>
</tr>
<tr>
<td>3.4.11</td>
<td>source .......................................... 31</td>
</tr>
<tr>
<td>3.4.12</td>
<td>isCompositional ................................ 31</td>
</tr>
<tr>
<td>3.4.13</td>
<td>equal ............................................ 31</td>
</tr>
<tr>
<td>3.4.14</td>
<td>implies .......................................... 31</td>
</tr>
<tr>
<td>3.4.15</td>
<td>Properties inherited from HL7:ANY ............ 31</td>
</tr>
<tr>
<td>4</td>
<td>CO                                              33</td>
</tr>
<tr>
<td>4.1</td>
<td>Coded Ordinal .................................... 33</td>
</tr>
<tr>
<td>4.2</td>
<td>Properties ....................................... 34</td>
</tr>
<tr>
<td>4.2.1</td>
<td>code ............................................. 34</td>
</tr>
<tr>
<td>4.2.2</td>
<td>originalText .................................... 34</td>
</tr>
<tr>
<td>4.2.3</td>
<td>translation ...................................... 34</td>
</tr>
<tr>
<td>4.2.4</td>
<td>value ............................................ 34</td>
</tr>
<tr>
<td>4.3</td>
<td>Constraint Properties ............................ 35</td>
</tr>
<tr>
<td>4.3.1</td>
<td>domain .......................................... 35</td>
</tr>
<tr>
<td>4.4</td>
<td>HL7 Comparison ................................... 36</td>
</tr>
<tr>
<td>4.4.1</td>
<td>value ............................................ 36</td>
</tr>
<tr>
<td>4.4.2</td>
<td>lessOrEqual ...................................... 37</td>
</tr>
<tr>
<td>4.4.3</td>
<td>lessThan ......................................... 37</td>
</tr>
<tr>
<td>4.4.4</td>
<td>greaterThan ...................................... 37</td>
</tr>
<tr>
<td>4.4.5</td>
<td>greaterOrEqual .................................. 37</td>
</tr>
<tr>
<td>4.4.6</td>
<td>Properties inherited from HL7:CD ............. 38</td>
</tr>
<tr>
<td>5</td>
<td>II                                             39</td>
</tr>
<tr>
<td>5.1</td>
<td>Instance Identifier .............................. 39</td>
</tr>
<tr>
<td>5.2</td>
<td>Properties ....................................... 40</td>
</tr>
<tr>
<td>5.2.1</td>
<td>root .............................................. 40</td>
</tr>
<tr>
<td>5.2.2</td>
<td>extension ........................................ 40</td>
</tr>
<tr>
<td>5.2.3</td>
<td>displayable ...................................... 40</td>
</tr>
<tr>
<td>5.3</td>
<td>HL7 Comparison ................................... 41</td>
</tr>
<tr>
<td>5.3.1</td>
<td>extension ........................................ 42</td>
</tr>
<tr>
<td>5.3.2</td>
<td>root .............................................. 42</td>
</tr>
<tr>
<td>5.3.3</td>
<td>identifierName .................................. 42</td>
</tr>
<tr>
<td>5.3.4</td>
<td>scope ............................................ 42</td>
</tr>
<tr>
<td>5.3.5</td>
<td>reliability ...................................... 42</td>
</tr>
<tr>
<td>5.3.6</td>
<td>displayable ...................................... 42</td>
</tr>
<tr>
<td>5.3.7</td>
<td>equal ............................................ 42</td>
</tr>
</tbody>
</table>
5.3.8 Properties inherited from HL7:ANY
6 INT
6.1 Integer
6.2 Properties
6.2.1 value
6.2.2 operator
6.3 Constraint Properties
6.3.1 minInclusive
6.3.2 maxInclusive
6.3.3 minExclusive
6.3.4 maxExclusive
6.4 HL7 Comparison
6.4.1 value
6.4.2 operator
6.4.3 successor
6.4.4 times
6.4.5 predecessor
6.4.6 negated
6.4.7 isNegative
6.4.8 nonNegative
6.4.9 dividedBy
6.4.10 remainder
6.4.11 isOne
6.4.12 Properties inherited from HL7:QTZ
7 PQ
7.1 Physical Quantity
7.2 Properties
7.2.1 value
7.2.2 unit
7.2.3 operator
7.2.4 translation
7.3 Constraint Properties
7.3.1 normal
7.3.2 minInclusive
7.3.3 maxInclusive
7.3.4 minExclusive
7.3.5 maxExclusive
7.4 HL7 Comparison
11.2.2 mediaType ........................................ 77
11.2.3 language ........................................ 77
11.2.4 compression ..................................... 77
11.2.5 integrityCheck ................................. 77
11.2.6 integrityCheckAlgorithm ...................... 77
11.2.7 reference ........................................ 78
11.2.8 thumbnail ....................................... 78
11.3 HL7 Comparison ................................. 79
  11.3.1 data ........................................... 79
  11.3.2 mediaType ..................................... 79
  11.3.3 charset ........................................ 79
  11.3.4 language ...................................... 80
  11.3.5 compression .................................. 80
  11.3.6 reference ..................................... 80
  11.3.7 integrityCheck ............................... 80
  11.3.8 integrityCheckAlgorithm .................. 81
  11.3.9 description ................................... 81
  11.3.10 thumbnail ................................... 81
  11.3.11 translation .................................. 81
  11.3.12 length ....................................... 81
  11.3.13 subpart ...................................... 81
  11.3.14 equal ........................................ 81
12 IVLPQ ............................................... 83
  12.1 Interval Physical Quantity ...................... 83
  12.2 Properties ....................................... 83
    12.2.1 low .......................................... 83
    12.2.2 high ......................................... 84
  12.3 HL7 Comparison ................................. 85
    12.3.1 low .......................................... 86
    12.3.2 lowClosed ................................... 86
    12.3.3 high ......................................... 86
    12.3.4 highClosed .................................. 86
    12.3.5 width ........................................ 86
    12.3.6 center ....................................... 86
    12.3.7 any ........................................... 87
    12.3.8 hull .......................................... 87
## 13 RTOPQ

13.1 Ratio Physical Quantity ........................................ 89
13.2 Properties ........................................................... 90
   13.2.1 numerator ....................................................... 90
   13.2.2 denominator .................................................... 90
13.3 HL7 Comparison ....................................................... 91
   13.3.1 numerator ....................................................... 91
   13.3.2 denominator .................................................... 91

## II Datatypes for Internal Use

14 PQT

14.1 Physical Quantity Translation ..................................... 95
14.2 Properties ........................................................... 96
   14.2.1 value ............................................................ 96
   14.2.2 unit ............................................................. 96
   14.2.3 codeSystem .................................................... 97
   14.2.4 codeSystemVersion ............................................ 97
   14.2.5 operator ......................................................... 97
14.3 HL7 Comparison ....................................................... 98
   14.3.1 value ............................................................ 99
   14.3.2 operator ......................................................... 99
   14.3.3 unit ............................................................. 99
   14.3.4 code ............................................................ 99
   14.3.5 displayName .................................................... 99
   14.3.6 codeSystem ..................................................... 99
   14.3.7 codeSystemName ............................................... 100
   14.3.8 codeSystemVersion ............................................ 100
   14.3.9 valueSet ....................................................... 100
   14.3.10 valueSetVersion ............................................... 100
   14.3.11 originalText .................................................. 100
   14.3.12 codingRationale .............................................. 100
   14.3.13 isCompositional .............................................. 100
   14.3.14 equal .......................................................... 100
   14.3.15 implies ......................................................... 101
14.3.16 Properties inherited from **HL7:ANY** ......................... 101
17 COT
17.1 Coded Ordinal - Translation ........................... 115
17.2 Properties ............................................. 116
  17.2.1 code ............................................... 116
  17.2.2 originalText ....................................... 116
  17.2.3 codeSystem ......................................... 116
  17.2.4 codeSystemVersion ................................. 117
  17.2.5 value .............................................. 117
17.3 HL7 Comparison ........................................ 118
  17.3.1 value ............................................... 119
  17.3.2 lessOrEqual ........................................ 119
  17.3.3 lessThan ........................................... 119
  17.3.4 greaterThan ........................................ 119
  17.3.5 greaterOrEqual ..................................... 119
  17.3.6 Properties inherited from HL7:CD .................. 119
18 CS
18.1 Coded Simple ........................................... 121
18.2 Properties ............................................. 122
  18.2.1 code ............................................... 122
18.3 HL7 Comparison ........................................ 123
  18.3.1 code ............................................... 123
  18.3.2 Properties inherited from HL7:ANY .................. 123
19 CWENT
19.1 Coded With Exceptions - No Translation ............... 125
19.2 Properties ............................................. 125
  19.2.1 code ............................................... 125
  19.2.2 originalText ....................................... 126
19.3 Constraint Properties .................................. 126
  19.3.1 domain ............................................. 126
19.4 HL7 Comparison ........................................ 127
  19.4.1 code ............................................... 128
19.4.2 displayName ................................. 128
19.4.3 codeSystem ................................. 128
19.4.4 codeSystemName ............................ 128
19.4.5 codeSystemVersion ......................... 128
19.4.6 valueSet ................................. 128
19.4.7 valueSetVersion ................................ 128
19.4.8 originalText ............................... 128
19.4.9 codingRationale ........................... 129
19.4.10 translation ................................ 129
19.4.11 source ................................ 129
19.4.12 isCompositional .......................... 129
19.4.13 equal ................................ 129
19.4.14 implies .................................. 129
19.4.15 Properties inherited from **HL7:ANY** .... 129

20 EDNT ............................................ 131
20.1 Encapsulated Data - No Thumbnail .......... 131
20.2 Properties .................................... 131
   20.2.1 data ................................ 132
   20.2.2 mediaType ................................ 133
   20.2.3 language ................................ 133
   20.2.4 compression ............................. 133
   20.2.5 integrityCheck ......................... 133
   20.2.6 integrityCheckAlgorithm .............. 133
   20.2.7 reference ................................ 133
20.3 HL7 Comparison ............................. 135
   20.3.1 data ................................ 135
   20.3.2 mediaType ................................ 135
   20.3.3 charset ................................ 135
   20.3.4 language ................................ 135
   20.3.5 compression ............................. 136
   20.3.6 reference ................................ 136
   20.3.7 integrityCheck ......................... 136
   20.3.8 integrityCheckAlgorithm .............. 137
   20.3.9 description .............................. 137
   20.3.10 thumbnail .............................. 137
   20.3.11 translation .............................. 137
   20.3.12 length ................................ 137
   20.3.13 subpart ................................ 137
   20.3.14 equal ................................ 137
# CONTENTS

## 21 BIN

21.1 Binary Data ........................................ 139  
21.2 Properties ........................................ 139  
  21.2.1 value ........................................ 140  
21.3 HL7 Comparison .................................... 141  
  21.3.1 value ........................................ 141

### III Appendix ........................................ 143

#### A CWE Cases ....................................... 145  
  A.1 Introduction ........................................ 145  
  A.2 Application provides code ............................ 145  
  A.3 Application provides code-originalText ............... 146  
  A.4 Application provides originalText - post-coded ....... 146  
  A.5 Interface provides code .............................. 147  
  A.6 Interface provides code-originalText ................. 148  
  A.7 Interface provides code-code system info ............ 149  
  A.8 Interface provides code-translation ................... 150  
  A.9 Unknown coding system ................................ 151

#### B PQ Cases .......................................... 153  
  B.1 Introduction ........................................ 153  
  B.2 Application provides unit code ........................ 153  
  B.3 Application provides unit code-originalText ............ 154  
  B.4 Application provides originalText - post-coded ....... 154  
  B.5 Interface provides code ................................ 155  
  B.6 Interface provides code, needs normalization ........ 156  
  B.7 Interface unknown coding system ........................ 157  
  B.8 Interface provides code and normalized translation .... 158

#### C Glossary ......................................... 161  
  C.1 Definitions ........................................ 161

#### D Change Log ....................................... 167  
  D.1 November 14, 2008 ................................... 167
Part I

Datatypes for Clinical Elements
Chapter 1

Introduction

Each detailed clinical model in the Clinical Element model defines a value choice between a datatype or a list of child Clinical Elements. This can be thought of as the payload of the model. This discussion will be regarding the value... The following section describes the datatypes used when a Clinical Element is modeled to contain a datatype. These datatypes are based on the design of the HL7 version 3 datatypes. We have chosen to use only a subset of the HL7 datatypes, and the datatypes used have been modified by the addition and deletion of various attributes we found necessary for use in a permanent datastore. In addition, it should be understood that the HL7's datatype UML description is a hierarchy of types. The Clinical Element implementation of the datatypes is not a hierarchy, but a flat declaration of the types with the appropriate attributes. To understand the following chapters, it is best to have some familiarity with the HL7 datatypes.

1.1 Datatypes

The Value Choice of a Clinical Element may use one of the following datatypes.

**Datatypes for Clinical Element**

---

1. As part of HL7’s RIM, a set of datatypes called the HL7 version 3 datatypes have been defined. These datatypes are used to represent actual information such as numbers, strings, and codes. A UML and textual description of the HL7 datatypes is available from HL7. HL7 has implemented the datatypes in XML schema, and is now working with Sun Microsystems to develop them in Java. HL7’s UML description describes methods and properties, which are both implemented in the Java types, yet the XML Schema only implements the properties.

2. It must be remembered, that the HL7 messages were not designed to be used in a permanent datastore; they were designed to be a temporary exchange format between disparate systems.

---
1.2. HL7 ATTRIBUTES AND METHODS

- CWE
- CO
- PQ
- ED
- TS
- ST
- REAL
- INT
- II
- RTOPQ

In addition to these datatypes, we have also defined the following datatypes, which are either used to define the previous datatypes, or are used directly by parts of the Clinical Element Model.

**Datatypes for use in other datatypes**

- ANY
- CS
- COT
- CET
- PQR
- EDNT
- CWENT
- TEL
- OID
- CNE

1.2 HL7 attributes and methods

The HI7 version 3 datatypes that we have chosen to implement are composite data types, meaning they have substructure with internal fields. These internal fields of the datatypes define both data properties which can house pieces of data, and derived properties which return a value based on the data properties. For example, the is-zero property which returns a boolean true or false, is derived from the value property by checking whether the value is zero or not. For now, our CEM datatypes ignore the derived properties of the HL7 datatypes. We will leave the derived
properties for a later implementation stage of development.

Our datatypes are more similar to the HL7 XML ITS, than the abstract specification. Unfortunately in R2 of the HL7 specification, these two specifications are not yet in sync.
Chapter 2

ANY

2.1 Any

ANY (Figure 2.1) is the base datatype in the HL7 hierarchy and all other datatypes derive from this type. ANY includes the attribute nullFlavor which is used to indicate the coded reason for the absence of data. Examples of these coded reasons include unknown (UNK) and not applicable (NA).

Our implementation should support nullFlavor for our datatypes, either by inheritance of nullFlavor or the direct declaration of nullFlavor in each datatype. ¹

¹In a previous implementation of the Clinical Element model, nullFlavor was not part of each datatype, but was itself its own datatype. Any Clinical Element instance could then instantiate this nullFlavor datatype as needed, rather than its own defined datatype. This was also an easy way to implement the co-occurrence constraints inherent in the use of nullFlavor.
2.2 Properties

Table 2.1: ANY Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nullFlavor</td>
<td>ST</td>
<td>A code describing the reason a datatype has a null value</td>
</tr>
</tbody>
</table>

2.2.1 nullFlavor

The property nullFlavor is a code providing the reason a datatype has a null value
2.3 HL7 Comparison

Figure 2.2: HL7:ANY declaration

```
abstract type DataValue alias ANY {
    TYPE   dataType;
    CS     nullFlavor;
    BN     nonNull;
    BN    isNull;
    BL     notApplicable;
    BL     unknown;
    BL     other;
    BL     equal(ANY x);
    protected BN  identical(ANY x);
};
```

Figure 2.3: ANY to HL7:ANY Comparison

The ANY datatype is the base type of the HL7 version 3 datatypes. The property `nullFlavor` is the only non-derived data property, and is the property we use for comparison in figure 2.3. The property `dataType`, while not exactly derived, is implied by the implementation, which means the implementation will implicitly know the datatype of the instanciated datatype. The properties `nonNull`, `isNull`, `notApplicable`, `unknown`, and `other` are all derived from the value in `nullFlavor`. The properties `equal` and `identical` are comparison methods used to evaluate the given datatype instance as compared to another. Thus, only the property `nullFlavor` is of importance to us.

2.3.1 `dataType`

We do not include the property `dataType`, because this can be resolved by the implementation. For example, the implementation will know if it is dealing with a `PQ` or a `CWE`.  

23
2.3.2 nullFlavor

We include the property nullFlavor.
Chapter 3

CWE

3.1 Coded With Exceptions

Coded With Exceptions (Figure 3.1) is used to store coded values. In our Clinical Element models, we will always use CWE for coded values. The reason for this decision is that CWE permits translations of the code, and for a storage model, translations will occur very often. Also, CWE permits the use of a textual description in lieu of a code from the primary coding system. It is important for us to store such data, because although this data is not coded and can not be operated on by decision support, a physician can still make timely use of this data.

Due to the performance requirements of a permanent storage based system,
we have chosen to only allow codes from the primary coding system in code, and alternate codes from other coding systems will be placed in translation.

For our storage form we have decided to only allow a single CET within translation. We have removed the property codingRationale, and instead rely on the following rule to determine if the code is original. If a translation exists, then the translation is the original code, and if it does not exist, then the code within CWE is the original code.\(^1\)

### 3.2 Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>ST.SIMPLE</td>
<td>A code for a concept defined in the primary code system.</td>
</tr>
<tr>
<td>originalText</td>
<td>ST.SIMPLE</td>
<td>Textual representation of the code</td>
</tr>
<tr>
<td>translation</td>
<td>SET&lt;CET&gt;</td>
<td>Zero to many translations of the code</td>
</tr>
</tbody>
</table>

#### 3.2.1 code

The property code contains a code for a concept defined in the primary coding system.

#### 3.2.2 originalText

The property originalText is used for textual representation of the code that was presented to the user, or the representation that came over the interface.

#### 3.2.3 translation

The property translation is used to represent Zero to many translations of the code from any code system. We have decided that in our storage system, we will only retain one translation.\(^2\)

\(^1\)For use cases see appendix chapter A.

\(^2\)We need further discussion to determine which translations get discarded and which remains. Do we keep the source or the original translation?
3.3 Constraint Properties

Table 3.2: CWE Constraint Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain</td>
<td>ST</td>
<td>A code for a domain of concepts</td>
</tr>
</tbody>
</table>

3.3.1 domain

The property domain is a code for a domain of concepts defined in the primary code system.
### 3.4 HL7 Comparison

Figure 3.2: HL7 CD declaration

```plaintext
type ConceptDescriptor alias CD specializes ANY {
    ST.SIMPLE code;
    ST displayName;
    UID codeSystem;
    ST.NT codeSystemName;
    ST.SIMPLE codeSystemVersion;
    UID valueSet;
    TS.DATETIME.FULL valueSetVersion;
    ED.TEXT originalText;
    SET<CS> codingRationale;
    DSET<CD> translation;
    CD source;
    BL isCompositional;
    BL equal(ANY x);
    BL implies(CD x);
};
```

The Clinical Element CWE datatype is much alike a HL7:CD with the coding strength qualifier CWE, except that a code outside the specified value set/code system may not be used. But, it is very often the case that we might receive text in lieu of a code, and we still need to store such data, so CWE makes sense.

For performance requirements, we have decided that only codes from the primary coding system will be allowed in `code`. The `codeSystem`, `codeSystemName`, and `codeSystemVersion` components are dropped because they are defaulted to the primary coding system. The `displayName` component is dropped because it’s an optional HL7 component that supplies default text, and we saw no practical reason for designating default text that would be usable across the entire enterprise.³

**HL7:CD** translation is defined as a `<SET>HL7:CD`, but we created `<SET>CET` for our translation. By defining CET, we were able to remove recursive translations, as well as remove other properties that we had stripped out of **HL7:CD**.

**HL7:CD** defines `originalText` as an **HL7:ED** datatype. We decided that the complexity of the **HL7:ED** datatype was not worth the cost, so we defined `originalText` to be an `ST`. If we need to store a thumbnail, or sound byte in `originalText`,

³In a previous implementation we had included the attribute `displayName` to be used as a best guest surface form for performance denormalization. Engineering decided they had easier ways to solve any performance problems with realtime data dictionary lookups, so `displayName` was removed.
Figure 3.3: CWE to HL7:CE Comparison

Figure 3.4: HL7 CE and Coding Strength Qualifiers

**CWE - Coded With Exceptions** This is a coding strength qualifier that is applied to a binding of a vocabulary domain with a CD. It signifies that a code outside the specified value set/code system may be used, or that free text may be used in lieu of a code.

**CNE - Coded, No Exceptions** This is another coding strength qualifier that is applied to a binding of a vocabulary domain with a CD. It signifies that a code outside the specified value set/code system is not permitted, nor is free text permitted in lieu of a code.

then one proposed solution would be to store a textual pointer, such as a URI, to the file.

**3.4.1 code**

We include the property code.
3.4.2 displayName

We do not include `displayName` due to the fact we only allow codes from our primary coding system in CWE, and a `displayName` can be generated at any time from the vocabulary server.

3.4.3 codeSystem

We do not include `codeSystem` because in CWE the `codeSystem` is defaulted to the primary coding system.

3.4.4 codeSystemName

We do not include `codeSystemName` because in CWE the `codeSystemName` is defaulted to the primary coding system.

3.4.5 codeSystemVersion

We do not include `codeSystemVersion` because in our primary coding system we don’t use versioning.\(^4\)

3.4.6 valueSet

We do not currently include the `valueSet`, which specifies the value set that applied when this CD was created.\(^5\)

3.4.7 valueSetVersion

We do not currently include the `valueSetVersion`, which specifies the value set version that applied when this CD was created. Even if we decided to add the `valueSet` property, I believe the version could be handled by the vocabulary server.

3.4.8 originalText

We include the property `originalText`.

---

\(^4\)I need to double check this.

\(^5\)We have never discussed the need for this.
3.4.9 **codingRationale**

We do not include the property `codingRationale`. For better or worse, we assume it is original if there isn’t a translation.\(^6\)

3.4.10 **translation**

We included the property `translation`.

3.4.11 **source**

We do not include the property `source`. This property identifies the translation from which this was translated. We have not discussed this issue, but it seems this is implicit due to the fact we are only allowing one translation in the storage form. The question remains whether or not this is important transactionally when we allow more than one translation.

3.4.12 **isCompositional**

We do not include the property `isCompositional`. This can be derived from the vocabulary server.

3.4.13 **equal**

We do not include the property `equal`. This is a comparison operation that can be handled by the vocabulary server.

3.4.14 **implies**

We do not include the property `implies`. This is a comparison operation that can be handled by the vocabulary server.

3.4.15 **Properties inherited from HL7:ANY**

Please see the chapter on `HL7:ANY`.

---

\(^6\)Tom brought up a good point, that we are only using part of the functionality that `codingRationale` provides, and we have not discussed the other aspects.
Chapter 4

CO

4.1 Coded Ordinal

Coded Ordinal (Figure 4.1) is used to store coded ordinal data such as the code for “2+”, which comes from an ordered domain of codes.

Due to the performance requirements of a permanent storage based system, we have chosen to only allow codes from the primary coding system in code, and alternate codes from other coding systems will be placed in translation.
Previously, we had the property *operator* in CO, but this has been removed, and there doesn’t seem to be any documentation explaining this.

### 4.2 Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>ST</td>
<td>A code for a concept defined in the primary code system.</td>
</tr>
<tr>
<td>originalText</td>
<td>ST</td>
<td>Textual representation of the code</td>
</tr>
<tr>
<td>translation</td>
<td>SET&lt;COT&gt;</td>
<td>Zero to many translations of the code</td>
</tr>
<tr>
<td>value</td>
<td>REAL</td>
<td>A numeric representation of the code’s meaning</td>
</tr>
</tbody>
</table>

#### 4.2.1 code

The property *code* is a code for a concept defined in the primary code system.

#### 4.2.2 originalText

The property *originalText* is a textual representation of the code that was presented to the user, or the representation that came over the interface.

#### 4.2.3 translation

The property *translation* is used to represent a Zero to many translations of the code from any code system.

#### 4.2.4 value

The property *value* provides for a numeric representation of the code’s meaning.
4.3 Constraint Properties

4.3.1 domain

The property *domain* represents a code for a domain of concepts defined in the primary code system.
4.4 HL7 Comparison

Figure 4.2: HL7:CO Declaration

```plaintext
type CodedOrdinal alias CO specializes CD {
    REAL value;
    BL lessOrEqual(CO o);
    BL lessThan(CO o);
    BL greaterThan(CO o);
    BL greaterOrEqual(CO o);
};
```

HL7:CO specializes HL7:CD, and only adds one additional data property which is value. We include the other properties inherited from HL7:CD as we did in CWE and CNE so please see either of these chapters for the reason for inclusion or exclusion of any data properties.

For performance requirements, we have decided that only codes from the primary coding system will be allowed in code. The codeSystem, codeSystemName, and codeSystemVersion components are dropped because they are defaulted to the primary coding system. The displayName component is dropped because it is an optional HL7 component that supplies default text, and we saw no practical reason for designating default text that would be usable across the entire enterprise.

HL7 does not allow translations for HL7:CO but we have relaxed this and allow translations within our CO and have defined this a SET<COT>.

HL7:CO defines originalText as an HL7:ED datatype. We decided that the complexity of the HL7:ED datatype was not worth the cost, so we defined originalText to be an ST. If we need to store a thumbnail, or sound byte in originalText, then one proposed solution would be to store a textual pointer, such as a URI, to the file.

### 4.4.1 value

We include the property value.

---

1. Previously HL7 didn’t have value in HL7:CO, so we added it to our CO and called it numericOperator. Now that HL7 has added this, we have changed the name of numericOperator to value.
2. In a previous implementation we had included the attribute displayName to be used as a best guest surface form for performance denormalization. Engineering decided they had easier ways to solve any performance problems with realtime data dictionary lookups, so displayName was removed.
3. We previously called this numericScore.
4.4.2 lessOrEqual

We do not include the property lessOrEqual because it is a comparison operator.

4.4.3 lessThan

We do not include the property lessThan because it is a comparison operator.

4.4.4 greaterThan

We do not include the property greaterThan because it is a comparison operator.

4.4.5 greaterOrEqual

We do not include the property greaterOrEqual because it is a comparison operator.
4.4.6 Properties inherited from HL7:CD

Please see the chapter on HL7:CD
Chapter 5

II

5.1 Instance Identifier

Figure 5.1: Instance Identifier

Instance Identifier (Figure 5.1) is used to uniquely identify an instance, thing or object. Examples of this include social security numbers or medical record numbers.

\[^1\]We need to discuss root, codes, and oids
5.2 Properties

Table 5.1: II Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>CS</td>
<td>A unique identifier that guarantees the global uniqueness</td>
</tr>
<tr>
<td>extension</td>
<td>ST</td>
<td>A unique identifier within the scope of the identifier root</td>
</tr>
<tr>
<td>displayable</td>
<td>CS</td>
<td>For human display or pure machine interoper- ation</td>
</tr>
</tbody>
</table>

5.2.1 root

The property root provides a unique identifier that guarantees the global uniqueness of the instance identifier. The root alone may be the entire instance identifier. An example of this would be an identifier that represents the concept of Social Security Number.

5.2.2 extension

The property extension is a character string that is unique in the namespace designated by the root. The extension property may be null in which case the root UID is the complete unique identifier. The root and extension scheme effectively means that the concatenation of root and extension must be a globally unique identifier for the item that this II value identifies.

5.2.3 displayable

The property displayable specifies if the identifier is intended for human display and data entry (displayable = true) as opposed to pure machine interoper- ation (displayable = false).

One of the use cases of displayable is that when a DICOM image is received, the OID represents the DICOM imaging and the extension is empty. The displayable is useful here to identify that the extension is not intended for human readable.
5.3 HL7 Comparison

Figure 5.2: HL7:II Declaration

```plaintext
type InstanceIdentifier alias II specializes ANY {
  ST.SIMPLE  extension;
  UID        root;
  ST.NT      identifierName;
  CS         scope;
  CS         reliability;
  BL         displayable;
  BL         equal(ANY x);
  literal ST.NT
};
```

Figure 5.3: II to HL7:II Comparison

We use the CS data type to represent the root property where HL7 uses HL7:OID. We do this to keep a consistent coding system in our repository. All registered OIDs will be incorporated into our vocabulary server as concepts and the OID will exist as one possible surface form for the concept.

In our version of II, we have removed the property assigningAuthority, but a code for the Assigning Authority is maintained in our coding system and mapped to the root so we can always determine the Assigning Authority from the root.
5.3.1 **extension**

We include the property *extension*.

5.3.2 **root**

We include the property *root*, but we have changed the type from an HL7:UID to a CS. We will be able to retrieve the HL7:UID representation from the vocabulary server.

5.3.3 **identifierName**

We do not include the property *identifierName* which is a human readable form of the HL7:UID, since this can just be retrieved as just another representation in the vocabulary server.

5.3.4 **scope**

We do not include the property *scope*. We feel the information provided by scope will be known by the context the III is used within a Clinical Element model.²

5.3.5 **reliability**

We do not include the property *reliability*. The issue of reliability of the value seems to be a global problem to any datatype, and we would move this out of the datatype and into a qualifier in the clinical element model.³

5.3.6 **displayable**

We do include the property *displayable*. But I’m wondering if this is correct? The answer to this should be the same for every given root. Thus the knowledge repository should be able to answer this question based on the root.⁴

5.3.7 **equal**

We do not include the property *equal*. This is just a comparison operator between two instance identifiers.

²Need to discuss with modeling team
³Need to discuss with modeling team
⁴Need to discuss with modeling team
5.3.8 Properties inherited from HL7:ANY

Please see the chapter on HL7:ANY
6.1 Integer

Integer Number (Figure 6.1) is used to represent an integer number ranging from negative to positive infinity. Operator was added by IHC to represent codes such as “greater than” or “less than”.

Examples include... 0, 1, 2, 100, 3398129, -12.
6.2 Properties

6.2.1 value

The property value represents the value of the integer number, e.g. 5, 26. Precision must be maintained by the implementation.

6.2.2 operator

The property operator provides a coded value that will represent either >, <, >=, or <=.

6.3 Constraint Properties

6.3.1 minInclusive

The property minInclusive is an integer that value must be greater than or equal to.

6.3.2 maxInclusive

The property maxInclusive is an integer that value must be less than or equal to.
6.3.3 minExclusive

The property minExclusive is an integer that value must be greater than.

6.3.4 maxExclusive

The property maxExclusive is an integer that value must be less than.
6.4 HL7 Comparison

Figure 6.2: HL7:INT Declaration

```plaintext
type IntegerNumber alias INT specializes QTZ {
  INT successor;
  INT times(INT x);
  INT predecessor;
  INT negated;
  BL isNegative;
  BL nonNegative;
  REAL dividedBy(REAL x);
  INT dividedBy(INT x);
  INT remainder(INT x);
  BL isOne;
  literal ST.SIMPLE;
};
```

Figure 6.3: INT to HL7:INT Comparison

Where HL7 would use an IVL<INT> to represent “greater than” or “less than” an integer, we use a coded operator within INT. For simplicity, we will not implement HL7’s IVL<INT> in our system.
6.4.1 value

The property value is not part of the HL7 abstract definition. The HL7 specification leaves it up to implementation where to store the numeric value of an integer. We have chosen to use value which is the same choice HL7 made in their version 3 XML ITS.

6.4.2 operator

The property operator is not part of the HL7 abstract definition. While HL7 would use an IVL<INT> to represent “greater than” or “less than” an integer, we use a coded operator within INT. For simplicity, we will not implement HL7’s IVL<INT> in our system.

6.4.3 successor

We do not include the property successor because it can be derived. This successor is just the next integer value above, so easily can be calculated.

6.4.4 times

We do not include the property times, which is just a calculation operator.

6.4.5 predecessor

We do not include the property predecessor. This predecessor is just the previous integer value below, so easily can be calculated.

6.4.6 negated

We do not include the property negated. The negated just returns the opposite sign of the current integer value.

6.4.7 isNegative

We do not include the property isNegative. This boolean result is derived from the current integer value.

6.4.8 nonNegative

We do not include the property nonNegative. This boolean result can just be derived by checking if the current integer value is zero or greater.
6.4.9 dividedBy

We do not include the property `dividedBy`, which is just a calculation operator.

6.4.10 remainder

We do not include the property `remainder`, which is just a calculation operator.

6.4.11 isOne

We do not include `isOne` because it can be derived by checking if the current integer value is one.

6.4.12 Properties inherited from HL7:QTZ

Please see the chapter on HL7:QTZ
Chapter 7

PQ

7.1 Physical Quantity

Physical Quantity (Figure 7.1) is used to store a real number value with a real physical unit to represent quantities such as 2.3 milligrams, 24 days, or 2 drops. It also allows the representation of greater than and less than some physical quantity. Moreover, PQ has the capability to represent equivalent translations of the Physical Quantity, with a different physical unit and an appropriately converted real number.
value.

The value in PQ always represents the normalized value and unit for use in the ECIS system. If a different original unit exists, it is represented in the translation section.\(^1\)

## 7.2 Properties

### Table 7.1: PQ Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>REAL</td>
<td>The magnitude of the quantity measured in terms of the unit</td>
</tr>
<tr>
<td>unit</td>
<td>CWENT(^2)</td>
<td>Unit of measure, e.g. mg, seconds</td>
</tr>
<tr>
<td>operator</td>
<td>CS</td>
<td>&gt;, &lt;, &gt;=, or &lt;=</td>
</tr>
<tr>
<td>translation</td>
<td>SET&lt;PQT&gt;</td>
<td>One or more translations of this PQ</td>
</tr>
</tbody>
</table>

### 7.2.1 value

The property value is the magnitude of the quantity measured in terms of the unit.

### 7.2.2 unit

The property Unit is a unit of measure, e.g. mg, seconds. Note that unit is of type CWENT which has no translations. The reason for this is that the PQ datatype itself has translations in the form of PQT, so any translated unit would just result in another translation within a PQT.

### 7.2.3 operator

The property operator is a coded value that will represent either >, <, >=, or <=.

### 7.2.4 translation

The property translation represents one or more translations of this PQ. Translations are of type PQT. For storage we only allow one PQT within the translation section of PQ. If this translation exists, it is always the original physical quantity.

\(^1\)For use cases see appendix chapter B.
and the representation in the upper section of the PQ is a normalized value taken from this original physical quantity.

### 7.3 Constraint Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>ST</td>
<td>A code for the normalized unit of measure</td>
</tr>
<tr>
<td>minInclusive</td>
<td>REAL</td>
<td>value must be greater than or equal</td>
</tr>
<tr>
<td>maxInclusive</td>
<td>REAL</td>
<td>value must be less than or equal</td>
</tr>
<tr>
<td>minExclusive</td>
<td>REAL</td>
<td>value must be greater than</td>
</tr>
<tr>
<td>maxExclusive</td>
<td>REAL</td>
<td>value must be less than</td>
</tr>
</tbody>
</table>

#### 7.3.1 normal

A code from the primary coding system, for a preferred unit of measure, which will be the required normalized value for this PQ.

#### 7.3.2 minInclusive

The property `minInclusive` represents a real number that `value` must be greater than or equal to.

#### 7.3.3 maxInclusive

The property `maxInclusive` is a real number that `value` must be less than or equal to.

#### 7.3.4 minExclusive

The property `minExclusive` is a real number that `value` must be greater than.

#### 7.3.5 maxExclusive

The property `maxExclusive` is a real number that `value` must be less than.
One major difference between PQ and HL7:PQ is that HL7 requires that unit be a UCUM code, but we have decided to code unit using the primary coding system.³ The UCUM system enables one to create physical quantities that are not constrained to any particular unit, but by using UCUM codes you could never create inexact physical quantities such as 2 drops. Even though our unit is not represented by a UCUM code, the code used should, when applicable, be mapped to the appropriate UCUM code in the vocabulary server.

Another difference is that HL7 uses IVL<PQ> to represent greater than and less than. Instead, we handle this directly in PQ with the operator property. The HL7 solution allows the representation of concrete ranges as well as the opened ended range of greater than and less than, so we use IVL<PQ> when that is appropriate.

³The primary coding system is the single coding system that has been selected for the storage of Clinical Elements.
7.4.1 value

We include the property value.

7.4.2 operator

The property operator is not part of the HL7 abstract definition. While HL7 would use an IVL<PQ> to represent “greater than” or “less than” a real number, we will also use a coded value within PQ to represent this.

7.4.3 unit

We do include the property unit but we have changed the type to our CWENT because we want to be able to store originalText along with the code. In addition, HL7 uses UCUM codes and we are using codes from our primary coding system. We do not want to internally use multiple coding systems, and in addition UCUM doesn’t support concepts like “drops” since they are not truly quantitative.
7.4.4 equal
We do not include the property equal because it is just a comparison operator.

7.4.5 lessOrEqual
We do not include the property lessOrEqual because it is just a comparison operator.

7.4.6 compares
We do not include the property compares because it is just a comparison operator.

7.4.7 canonical
We do not include the property canonical. The reason being that we always put the canonical form in our PQ. Any value we receive that is not in the canonical or what we call “normal” form is normalized, and the original is stored in translation.

7.4.8 codingRationale
We do not include the property codingRationale.\(^4\)

7.4.9 translation
We do include the property translation but have changed the type from HL7:DSET<PQR> to our SET<PQT>.

7.4.10 negated
We do not include the property negated because it is derived as a negation of the current value in the PQ.

7.4.11 times
We do not include the property times because it is just a calculation operator.

7.4.12 dividedBy
We do not include the property dividedBy because it is just a calculation operator.

\(^4\)This needs further discussion with the modeling team.
7.4.13 inverted
We do not include the property inverted because it is just a calculation operator.

7.4.14 power
We do not include the property power because it is just a calculation operator.

7.4.15 isOne
We do not include the property isOne because it is just a boolean check to see if the current value in the PQ is one or not.
Chapter 8

REAL

8.1 Real Number

Real Number (Figure 8.1) is used to store real number values such as 1.25678, 3, or 0. REAL contains the property operator to represent concepts such as “greater than” or “less than”.

8.2 Properties

8.2.1 value

The property value represents the value of the real number, such as 1.25678, 3.00, or 0. Precision must be maintained by the implementation.
8.3. CONSTRAINT PROPERTIES

Table 8.1: REAL Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>real</td>
<td>Value of the real number</td>
</tr>
<tr>
<td>operator</td>
<td>CS</td>
<td>&gt;, &lt;, &gt;=, or &lt;=</td>
</tr>
</tbody>
</table>

8.2.2 operator

The property operator provides a coded value that will represent either >, <, >=, or <=.

8.3 Constraint Properties

Table 8.2: REAL Constraint Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minInclusive</td>
<td>REAL</td>
<td>value must be greater than or equal</td>
</tr>
<tr>
<td>maxInclusive</td>
<td>REAL</td>
<td>value must be less than or equal</td>
</tr>
<tr>
<td>minExclusive</td>
<td>REAL</td>
<td>value must be greater than</td>
</tr>
<tr>
<td>maxExclusive</td>
<td>REAL</td>
<td>value must be less than</td>
</tr>
</tbody>
</table>

8.3.1 minInclusive

The property minInclusive is a real number that value must be greater than or equal to.

8.3.2 maxInclusive

The property maxInclusive is a real number that value must be less than or equal to.

8.3.3 minExclusive

The property minExclusive is a real number that value must be greater than.
8.3.4 maxExclusive

The property maxExclusive is a real number that value must be less than.
8.4. HL7 COMPARISON

CHAPTER 8. REAL

8.4 HL7 Comparison

Figure 8.2: HL7:REAL Declaration

```plaintext
type RealNumber alias REAL specializes QTZ {
    REAL negated;
    REAL times(REAL x);
    REAL dividedBy(REAL x);
    REAL inverted;
    BL isOne;
    REAL power(REAL x);
    literal ST.SIMPLE;
    INT precision;
    demotion INT;
    promotion REAL (INT x);
    promotion PQ;
};
```

Figure 8.3: REAL to HL7:REAL Comparison

While HL7 would use an `IVL<REAL>` to represent “greater than” or “less than” some real number, we use a coded operator within `REAL`. For simplicity, we will not implement HL7’s `IVL<REAL>` in our system.

8.4.1 value

The property `value` is not part of the HL7 abstract definition. The HL7 specification leaves it up to implementation where to store the numeric value of a real number.
CHAPTER 8. REAL

8.4. HL7 COMPARISON

We have chosen to use value which is the same choice HL7 made in their version 3 XML ITS.

8.4.2 operator

The property operator is not part of the HL7 abstract definition. While HL7 would use an IVL<REAL> to represent “greater than” or “less than” a real number, we use a coded operator within REAL. For simplicity, we will not implement HL7’s IVL<REAL> in our system.

8.4.3 negated

We do not include the property negated. The negated just returns the opposite sign of the current real number value.

8.4.4 times

We do not include the property times, which is just a calculation operator.

8.4.5 dividedBy

We do not include the property dividedBy, which is just a calculation operator.

8.4.6 inverted

We do not include the property inverted, which is just a calculation operator.

8.4.7 isOne

We do not include isOne because it can be derived by checking if the current real value is one.

8.4.8 power

We do not include the property power, which is just a calculation operator.

8.4.9 precision

We do not include the property precision, which is the number of significant digits of the decimal representation. This can be derived from our value property.
8.4.10 Properties inherited from HL7:QTZ

Please see the chapter on HL7:QTZ
Chapter 9

ST

9.1 Character String

Figure 9.1: Character String

Character String (Figure 9.1) is used to store text/plain data.

9.2 Properties

Table 9.1: ST Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>string</td>
<td>String of Characters</td>
</tr>
</tbody>
</table>
9.2.1 value

The property value is used to hold the string of characters. When ST is used in some implementation, the explicit representation of value is not required, but only need be implied. ST is frequently used in the internal representation of other datatypes, and it would be a hinderance to expect an implementation to have a value field internal to each usage of ST. For example, CWE (Figure 3.1) has the property code which is declared to be of type ST, and an implementation would probably access the value of code through something like cwe.code rather than cwe.code.value.

9.3 Constraint Properties

Table 9.2: ST Constraint Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>INT</td>
<td>An integer value for the maximum string length</td>
</tr>
<tr>
<td>min</td>
<td>INT</td>
<td>An integer value for the minimum string length</td>
</tr>
</tbody>
</table>

9.3.1 max

The property max is an integer value for the maximum string length

9.3.2 min

The property min is an integer value for the minimum string length
9.4 HL7 Comparison

In the HL7 hierarchy, **HL7:ST** is a subtype of **HL7:ED** with `mediaType` fixed to `text/plain` data. We have not physically implemented this as a subtype, but it is implied.

### 9.4.1 value

We have added the property `value` as a direct replacement for the property `data` which is inherited in HL7 from **HL7:ED**.¹

### 9.4.2 data

We do not include the property `data`, and instead put the string characters in the property `value`.

¹Need to discuss with modeling team. Should we change this property to `data`. Although our datatypes all inherit from `ANY`, and we have full control over the naming of properties, should we still retain the spirit of the HL7 inheritance tree in our naming conventions. One ugly result would surface in our Clinical Element path notation, with “data.st.data”, but then again we already have that problem with “data.ed.data”. Maybe we should change our **ED** datatype as well, because we use `value` throughout most datatypes.
9.4.3 mediaType

We do not include the property mediaType, as this is fixed to “text/plain”.

9.4.4 charset

We do not include the property charset.³

9.4.5 language

We do not include the property language.⁴

9.4.6 length

We do not include the property length. This can be calculated from the value.

9.4.7 headCharacter

We do not include the property headCharacter. This can be calculated from the value.

9.4.8 tailString

We do not include the property tailString. Not sure if this can be derived from the value.⁵

9.4.9 translation

We do not include the property translation.⁶ nical element model.⁷

9.4.10 Other Properties inherited from HL7:ED

Please see the chapter on HL7:ED for additional properties not stated here. Also note that since the string characters are fixed to “text/plain” and can not be compressed, that the properties compression, reference, integrityCheck, algorithm, and thumbnail are not needed in this context.

³Never considered in modeling discussions, needs discussion
⁴Never considered in modeling discussions, needs discussion
⁵Discuss with modeling team for clarification.
⁶Never considered in modeling discussions, needs discussion
⁷Need to discuss with modeling team
Chapter 10

TS

10.1 Point in Time

Point in Time (Figure 10.1) is used to store a point in time. Operator was added by IHC to represent concepts such as “greater than” or “less than”.

The format recommended for value is that specified by constrained ISO 8601 (a simpler ISO 8601 variant), which is defined in ISO 8824 (ASN.1) under clause 32 (generalized time). The syntax is YYYYMMDDHHMMSS.UUUU[+|-ZZzz]” For example, the literal form for April 1, 2000, 3:15 and 20.34 Eastern Standard Time is “20000401031520.34-0500”. Note the dash is a minus sign and begins the time zone.
10.2. PROPERTIES

It should be noted that the XML ITS of the HL7 version 3 specification has a different format and follows the uses the literal form defined by the XML Schema Part 2: Datatypes specification available from the World Wide Web Consortium.

To represent $\textgreater$ or $\textless$ a particular point in time, ECIS uses TS with a coded operator, whereas HL7 would use another datatype – IVL<TS>. For simplicity, we aren’t implementing IVL<TS>.

If value can support specification of time zone, we don’t need an additional timeZone component. If value cannot specify time zone, we need the timeZone component.

### 10.2 Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>ST</td>
<td>UTC representation of the point in time</td>
</tr>
<tr>
<td>operator</td>
<td>CS</td>
<td>$&gt;$, $&lt;$, $\geq$, or $\leq$</td>
</tr>
<tr>
<td>timeZone</td>
<td>CS</td>
<td>Time zone of the point in time</td>
</tr>
</tbody>
</table>

#### 10.2.1 value

The property *value* is often represented as a calendar expression, which is a text string. The constrained ISO 8601 (simper ISO 8601 variant), which is defined in ISO 8824 (ASN.1) under clause 32 (generalized time) is well established in use in HL7. The simpler ISO 8601 variant has no decorating dashes, colons and no “T” between the date and time. The syntax is YYYYMMDDHHMMSS.UUUU[±ZZzz].

#### 10.2.2 operator

The property *operator* is a coded value that will represent either $>$, $<$, $\geq$, or $\leq$.

#### 10.2.3 timeZone

The property *timeZone* is used to store time zone information. The value for timeZone is a code from the time zone code domain in our own code system. Because we would convert all the point in time data to a standard time zone like GMT before
storing them in our clinical data repository, and we cannot guarantee that we will always have geographic information stored. We need local time zone information to allow us to convert the time back to the local time the data was collected. One of the clinical examples where being able to convert back to local time is very useful is, when a patient has seizure, we not only want to know the time the seizure was happened, but also would like to know this patient’s seizures usually happen in the morning. Without time zone and geographic location information, we lose the ability to know the seizure happens in the morning.
10.3 HL7 Comparison

Figure 10.2: HL7:TS Declaration

```plaintext
type PointInTime alias TS specializes QTY {
  PQ  offset;
  CS  calendar;
  INT precision;
  PQ  timezone;
  BL  equal(ANY x);
  TS  plus(PQ x);
  TS  minus(PQ x);
  PQ  minus(TS x);
  literal ST.SIMPLE;
};
```

Figure 10.3: TS to HL7:TS Comparison

We have added the coded `operator` property to represent “greater than” or “less than” a Point in Time. For concrete intervals of time, we will define two separate Clinical Element models, one for each end point.

We have added a coded `timeZone` property because a UTC requires a comparison with a geographic location to retrieve the true local time. This has to do with the fact that not all locations follow daylight savings time.
10.3.1 value
Like the numeric datatypes, HL7 doesn’t have a field for the value of HL7:TS and leaves this to the implementation. As in our other datatypes, we have chosen to use the property value.

10.3.2 operator
As in our numeric datatypes, we have added the property operator to signify “greater than” and “less than”.

10.3.3 offset
We do not include the property offset which is defined as the elapsed time since any constant epoch, measured as a physical quantity in the dimension of time. We do not yet have a use case where the normal representation does not suffice.

10.3.4 calendar
We do not include the property calendar and default this value to the Gregorian calendar. This is the default calendar used by HL7.

10.3.5 precision
We do not include the property precision which is the number of significant digits of the calendar expression representation. The precision can be calculated from the current value.

10.3.6 timezone
We do include the property timezone, but change the type to a coded CS rather than use an HL7:PQ.

10.3.7 equal
We do not include the property equal because it is a comparison operator.

10.3.8 plus
We do not include the property equal because it is a calculation operator.
10.3.9 **minus**

We do not include the property *equal* because it is a calculation operator.
Chapter 11

ED

11.1 Encapsulated Data

Encapsulated Data (Figure 11.1) is used to convey any data from a plain character string, formatted text, to multimedia binary data. It may also contain formatted data from another standard such as XML.

11.2 Properties

Table 11.1: ED Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>BIN</td>
<td>Binary data</td>
</tr>
<tr>
<td>mediaType</td>
<td>CS</td>
<td>Type of binary data</td>
</tr>
<tr>
<td>language</td>
<td>CS</td>
<td>Human Language</td>
</tr>
<tr>
<td>compression</td>
<td>CS</td>
<td>Compression Algorithm of binary data</td>
</tr>
<tr>
<td>integrityCheck</td>
<td>CS</td>
<td>Value generated by Integrity Check Algorithm</td>
</tr>
<tr>
<td>integrityCheckAlgorithm</td>
<td>CS</td>
<td>Algorithm used to generate value</td>
</tr>
<tr>
<td>reference</td>
<td>URL</td>
<td>Pointer to externally stored binary data</td>
</tr>
<tr>
<td>thumbnail</td>
<td>EDNT</td>
<td>Abbreviated rendition of binary data</td>
</tr>
</tbody>
</table>
Figure 11.1: Encapsulated Data
11.2.1 data

The property data represents raw binary data. We have decided to store large binary data, such as X-Rays, in an external data store, but this data will still be present in the data property over the wire. The property reference will contain the pointer to the data in the external data store. For smaller binary data, it will actually be stored in the data property, and reference will be null. The services should function in a seamless manner, so that users of the service need not be aware whether the data was stored inline or as part of the external data store.

11.2.2 mediaType

The property mediaType represents the type of the encapsulated data and identifies a method to interpret or render the data.

11.2.3 language

The property language represents character based information specifying the human language of the text.

11.2.4 compression

The property compression indicates whether the raw byte data is compressed, and what compression algorithm was used.

11.2.5 integrityCheck

The property integrityCheck is a short binary value representing a cryptographically strong checksum that is calculated over the binary data. The purpose of this property, when communicated with a reference is for anyone to validate later whether the reference still resolves to the same data that the reference resolved to when the encapsulated data value with reference was created.

The integrity check is calculated over the raw binary data that is contained in the data component, or that is accessible through the reference. No transformations are made before the integrity check is calculated. If the data is compressed, the integrity check is calculated over the compressed data.

11.2.6 integrityCheckAlgorithm

The property integrityCheckAlgorithm specifies the algorithm used to compute the integrityCheck value. The cryptographically strong checksum algorithm Secure Hash Algorithm-1 (SHA-1) is currently the industry standard. It superseded
the MD5 algorithm several years ago, when certain flaws in the security of MD5 were discovered. Currently the SHA-1 hash algorithm is the default choice for the integrity check algorithm. Note that SHA-256 is also entering widespread usage.

11.2.7 reference

The property reference is a URL which will resolve to precisely the same binary data that could as well have been provided as inline data. This serves as the pointer to the external data source where the data of ED is actually stored.

An IHE Profile will be used to define the format of URL. The IHE profile basically specifies a common URL format to use and calls for the use of a UID or OID to reference the thing on the other end. The consistent URL format makes it possible to rewrite a stored URL to hit a different server using the same query at a later time.

11.2.8 thumbnail

The property thumbnail is an abbreviated rendition of the full data. A thumbnail requires significantly fewer resources than the full data, while still maintaining some distinctive similarity with the full data. A thumbnail is typically used with by-reference encapsulated data. It allows a user to select data more efficiently before actually downloading through the reference.
11.3 HL7 Comparison

Figure 11.2: HL7:ED Declaration

```plaintext
    type EncapsulatedData alias ED specializes ANY {
        BIN data;
        CS mediaType;
        CS charset;
        CS language;
        CS compression;
        TEL.URL reference;
        BIN integrityCheck;
        CS integrityCheckAlgorithm;
        ST description;
        ED thumbnail;
        DSET<ED> translation;
        INT length;
        ED subPart(INT start, INT end);
        BL equal(ANY x);
    };
```

HL7 uses **HL7:TEL** for the *reference* property, which we have substituted with **URL**. Another small change we make to avoid recursion in the *thumbnail* property, is to assign **EDNT** rather than **ED**.

11.3.1 data

We include the property *data*.¹

11.3.2 mediaType

We include the property *mediaType*.

11.3.3 charset

We do not include the property *charset* for character based encoding.²

¹ Need to discuss with modeling team. One ugly result is the problem with the path statement “data.ed.data”. Should we change *data* to *value* which is used throughout most of our datatypes. Note that in *ST* we are using *value*

² Need to discuss with modeling team, I believe this was skipped over.
11.3.4 language

We include the property *language* which is used for character based data.

11.3.5 compression

We include the property *compression* which is used to identify the compression algorithm used.

11.3.6 reference

We include the property *reference* to identify external references.

11.3.7 integrityCheck

We include the property *integrityCheck* but currently have this defined as a type CS instead of a BIN. I think this may be an error.³

³Discuss with modeling team.
11.3.8 **integrityCheckAlgorithm**

We include the property *integrityCheckAlgorithm*.

11.3.9 **description**

We do not include the property *description*. This property is intended to be a short description of the media contained in case the media cannot be presented.\(^4\)

11.3.10 **thumbnail**

We include the property *thumbnail*, but change the type from *HL7:ED* to our *EDNT* which does not allow a thumbnail. This prevents a recursive nesting of thumbnails.

11.3.11 **translation**

We do not include the property *translation*, which allows for alternate renditions of the same content translated into a different language or a different *mediaType*.\(^5\)

11.3.12 **length**

We do not include the property *length*, as the length of the binary data can be calculated by the implementation.

11.3.13 **subpart**

We do not include the property *subpart*, as this is an operator on the stored data.

11.3.14 **equal**

We do not include the property *equal*, as this is a comparison operator on the stored data.

---

\(^4\)Discuss with modeling team.

\(^5\)Discuss with modeling team. Do we want translations?
Chapter 12

IVLPQ

12.1 Interval Physical Quantity

Interval Physical Quantity (Figure 12.1) is used to represent an interval of Physical Quantities. We will only use IVLPQ for closed intervals, because open intervals can be handled by our PQ using the operator property.

12.2 Properties

12.2.1 low

The property low represents the low value of the interval
Table 12.1: IVLPQ Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>PQ</td>
<td>The low end of the interval</td>
</tr>
<tr>
<td>high</td>
<td>PQ</td>
<td>The high end of the interval</td>
</tr>
</tbody>
</table>

12.2.2 high

The property high represents the high of the interval
12.3 HL7 Comparison

The HL7 datatype IVL<PQ> has been restructured to no longer include a high and low HL7:PQ which resulted in redundant units of measure. Now the unit of measure has been promoted, and the interval is instead defined by real numbers.\footnote{Discuss with modeling team, and suggest a remodeling}

\begin{figure}[h]
\begin{center}
\begin{minipage}{0.8\textwidth}
\begin{verbatim}
type Interval<T> alias IVL<T> specializes QSET<T> { 
    T low;
    BL lowClosed;
    T high;
    BL highClosed;
    QTY width;
    T center;
    T any;
    IVL<T> hull;
    IVL<T> hull(IVL<T> x);
    literal ST.SIMPLE;
    promotion IVL<T> (T x);
    demotion T;
};

type Interval<PQ> alias IVL<PQ> {
    IVL<REAL> value;
    CS unit;
};
\end{verbatim}
\end{minipage}
\end{center}
\caption{HL7 IVL<PQ> declaration}
\end{figure}

Figure 12.3: IVLPQ to HL7:IVL<PQ> Comparison

We have removed the properties width and center from HL7:IVL<PQ> because both are calculated values. If it is considered important for us to index on
these properties, we could add them back at a later time.

We have also removed the properties lowClosed and highClosed because they will always be true in our case.\(^2\)

### 12.3.1 low

We include the property low, but it is currently a PQ instead of a REAL.

### 12.3.2 lowClosed

We do not include the property lowClosed, and always assume lowClosed is true, which means the lower point is included in the interval.\(^3\)

### 12.3.3 high

We include the property high, but it is currently a PQ instead of a REAL.

### 12.3.4 highClosed

We do not include the property highClosed, and always assume highClosed is true, which means the higher point is included in the interval.\(^4\)

### 12.3.5 width

We do not include the property width, as this is a calculation based on the high and low values.

### 12.3.6 center

We do not include the property center, as this is a calculation based on the high and low values.

\(^2\)At least I believe this to be true, but if these properties are actually meant to define inclusive and exclusive end points... then I could be wrong. And on that note, how do you define an inclusive or exclusive end point that is not in the literal HL7 form?

\(^3\)Discuss with modeling team.

\(^4\)Discuss with modeling team.
12.3.7  any

We do not include the property any, which specifies that nothing is known about the interval except that some particular value lies within the interval.\(^5\)

12.3.8  hull

We do not include the property hull, because this a calculation.

\(^5\)We can do the same with a qualifier or this can be known by the context in a model, discuss with modeling team.
Chapter 13

RTOPQ

13.1 Ratio Physical Quantity

Figure 13.1: Ratio Physical Quantity

Ratio Physical Quantity (Figure 13.1) is used to represent a quantity constructed through the division of a numerator quantity with a denominator quantity. The RTO data type supports titers (e.g. \( \text{\AA} \text{\textit{IJ}}:128\text{\AA} \text{\textit{I}} \)) and other quantities produced by laboratories that truly represent ratios. Blood pressure measurements (e.g. \( \text{\AA} \text{\textit{IJ}}120/60\text{\AA} \text{\textit{I}} \)) are not ratios.
13.2 Properties

13.2.1 numerator

The property \textit{numerator} stands for the numerator of the ratio.

13.2.2 denominator

The property \textit{denominator} is the denominator of the ratio.
13.3 HL7 Comparison

Figure 13.2: HL7 RTO declaration

```plaintext
1 type Ratio<QTZ N, QTZ D> alias RTO specializes QTY {
2     N    numerator;
3     D    denominator;
4     literal ST.SIMPLE;
5     demotion REAL;
6     demotion PQ;
7 }
```

Figure 13.3: RTOPQ to HL7:RTO<PQ> Comparison

There are no structural differences between our RTOPQ and HL7:RTO<PQ> at least at the level these are defined. But it should be noted that the PQ and HL7:PQ which are internally referenced by these two types are different.

13.3.1 numerator

We include the property numerator.

13.3.2 denominator

We include the property denominator.
Part II

Datatypes for Internal Use
Chapter 14

PQT

14.1 Physical Quantity Translation

Physical Quantity Translation (Figure 14.1) is used to store a translation of a real number value with a real physical unit to represent quantities such as 2.3 milligrams, 24 days, or 2 drops. The translation could simply be a different repre-
14.2 Properties

Table 14.1: PQT Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>REAL</td>
<td>The magnitude of the quantity measured in terms of the unit</td>
</tr>
<tr>
<td>unit</td>
<td>ST</td>
<td>A code for unit of measure, e.g. mg, seconds</td>
</tr>
<tr>
<td>codeSystem</td>
<td>CS</td>
<td>The coding system from which unit is defined</td>
</tr>
<tr>
<td>codeSystemVersion</td>
<td>ST</td>
<td>The version of the coding system</td>
</tr>
<tr>
<td>operator</td>
<td>CS</td>
<td>&gt;, &lt;=, &gt;=, or &lt;=</td>
</tr>
</tbody>
</table>

14.2.1 value

The property value is the magnitude of the quantity measured in terms of the unit

14.2.2 unit

The property unit is a code for unit of measure, e.g. mg, seconds
14.2.3 codeSystem

The property codeSystem is code for the coding system from which the value in unit is defined.

14.2.4 codeSystemVersion

The version of the coding system

14.2.5 operator

The property operator is a coded value that will represent either >, <, >=, or <=.
14.3 HL7 Comparison

Figure 14.2: HL7:PQR declaration

```java
flavor CodedValue alias CV constrains CD;

invariant (CV x) where x.nonNull {
  x.translation.isEmpty;
  x.source.isNull;
};

protected type PhysicalQuantityRepresentation alias PQR specializes CV {
  REAL value;
};
```

Figure 14.3: PQT to HL7:PQR Comparison

HL7 derived their HL7:PQR from the HL7:CD datatype and then added the
property value. For this reason, all the data regarding the unit of measure goes into the code related fields from **HL7:CD**. This is a little confusing to users, because the unit of measure is now in a property called code. Because of this confusion, we renamed the code property to unit, but we left the confusion of codeSystem and codeSystemVersion which now are details regarding the code within the unit property.

Just as in PQ, we represent “greater than” and “less than” with the operator property.

### 14.3.1 value

We include the property value.

### 14.3.2 operator

The property operator is not part of the HL7 abstract definition. While HL7 would use an IVL<PQ> to represent “greater than” or “less than” a real number, we will also use a coded value within PQT to represent this.

### 14.3.3 unit

We do include the property unit but we have changed the type to our CWENT because we want to be able to store originalText along with the code. In addition, HL7 uses UCUM codes and we are using codes from our primary coding system. We do not want to internally use multiple coding systems, and in addition UCUM doesn’t support concepts likes “drops” since they are not truly quantitative.

### 14.3.4 code

We do not include the property code. We have renamed this property unit.

### 14.3.5 displayName

We do not include displayName. We do face some danger here if the code came from an external system and the unit code is obtuse. We could put something readable into originalText.¹

### 14.3.6 codeSystem

We do include the property codeSystem.

¹This needs further analysis with the modeling team.
14.3.7 codeSystemName

We do not include the property codeSystemName because this is just a representation of the codeSystem and can be provided by the vocabulary server.

14.3.8 codeSystemVersion

We do include codeSystemVersion.

14.3.9 valueSet

We do not currently include the valueSet, which specifies the value set that applied when this PQR was created.\(^2\)

14.3.10 valueSetVersion

We do not currently include the valueSetVersion, which specifies the value set version that applied when this PQR was created. Even if we decided to add the valueSet property, I believe the version could be handled by the vocabulary server.

14.3.11 originalText

We include the property originalText.

14.3.12 codingRationale

We do not include the property codingRationale. We need to examine use cases and come to a conclusion on this.\(^3\)

14.3.13 isCompositional

We do not include the property isCompositional. This can be derived from the vocabulary server.

14.3.14 equal

We do not include the property equal. This is a comparison operation that can be handled by the vocabulary server.

\(^2\)Need to email Gunther about this. HL7 doesn’t remove this from CD, but they don’t discuss it.

\(^3\)Tom brought up a good point, that we are only using part of the functionality that codingRationale provides, and we have not discussed the other aspects.
14.3.15 **implies**

We do not include the property *implies*. This is a comparison operation that can be handled by the vocabulary server.

14.3.16 **Properties inherited from HL7:ANY**

Please see the chapter on **HL7:ANY**
15.1 Coded With Exceptions - Translation

Coded With Exceptions - Translation (Figure 15.1) is used internally\(^1\) within the CWE datatype, and is used to store translations of the primary code. Multiple CETs may be carried by any one CWE instance.

\(^1\)It is important to note that CET can not be directly used within a Clinical Element model, but only within a CWE datatype.
One example of its use would be if a code from an external coding system came over an interface, this external code would be placed in a CET datatype, which would then be stored in the translation section of a CWE, with the a code from the primary coding system going in the top section of the CWE datatype.

15.2 Properties

Table 15.1: CET Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>ST</td>
<td>A code for a concept defined in any code system.</td>
</tr>
<tr>
<td>originalText</td>
<td>ST</td>
<td>The textual representation of the code</td>
</tr>
<tr>
<td>codeSystem</td>
<td>CS</td>
<td>The coding system from which code is defined</td>
</tr>
<tr>
<td>codeSystemVersion</td>
<td>ST</td>
<td>The version of the coding system</td>
</tr>
</tbody>
</table>

15.2.1 code

The property code is the code for a concept defined in any code system; this code system is specified in codeSystem.

15.2.2 originalText

The property originalText is a textual representation of the code that was presented to the user, or the representation that came over the interface.

15.2.3 codeSystem

The property codeSystem represents a coding system from which the value in code is defined.

15.2.4 codeSystemVersion

The property CodeSystemVersion represents a version of the coding system.
15.3 HL7 Comparison

We have evolved CET from the HL7:CD datatype. In doing so, we stripped out various properties we did not need, as noted in CWE.

HL7:CD translation has recursive translations. By defining CET, which does not itself contain a translation component, we avoid nested translations.

HL7:CD defines originalText as an HL7:ED datatype. We decided that the complexity of the HL7:ED datatype was not worth the cost, so we defined originalText to be an ST. If a translation contains a file that requires the complexity of the HL7:ED datatype, then that file would be handled at the CWE level.

15.3.1 code

We include the property code.

15.3.2 displayName

We do not include displayName. We do face some danger here if the code came from an external system and the unit code is obtuse. We could put something readable into originalText.²

²This needs further analysis with the modeling team.

---

Figure 15.2: HL7 CD declaration

```plaintext
type ConceptDescriptor alias CD specializes ANY {
  ST.SIMPLE code;
  ST displayName;
  UID codeSystem;
  ST.NT codeSystemName;
  ST.SIMPLE codeSystemVersion;
  UID valueSet;
  TS.DATETIME.FULL valueSetVersion;
  ED.TEXT originalText;
  SET<CS> codingRationale;
  DSET<CD> translation;
  CD source;
  BL isCompositional;
  BL equal(ANY x);
  BL implies(CD x);
};
```
15.3.3 codeSystem

We do include the property codeSystem.

15.3.4 codeSystemName

We do not include the property codeSystemName because this is just a representation of the codeSystem and can be provided by the vocabulary server.

15.3.5 codeSystemVersion

We do include codeSystemVersion.
15.3.6 valueSet
We do not currently include the valueSet, which specifies the value set that applied when this CD was created.

15.3.7 valueSetVersion
We do not currently include the valueSetVersion, which specifies the value set version that applied when this CD was created. Even if we decided to add the valueSet property, I believe the version could be handled by the vocabulary server.

15.3.8 originalText
We include the property originalText.

15.3.9 codingRationale
We do not include the property codingRationale. We need to examine use cases and come to a conclusion on this.\(^3\)

15.3.10 isCompositional
We do not include the property isCompositional. This can be derived from the vocabulary server.

15.3.11 equal
We do not include the property equal. This is a comparison operation that can be handled by the vocabulary server.

15.3.12 implies
We do not include the property implies. This is a comparison operation that can be handled by the vocabulary server.

15.3.13 Properties inherited from HL7:ANY
Please see the chapter on HL7:ANY

\(^3\)Tom brought up a good point, that we are only using part of the functionality that codingRationale provides, and we have not discussed the other aspects.
Coded No Exceptions (Figure 16.1) requires that a code always be present.

Due to the performance requirements of a permanent storage based system, we have chosen to only allow codes from the primary coding system in `code`, and alternate codes from other coding systems will be placed in `translation`. 
### 16.2 Properties

#### 16.2.1 code

The property `code` contains a code for a concept defined in the primary coding system.

#### 16.2.2 originalText

The property `originalText` is used for textual representation of the code that was presented to the user, or the representation that came over the interface.

#### 16.2.3 translation

The property `translation` is used to represent Zero to many translations of the code from any code system.

### 16.3 Constraint Properties

#### 16.3.1 domain

The property `domain` is a code for a domain of concepts defined in the primary code system.
16.4 HL7 Comparison

The Clinical Element CNE datatype is derived from the structure of the HL7:CD datatype, but is actually similar in function to HL7:CD with the CNE coding strength qualifier. See Figure 16.4 for a description of HL7:CD with it’s coding strength qualifiers.

16.4.1 code

We include the property code.

16.4.2 displayName

We do not include displayName due to the fact we only allow codes from our primary coding system in CWE, and a displayName can be generated at any time from the vocabulary server.

16.4.3 codeSystem

We do not include codeSystem because in CWE the codeSystem is defaulted to the primary coding system.
16.4. HL7 COMPARISON

Figure 16.3: CNE to HL7:CE Comparison

Figure 16.4: HL7 CE and Coding Strength Qualifiers

**CWE - Coded With Exceptions** This is a coding strength qualifier that is applied to a binding of a vocabulary domain with a CD. It signifies that a code outside the specified value set/code system may be used, or that free text may be used in lieu of a code.

**CNE - Coded, No Exceptions** This is another coding strength qualifier that is applied to a binding of a vocabulary domain with a CD. It signifies that a code outside the specified value set/code system is not permitted, nor is free text permitted in lieu of a code.

16.4.4 codeSystemName

We do not include codeSystemName because in CWE the codeSystemName is defaulted to the primary coding system.
16.4.5 codeSystemVersion

We do not include codeSystemVersion because in our primary coding system we don’t use versioning.¹

16.4.6 valueSet

We do not currently include the valueSet, which specifies the value set that applied when this CD was created.²

16.4.7 valueSetVersion

We do not currently include the valueSetVersion, which specifies the value set version that applied when this CD was created. Even if we decided to add the valueSet property, I believe the version could be handled by the vocabulary server.

16.4.8 originalText

We include the property originalText.

16.4.9 codingRationale

We do not include the property codingRationale. For better or worse, we assume it is original if there isn’t a translation.³

16.4.10 translation

We included the property translation.

16.4.11 source

We do not include the property source. This property identifies the translation from which this was translated. We have not discussed this issue, but it seems this is implicit due to the fact we are only allowing one translation in the storage form. The question remains whether or not this is important transactionally when we allow more than one translation.

¹I need to double check this.
²We have never discussed the need for this.
³Tom brought up a good point, that we are only using part of the functionality that codingRationale provides, and we have not discussed the other aspects.
16.4.12  isCompositional

We do not include the property isCompositional. This can be derived from the vocabulary server.

16.4.13  equal

We do not include the property equal. This is a comparison operation that can be handled by the vocabulary server.

16.4.14  implies

We do not include the property implies. This is a comparison operation that can be handled by the vocabulary server.

16.4.15  Properties inherited from HL7:ANY

Please see the chapter on HL7:ANY
Chapter 17

COT

17.1 Coded Ordinal - Translation

Figure 17.1: Coded Ordinal - Translation

Coded Ordinal - Translation (Figure 17.1) is used internally\(^1\) within the COT

\(^1\)It is important to note that COT can not be directly used within a Clinical Element model, but
17.2. PROPERTIES

datatype, and is used to store translations of the primary code. Multiple COTs may be carried by any one CO instance.

One example of it’s use would be if a code representing an ordered concept from an external coding system came over an interface, this external code would be placed in a COT datatype, which would then be stored in the translation section of a CO, with the a code from the primary coding system going in the top section of the CO datatype.

17.2 Properties

Table 17.1: CET Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>ST</td>
<td>A code for a concept defined in any code system.</td>
</tr>
<tr>
<td>originalText</td>
<td>ST</td>
<td>The textual representation of the code</td>
</tr>
<tr>
<td>codeSystem</td>
<td>CS</td>
<td>The coding system from which code is defined</td>
</tr>
<tr>
<td>codeSystemVersion</td>
<td>ST</td>
<td>The version of the coding system</td>
</tr>
<tr>
<td>value</td>
<td>REAL</td>
<td>A numeric representation of the code’s meaning</td>
</tr>
</tbody>
</table>

17.2.1 code

The property code is the code for a concept defined in any code system; this code system is specified in codeSystem.

17.2.2 originalText

The property originalText is a textual representation of the code that was presented to the user, or the representation that came over the interface.

17.2.3 codeSystem

The property codeSystem is a coding system from which code is defined only within a CO datatype.
CHAPTER 17. COT

17.2. PROPERTIES

17.2.4 codeSystemVersion

The property CodeSystemVersion is a version of the coding system.

17.2.5 value

The property value is a numeric representation of the code’s meaning\(^2\)

\(^2\)Previously called numericScore.
17.3  HL7 Comparison

Figure 17.2: HL7:CO Declaration

```plaintext
1 type CodedOrdinal alias CO specializes CD {
2     REAL value;
3     BL lessOrEqual(CO o);
4     BL lessThan(CO o);
5     BL greaterThan(CO o);
6     BL greaterOrEqual(CO o);
};
```

Figure 17.3: COT to HL7:CD Comparison

We have evolved COT from the HL7:CD datatype. In doing so, we stripped out various properties we did not need, as noted in CO.
**CHAPTER 17. COT**

17.3. **HL7 COMPARISON**

**HL7:CD** translation has recursive translations. By defining **COT**, which does not itself contain a translation component, we avoid nested translations.

**HL7:CD** defines *originalText* as an **HL7:ED** datatype. We decided that the complexity of the **HL7:ED** datatype was not worth the cost, so we defined *originalText* to be an **ST**. If a translation contains a file that requires the complexity of the **HL7:ED** datatype, then that file would be handled at the **CO** level.

17.3.1 **value**

We include the property *value*.³

17.3.2 **lessOrEqual**

We do not include the property *lessOrEqual* because it is a comparison operator.

17.3.3 **lessThan**

We do not include the property *lessThan* because it is a comparison operator.

17.3.4 **greaterThan**

We do not include the property *greaterThan* because it is a comparison operator.

17.3.5 **greaterOrEqual**

We do not include the property *greaterOrEqual* because it is a comparison operator.

17.3.6 **Properties inherited from HL7:CD**

Please see the chapter on **HL7:CD**

---

³We previously called this *numericScore*. 
Chapter 18

CS

18.1 Coded Simple

Coded Simple (Figure 18.1) simply contains the property code. This datatype is not allowed for use in Clinical Element models, but it is used internally in the definitions of datatypes. In each use case, the domain of values is fixed by the implementation to a particular domain.¹

Because the code property within each CS is defined as data type ST, which is String, some people may question the necessity of defining CS. The difference between CS and ST is that, for CS, the string value in the code property must come from a code system; you cannot simply put any string into CS. In the implementation, this means there will be a need to check whether the string value is a valid code from the specified domain in the primary coding system. The purpose

¹These domains are only implied by the meaning of each use of CS, and we need to formally define each of these domains
18.2 Properties

The property code is a code for a concept defined in the primary code system.
18.3 HL7 Comparison

Figure 18.2: HL7:CS declaration

```plaintext
1  type CodedSimpleValue alias CS specializes CV {
2     ST.SIMPLE code;
3     literal ST.SIMPLE;
4  };
```

Figure 18.3: CS to HL7:CS Comparison

Our use of CS is now consistent with the current HL7:CS. It should be noted that a previous HL7:CS contained the property originalText, but they have now removed this as we had.

18.3.1 code

We include the property code.

18.3.2 Properties inherited from HL7:ANY

Please see the chapter on HL7:ANY
Chapter 19

CWENT

19.1 Coded With Exceptions - No Translation

![Figure 19.1: Coded With Exceptions - No Translation]

Coded With Exceptions - No Translation (Figure 19.1) is identical to our datatype CWE except it does not allow translations of the primary code. It is only for internal use, and is not used in Clinical Element models.

19.2 Properties

19.2.1 code

The property code is a code for a concept defined in the primary code system.
19.3. CONSTRAINT PROPERTIES

Table 19.1: CWENT Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>ST</td>
<td>A code for a concept defined in the primary code system.</td>
</tr>
<tr>
<td>originalText</td>
<td>ST</td>
<td>The textual representation of the code</td>
</tr>
</tbody>
</table>

19.2.2 originalText

The property `originalText` is a textual representation of the code that was presented to the user, or the representation that came over the interface.

19.3 Constraint Properties

Table 19.2: CWE Constraint Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain</td>
<td>ST</td>
<td>A code for a domain of concepts</td>
</tr>
</tbody>
</table>

19.3.1 domain

The property `domain` represents code for a domain of concepts defined in the primary code system.
19.4 HL7 Comparison

Figure 19.2: HL7 CD declaration

```plaintext
type ConceptDescriptor alias CD specializes ANY {
  ST.SIMPLE code;
  ST displayName;
  UID codeSystem;
  ST.NT codeSystemName;
  ST.SIMPLE codeSystemVersion;
  UID valueSet;
  TS.DATETIME.FULL valueSetVersion;
  ED.TEXT originalText;
  SET<CS> codingRationale;
  DSET<CD> translation;
  CD source;
  BL isCompositional;
  BL equal(ANY x);
  BL implies(CD x);
};
```

Figure 19.3: CWENT to CWE Comparison

Comparison to HL7:CD is identical to the comparison of CWE with HL7:CD, with the additional restriction that we have removed the translation property.
19.4.1 code
We include the property code.

19.4.2 displayName
We do not include displayName due to the fact we only allow codes from our primary coding system in CWE, and a displayName can be generated at any time from the vocabulary server.

19.4.3 codeSystem
We do not include codeSystem because in CWE the codeSystem is defaulted to the primary coding system.

19.4.4 codeSystemName
We do not include codeSystemName because in CWE the codeSystemName is defaulted to the primary coding system.

19.4.5 codeSystemVersion
We do not include codeSystemVersion because in our primary coding system we don’t use versioning.\(^1\)

19.4.6 valueSet
We do not currently include the valueSet, which specifies the value set that applied when this CD was created.\(^2\)

19.4.7 valueSetVersion
We do not currently include the valueSetVersion, which specifies the value set version that applied when this CD was created. Even if we decided to add the valueSet property, I believe the version could be handled by the vocabulary server.

19.4.8 originalText
We include the property originalText.

\(^{1}\)I need to double check this.

\(^{2}\)We have never discussed the need for this.
19.4.9 codingRationale

We do not include the property codingRationale. For better or worse, we assume it is original if there isn’t a translation.\(^3\)

19.4.10 translation

We do not include the property translation, which is the purpose of CWENT.

19.4.11 source

We do not include the property source. This property identifies the translation from which this was translated. We have not discussed this issue, but it seems this is implicit due to the fact we are only allowing one translation in the storage form. The question remains whether or not this is important transactionally when we allow more than one translation.

19.4.12 isCompositional

We do not include the property isCompositional. This can be derived from the vocabulary server.

19.4.13 equal

We do not include the property equal. This is a comparison operation that can be handled by the vocabulary server.

19.4.14 implies

We do not include the property implies. This is a comparison operation that can be handled by the vocabulary server.

19.4.15 Properties inherited from HL7:ANY

Please see the chapter on HL7:ANY.

\(^3\)Tom brought up a good point, that we are only using part of the functionality that codingRationale provides, and we have not discussed the other aspects.
Chapter 20

EDNT

20.1 Encapsulated Data - No Thumbnail

Encapsulated Data No Thumbnail (Figure 20.1) is used to convey any data from a plain character string, formatted text, to multimedia binary data. It may also contain formatted data from another standard such as XML.

EDNT is not for use in Clinical Element models. It is only used internally in the ED data type, to represent the thumbnail property, and avoid recursion of the thumbnail property.

20.2 Properties

Table 20.1: EDNT Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>BIN</td>
<td>Binary data</td>
</tr>
<tr>
<td>mediaType</td>
<td>CS</td>
<td>Type of binary data</td>
</tr>
<tr>
<td>language</td>
<td>CS</td>
<td>Human Language</td>
</tr>
<tr>
<td>compression</td>
<td>CS</td>
<td>Compression Algorithm of binary data</td>
</tr>
<tr>
<td>integrityCheck</td>
<td>CS</td>
<td>Value generated by Integrity Check Algorithm</td>
</tr>
<tr>
<td>integrityCheckAlgorithm</td>
<td>CS</td>
<td>Algorithm used to generate value</td>
</tr>
<tr>
<td>reference</td>
<td>URL</td>
<td>Pointer to externally stored binary data</td>
</tr>
</tbody>
</table>
20.2. PROPERTIES

The property *data* is raw binary data. We have decided to store large binary data, such as X-Rays, in an external data store, but this data will still be present in the *data* property over the wire. The property *reference* will contain the pointer to the data in the external data store. For smaller binary data, it will actually be stored in the *data* property, and *reference* will be null. The services should function in a seamless manner, so that users of the service need not be aware whether the data was stored inline or as part of the external data store.
20.2.2 mediaType

The property mediaType identifies the type of the encapsulated data and identifies a method to interpret or render the data.

20.2.3 language

The property language is character based information specifying the human language of the text.

20.2.4 compression

The property compression indicates whether the raw byte data is compressed, and what compression algorithm was used.

20.2.5 integrityCheck

The property integrityCheck is a short binary value representing a cryptographically strong checksum that is calculated over the binary data. The purpose of this property, when communicated with a reference is for anyone to validate later whether the reference still resolves to the same data that the reference resolved to when the encapsulated data value with reference was created.

The integrity check is calculated over the raw binary data that is contained in the data component, or that is accessible through the reference. No transformations are made before the integrity check is calculated. If the data is compressed, the integrity check is calculated over the compressed data.

20.2.6 integrityCheckAlgorithm

The property integrityCheckAlgorithm specifies the algorithm used to compute the integrityCheck value. The cryptographically strong checksum algorithm Secure Hash Algorithm-1 (SHA-1) is currently the industry standard. It superseded the MD5 algorithm several years ago, when certain flaws in the security of MD5 were discovered. Currently the SHA-1 hash algorithm is the default choice for the integrity check algorithm. Note that SHA-256 is also entering widespread usage.

20.2.7 reference

The property reference is a URL which will resolve to precisely the same binary data that could as well have been provided as online data. This serves as the pointer to the external data source where the data of ED is actually stored.
An IHE Profile will be used to define the format of URL. The IHE profile basically specifies a common URL format to use and calls for the use of a UID or OID to reference the thing on the other end. The consistent URL format makes it possible to rewrite a stored URL to hit a different server using the same query at a later time.
### 20.3 HL7 Comparison

Figure 20.2: HL7:ED Declaration

```plaintext

type EncapsulatedData alias ED specializes ANY {
  BIN data;
  CS mediaType;
  CS charset;
  CS language;
  CS compression;
  TEL.URL reference;
  BIN integrityCheck;
  CS integrityCheckAlgorithm;
  ST description;
  ED thumbnail;
  DSET<ED> translation;
  INT length;
  ED subPart(INT start, INT end);
  BL equal(ANY x);
};
```

#### 20.3.1 data

We include the property `data`\(^1\)

#### 20.3.2 mediaType

We include the property `mediaType`.

#### 20.3.3 charset

We do not include the property `charset` for character based encoding.\(^2\)

#### 20.3.4 language

We include the property `language` which is used for character based data.

\(^1\)Need to discuss with modeling team. One ugly result is the problem with the path statement “data.ed.data”. Should we change `data` to `value` which is used throughout most of our datatypes. Note that in `ST` we are using `value`.

\(^2\)Need to discuss with modeling team, I believe this was skipped over.
20.3.5 compression

We include the property *compression* which is used to identify the compression algorithm used.

20.3.6 reference

We include the property *reference* to identify external references. HL7 uses **HL7:TEL** for the *reference* property, which we have substituted with **URL**.

20.3.7 integrityCheck

We include the property *integrityCheck* but currently have this defined as a type **CS** instead of a **BIN**. I think this may be an error.\(^3\)

\(^3\)Discuss with modeling team.
20.3.8 \textbf{integrityCheckAlgorithm}

We include the property \textit{integrityCheckAlgorithm}.

20.3.9 \textbf{description}

We do not include the property \textit{description}. This property is intended to be a short description of the media contained in case the media cannot be presented.\footnote{Discuss with modeling team.}

20.3.10 \textbf{thumbnail}

We do not include the property \textit{thumbnail}, which is the purpose of the datatype EDNT.

20.3.11 \textbf{translation}

We do not include the property \textit{translation}, which allows for alternate renditions of the same content translated into a different language or a different mediaType.\footnote{Discuss with modeling team. Do we want translations?}

20.3.12 \textbf{length}

We do not include the property \textit{length}, as the length of the binary data can be calculated by the implementation.

20.3.13 \textbf{subpart}

We do not include the property \textit{subpart}, as this is an operator on the stored data.

20.3.14 \textbf{equal}

We do not include the property \textit{equal}, as this is a comparison operator on the stored data.
Chapter 21

BIN

21.1 Binary Data

Figure 21.1: Binary Data

Binary Data (Figure 21.1) is used to represent a raw block of bits. It is not used within Clinical Element models and is only used in the internal definition of the datatypes.

21.2 Properties

Table 21.1: BIN Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>binary</td>
<td>Raw block of bits</td>
</tr>
</tbody>
</table>
21.2. PROPERTIES

21.2.1 value

The property value is a raw block of bits
21.3 HL7 Comparison

Figure 21.2: HL7:INT Declaration

```plaintext
protected type BinaryData alias BIN specializes LIST<BN>;

type Sequence<T> alias LIST<T> specializes COLL<T> {
    T head;
    LIST<T> tail;
    BL isEmpty;
    BL notEmpty;
    T item(INT index);
    BL contains(T item);
    INT length;
    LIST<T> subList(start INT, end INT);
    LIST<T> subList(start INT);
    literal ST.SIMPLE;
    promotion LIST<T> (T x);
    demotion BAG<T>;
};
```

Figure 21.3: BIN to HL7:BIN Comparison

There are no differences between HL7:BIN and our BIN.

21.3.1 value

The property value is not part of the HL7 abstract definition. The HL7 specification leaves it up to implementation where to store the value of binary data. We have chosen to use value.
Part III

Appendix
Appendix A

CWE Cases

A.1 Introduction

We have removed codingRationale out of both our coded types and physical quantity. We decided to only store one translation rather than a set, and due to this we think we can remove codingRationale and implicitly know the codingRationale value.

While this is true for whether the datatype is original, it is not true for whether the datatype was post coded, but we have decided to store post coded information in the wrapper structure which contains a clinical element instance.

Prior to removing codingRationale we investigated the possible use cases to analyze whether this made sense. The following sections walk through each of the use cases.

A.2 Application provides code

One of our applications provides the following ...

```
CWE
   code          SER_NA_ECID
   originalText

It is stored in the repository as ...
```

```
CWE
   code          SER_NA_ECID
   originalText
```
### A.3 Application provides code-originalText

One of our applications provides the following ...

<table>
<thead>
<tr>
<th>CWE</th>
<th>Code</th>
<th>OriginalText</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SER_NA_ECID</td>
<td>SERUM SODIUM</td>
</tr>
</tbody>
</table>

It is stored in the repository as ...

<table>
<thead>
<tr>
<th>CWE</th>
<th>Code</th>
<th>OriginalText</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SER_NA_ECID</td>
<td>SERUM SODIUM</td>
</tr>
</tbody>
</table>

### A.4 Application provides originalText - post-coded

The following data came from an application, with no code because the user couldn’t find an appropriate code:

<table>
<thead>
<tr>
<th>CWE</th>
<th>Code</th>
<th>OriginalText</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SER_NA_ECID</td>
<td>SERUM SODIUM</td>
</tr>
</tbody>
</table>

It is stored in the repository and then post-coded by a human ...

<table>
<thead>
<tr>
<th>CWE</th>
<th>Code</th>
<th>OriginalText</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SER_NA_ECID</td>
<td>SERUM SODIUM</td>
</tr>
</tbody>
</table>
The SER_NA_ECID was coded after the fact by a human, based on what was in originalText. But this looks just like the case above, where a code and an originalText both came in. So how do we indicate that in this case, SER_NA_ECID was coded after the fact – that it wasn’t present in what was originally received?

The possible ways to represent this are listed as such:

1. Store coding rationale (HL7 has a \texttt{AIJPost Coded} value for codingRationale which we could use).
2. Store one version of the data type instance with just original text, then store another version adding a code
3. Use “translated” attribution in the Clinical Element wrapper to indicate that this instance was translated by a human.
4. A combination of the above

We have decided to not use \texttt{codingRational} for this purpose. Instead we will use a “translated” attribution in the Clinical Element wrapper to indicate that this instance was translated by a human. We most likely will also store the original message as well.\footnote{This is a good idea in general. Even instances coming over the interface as HL7 messages could be stored as the original form.}

### A.5 Interface provides code

This is what came in over an interface:

```plaintext
CD
code        SERUM_NA
displayName
cdSys       
cdSysName
cdSysVersion
originalText
translation
codingRation
source
```

Based on sender-receiver agreement, we know the code system and version are ACME Coding Standard version 2.34

It is stored in the repository as:
A.6. INTERFACE PROVIDES CODE-ORIGINALTEXT

APPENDIX A. CWE CASES

If originalText is present in the message, we will use it in our CWE originalText. If there is no originalText in the message, then we will use the message’s code in the CWE originalText. We will also be representing the message’s code in the CET translation, so this is redundant.

One interesting thing to note is that in this example, we implicitly know the coding system and version of the original message, yet we do not represent this. That is because we have decided that the translation is a copy of the message, and this implicit information was not in the message. Our requirement is that we store what came over the wire.

A.6 Interface provides code-originalText

This is what came in over an interface:

Based on sender-receiver agreement, we know the code system and version are ACME Coding Standard version 2.34.

It is stored in the repository as:
According to the rule stated in section A.5, the message’s originalText is copied into originalText of the CWE. Again, we do not represent the implicit information we know about the coding system and version, because this did not come over the wire.

A.7 Interface provides code-code system info

This is what came in over an interface:

```
CD
 code     SERUM NA
 displayName
 cdSys     ACME Coding
 cdSysName  ACME Coding Standard
 cdSysVersion 2.34
 originalText
 translation
 codingRation
 source
```

It is stored in the repository as:

```
CWE
 code     SER NA ECID
 originalText SERUM NA
 translation CET
 code     SERUM NA
```
The important thing to note in this example is in the translation where we store what came over the wire. For the coding system, we do not translate the code “ACME_Coding” into a code such as “ACME_Coding_ECID”. Instead we store exactly what we received.

### A.8 Interface provides code-translation

This is what came over an interface:

```markdown
<table>
<thead>
<tr>
<th>CD</th>
<th>code</th>
<th>SERUM_NA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>displayName</td>
<td>ACME_Coding</td>
</tr>
<tr>
<td></td>
<td>cdSys</td>
<td>ACME Coding Standard</td>
</tr>
<tr>
<td></td>
<td>cdSysVersion</td>
<td>2.34</td>
</tr>
<tr>
<td></td>
<td>originalText</td>
<td>translation</td>
</tr>
<tr>
<td></td>
<td>source</td>
<td>source</td>
</tr>
</tbody>
</table>
```

```markdown
<table>
<thead>
<tr>
<th>CD</th>
<th>code</th>
<th>SERUM_SOD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>displayName</td>
<td>ACME_Coding</td>
</tr>
<tr>
<td></td>
<td>cdSys</td>
<td>3M_Coding</td>
</tr>
<tr>
<td></td>
<td>cdSysName</td>
<td>3M_Coding Standard</td>
</tr>
<tr>
<td></td>
<td>cdSysVersion</td>
<td>1.56</td>
</tr>
<tr>
<td></td>
<td>originalText</td>
<td>translation</td>
</tr>
<tr>
<td></td>
<td>translation</td>
<td>codingRation</td>
</tr>
<tr>
<td></td>
<td>source</td>
<td>Original</td>
</tr>
</tbody>
</table>

It is stored in the repository as:

```markdown
| CWE | code | SER_NA_ECID |
```

150
A.9. UNKNOWN CODING SYSTEM

This is what came in over an interface, KTMI doesn’t recognize Bob’s CodeSystem

It is stored in the repository as:

---

We didn’t discuss this fact, and what is meant by “original”. Also, this is another case for always saving the message and associating it with the resulting translation.
According to the rule stated in section A.5, the message’s `originalText` are copied directly into the CWE. The `code` is unable to be translated, so it is ignored in the CWE. The `translation` will store all of the messages parts including the code that could not be translated.\(^3\)

\(^3\)In an HL7 v3 message, we know that either original text or code or both will be present. In non-HL7 v3 messages, we assume at least a code will be present.
Appendix B

PQ Cases

B.1 Introduction

We have removed codingRationale out of both our coded types and physical quantity. We decided to only store one translation rather than a set, and due to this we think we can remove codingRationale and implicitly know the codingRationale value.

While this is true for whether the datatype is original, it is not true for whether the datatype was post coded, but we have decided to store post coded information in the wrapper structure which contains a clinical element instance.

Prior to removing codingRationale we investigated the possible use cases to analyze whether this made sense. The following sections walk through each of the use cases.

B.2 Application provides unit code

One of our applications provides the following ...

```plaintext
PQ
    value 120
    unit
    CWENT
        code mmHg_ECID
    originalText
```

It is stored in the repository as ...
B.3 Application provides unit code-originalText

One of our applications provides the following ...

```plaintext
PQ
  value 120
  unit
    CWENT
      code mmHg_ECID
      originalText mmHg
```

The application has provided the normal value so it is stored in the PQ section. Since there is no translation, we also know that this is the original.

B.4 Application provides originalText - post-coded

One of our applications provides the following ...

```plaintext
PQ
  value 120
  unit
    CWENT
      code mmHg_ECID
      originalText mmHg
```

The application has provided the normal value so it is stored in the PQ section. Since there is no translation, we also know that this is the original.
It is stored in the repository and then post-coded by a human ...

The mmHg_ECID was coded after the fact by a human, based on what was in originalText. But this looks just like what was stored in section B.3, with a code and an originalText. So how do we indicate that in this case, mmHg_ECID was coded after the fact – that it wasn’t present in what was originally received?

The possible ways to represent this are listed as such:

1. Store coding rationale (HL7 has a Post Coded value for codingRationale which we could use).

2. Store one version of the data type instance with just original text, then store another version adding a code

3. Use “translated” attribution in the Clinical Element wrapper to indicate that this instance was translated by a human.

4. A combination of the above

We have decided to not use codingRational for this purpose. Instead we will use a “translated” attribution in the Clinical Element wrapper to indicate that this instance was translated by a human. We most likely will also store the original message as well.¹

B.5 Interface provides code

This is what came in over an interface:

¹This is a good idea in general. Even instances coming over the interface as HL7 messages could be stored as the original form.
B.6 INTERFACE PROVIDES CODE, NEEDS NORMALIZATION. PQ CASES

The original PQ that came over the interface is actually the normal but the unit is from a different coding system. We still create a translation and put the original PQ into the PQT. There is one big difference here compared to how we handled CWE, and that is we add the code system as an ECID code. Thus the translation is not exactly what came over the wire. We do this because we need to quickly know the coding system so that we can do more translations from PQT to other possible units.²

B.6 Interface provides code, needs normalization

This is what came in over an interface:

²This is another case which shows we should save what came over the wire.
APPENDIX B. PQ CASES B.7. INTERFACE UNKNOWN CODING SYSTEM

We know the upper portion is normal, and we know it is not original because of translation. The only difference from this and the previous example is that we don’t put the code that came over the wire in originalText. This is because that unit is not a surface form of the “mmHg_ECID” code. So the rule is that we only copy the unit surface form when we are not normalizing to a new unit.

B.7 Interface unknown coding system

This is what came in over an interface:

It is stored in the repository as:

---

157
This slightly breaks the rule we have stated that the normal unit is in the **PQ** section, so we must expand that rule to state if there is a *code* in the **PQ** section, then it is the normal. We also add an additional translation which is identical to the **PQ** section. This is because we may eventually post code the **PQ** and then we couldn’t tell it was original.\(^3\)

**B.8 Interface provides code and normalized translation**

This is what came in over an interface:

```plaintext
PQ
value  .157
unit
CS
code atm_UCUM
translation
PQR
value  120
unit mmHg_UCUM
codeSys UCUM
codingRat original
```

It is stored in the repository as:

```plaintext
PQ
value  120
unit
CWENT
code mmHg_ECID
originalText mmHg_UCUM
translation
PQT
value  120
unit mmHg_UCUM
codeSys UCUM_ECID
```

Our system will only keep one of the units that came over the interface. In the **CWE** case we left this up to the interface team, but in the **PQ** case, it is always the original that is retained. This is due to rounding errors in the value that occur with every calculated translation. So basically we treat this use case just like we

\(^3\)I think we may not need the translation until we post code
APPENDIX B. PQ CASES

B.8. INTERFACE PROVIDES CODE AND NORMALIZED TRANSLATION

received the translation, which makes it a case we have seen previously. We are losing data that came over the wire, so this is another case which shows we should save the original message in some manner.
B.8. INTERFACE PROVIDES CODE AND NORMALIZED TRANSLATION
Appendix C

Glossary

C.1 Definitions

Authoring CEML  See CEML

Clinical Element Abstract Instance Model  The Clinical Element Abstract Instance
Model defines a recursive structure that can hold patient instance data. It is
defined abstractly, and thus this model must be implemented in an actual
language such as XML or Java.

Clinical Element Abstract Constraint Model  The Clinical Element Abstract Con-
straint Model is the model used to constrain and describe allowable instances
of patient data. This is the abstract description of our constraint formal-
ism, which is then actually implemented. We have an XML implementation
which is called CEML or Clinical Element Modeling Language.

<cetype/>  A CEML construct. The element which is used to define a Clinical
Element Constraint Type, or CEType.

CE  See Clinical Element. This should not be confused with HL7:CE which is a
now deprecated HL7 version 3 datatype Coded With Equivalents.

CEM  See Clinical Element Model.

CEML  See Clinical Element Modeling Language.

CEO  See Clinical Element Object.

CEType  See Clinical Element Constraint Type
Clinical Element The terms which unfortunately is used rather loosely and incorrectly. The correct usage is that a Clinical Element is the recursive structure in the Abstract Instance Model, also called a Clinical Element Instance Node. It is NOT a Clinical Element Constraint Type.

Clinical Element Instance Node A Clinical Element Instance Node is the recursive structure in the Abstract Instance Model. It consists of a key, a type, a value choice (data or items), modifiers, and qualifiers.

Clinical Element Constraint Type This is a construct defined by the Abstract Constraint Model which is a collection of constraints used to validate an instance from the Abstract Instance Model. The definition of a particular Clinical Element Constraint Type such as a BloodPressurePanel. Instances of this BloodPressurePanel must conform to the constraints stated within this constraint definition.

Clinical Element Instance An instance of patient data in a format that conforms to the Clinical Element Abstract Instance Model. This instance data can then be constrained by a Clinical Element Constraint Type. For example, an instance of patient data such as a hematocrit with a value of 38.9 stored in the EMR for John Doe, that conforms to the constraint type called LabObservationHematocrit.

Clinical Element Model (CEM) Unfortunately, this term is erroneously used to refer to an individual Clinical Element Constraint Type (CEType).

The Clinical Element Model (The CEM) The Clinical Element Model is a term used to denote the global modeling effort as a whole. It is the combination of the Abstract Instance Model and the Abstract Constraint Model.

Clinical Element Modeling Language (CEML) This is our Implementation Technology Specification of the Abstract Constraint Model. It is an XML based syntax. We have 2 forms a Strict CEML and Authoring CEML. Strict CEML follows the Abstract Constraint Model constructs exactly. Authoring CEML includes the syntax of Strict CEML, but also adds shortcut or macro elements to make the definitions more succinct.

Clinical Element Object (CEO) This is a programatic object that is used to manipulate Clinical Element Instance Data. It is analogous to an XML DOM.

Compound Statement A Statement whose meaning is conveyed by multiple clinical values, with associated modifiers and qualifiers. The meaning of the Compound Statement is dependent on a set of elements with values being
interpreted together within the context of the collection. In The Clinical Element Model, a Compound Statement has the value choice of items rather than data. For instance, a pharmacy order is a compound statement. See Statement.

**Component** A CE Constraint Type that is only used within another CE Constraint Type (as an item, qualifier or modifier). A CE Instance that conforms to a component **CEType** can NOT be stored in the patient EMR on it’s own, but only as an internal part of another instance. Examples of Component CETypes include Specimen, MethodAndDevice, BodyPosition, and Length.

**Data** A Construct of the Abstract Instance Model which contains an HL7-derived data type that serves as the **âÁIvalueâÁI** of a Simple Statement.

<item/> An Authoring CEML construct used to constrain Clinical Element Instance Nodes within within the **Items** of a Clinical Element Instance.

**Key** An property within a Clinical Element Instance that is an HL7:CWE. The key’s code links the Clinical Element instance to a real world coding system.

<key/> An Authoring CEML construct used to constrain the **Key** of a Clinical Element Instance.

**Label** DEPRECATED: These are now replaced by Semantic Links with a target of a Coded Concept.

<mod/> An Authoring CEML construct used to constrain Clinical Element Instance Nodes within within the **Mods** of a Clinical Element Instance.

**Modifier** A Clinical Element Instance node which modifies the content of the Value Choice in the containing Clinical Element Instance. The extent of this modification is so great, that the value choice can never be considered independently without simultaneously considering the effect of the modifier on the value choice.

**noninstantiable** noninstantiable is a potential value of the property “kind” in a Clinical Element Constraint Type that indicates that no patient data can be instantiated using this constraint type. Instead, this constraint type is used as a starting point to define other constraint types. A Noninstantiable Clinical Element Constraint Type is a type that does not contain enough information (items, modifiers, qualifiers, etc.) to be instantiated.
Panel  Represents a common grouping of clinical observations. A chem7 lab result is an example of a common lab panel. A panel is a collection of statements that can exist independently. Synonyms used for panel include battery and collection.

Qualifier  Clinical Element Instance node which give more information about the Value Choice in the containing Clinical Element Instance. The degree to which this qualification changes the meaning of the value choice varies, but it is never to the degree of a modifier. In medical informatics circles, some argue you can never even truly ignore a qualifier, so why make the distinction between a qualifier and a modifier.

<qual/>  An Authoring CEMl construct used to constrain Clinical Element Instance Nodes within within the Quals of a Clinical Element Instance.

Semantic Link  Semantic Link is the construct that is used to establish a relationship between separate independent Clinical Element Instances. The semantic link specifies a coded relationship between CE instances; they are not used for static, a priori relationships.

For example, a semantic link may appropriately be used to express the relationship between a particular medication order for Tom and the reason for the order by reference to a specific problem in Tom’s problem list only that particular order is related to that particular problem.

An inappropriate use of a semantic link would be to express the relationship between a lab observation and a specimen all lab observations are related to specimens. This type of static relationship is best defined within the lab observation CE Constraint Type.

Simple Statement  A Statement whose meaning is conveyed by a single clinical value, with associated modifiers and qualifiers. In The Clinical Element Model, A Simple Statement has the value choice of data rather than items. An example of a simple statement is a hematocrit lab result. See Statement.

Specializable CETYPE  A Specializable Clinical Element Constraint Type does not specify one particular value to be used as its Key.code. Instead, the Constraint Types specifies a domain of permitted Key.codes.

For example, quantitative lab result is a Specializable CE Constraint Type, in which the Type is quantitative lab result, and the Key is constrained to the domain of specific quantitative labs like hct, serum sodium, etc.
Specific CEType A Specific Clinical Element Constraint Type specifies a single code as the permissible value of its Key.code.

LabObservationHematocrit is an example of a Specific Clinical Element Constraint Type. Its parent type is the Specializable CEType LabObservationQuantitative. The LabObservationHematocrit CE’s Type is LabObservationHematocrit, and the only Key value allowed is Hct.

A Specializable Clinical Element Constraint Type may be used to validate patient data where no more specific model is available, but the ultimate goal is to have a Specific Clinical Element Constraint Type for everything stored in a patient record.

Statement A complete assertion about a particular aspect, characteristic or condition of a patient. A statement contains a value choice (data or items), and may also contain modifiers and qualifiers. The parts of a Statement are not meaningful by themselves out of context of the Statement. There are two types of statements: simple statements and compound statements.

Strict CEML See CEML
Appendix D

Change Log

Changes to this document are listed below. It should be noted that we did not track changes to the datatype document prior to initially so version Nov. 15, 2008 is the beginning of change tracking.

D.1 November 14, 2008

- **Update** - Discrepancy in normal for PQ fixed between text and table.
- **Update** - Fixed II images