V2_CONFORM_R1_N1_2020FEB

HL7 Version 2 Conformance Methodology Release 1

February 2020

HL7 Normative Ballot

Sponsored by:
Conformance Work Group

Conformance Co-chair: Principle Author	Robert Snelick, National Institute of Standards and Technology (NIST)
Conformance Co-chair	Frank Oemig, Deutsche Telekom Healthcare Security Solutions GmbH
Conformance Co-chair	Nathan Bunker, American Immunization Registry Association (AIRA)
Conformance Co-chair	Ioana Singureanu, Eversolve, LLC

Questions or comments regarding this document should be directed to the Conformance Workgroup (cgit@lists.hl7.org) or HL7 Zulip Conformance stream (https://chat.hl7.org).

Acknowledgments

The authors of this document wish to recognize the following participants who contributed their time and expertise to the development of this specification.

Craig Newman	Altarum
Caroline Rosin	National Institute of Standards and Technology (NIST)
Hossam Tamri	National Institute of Standards and Technology (NIST)
Sheryl Taylor	National Institute of Standards and Technology (NIST)

Portions of this document were adopted ("as-is" or "revised") from external sources. These include:

- Healthcare Interoperability Standards Compliance Handbook. F. Oemig, R. Snelick 2016.
- HL7 v2.8.2 Chapter 2B Control: Conformance
- Selected HL7 v2 Balloted Implementation Guides
- Specifications and materials created by the National Institute of Standards and Technology

Copyrights

TABLE OF CONTENTS

Table of Contents

Table of Contents

Table of Contents

INDEX OF TABLES

Index of Tables

INDEX OF FIGURES

1 INTRODUCTION

This document (*HL7 Version 2 Conformance Methodology)* explains the procedures and processes for constraining HL7 v2 message specifications, encompassing both message profiles and implementation guides that contain message profiles. A message profile provides a precise and unambiguous specification of a single message definition. An implementation guide provides a broader context in which typically a set of message profiles are used to satisfy one or more use cases. A use case may include applications (actors) and the interaction between them (interaction model) as well as application functional requirements.

The base HL7 v2 standard is a framework that contains many message events, and for each event it provides an initial message template (starting point) that is intended to be constrained for a specific use and context. The process of placing additional constraints on a message definition is called **profiling**. The resulting constrained message definition is called a **message profile** (also referred to as a conformance profile and, hereafter, message profile). Message profiles provide measurable requirements that can be used to determine the degree to which implementations are conformant. Profiles reduce or eliminate the optionality (or openness) of the base standard. An example of a constraint is changing optional usage for a data element in the base standard message definition to required usage in the message profile.

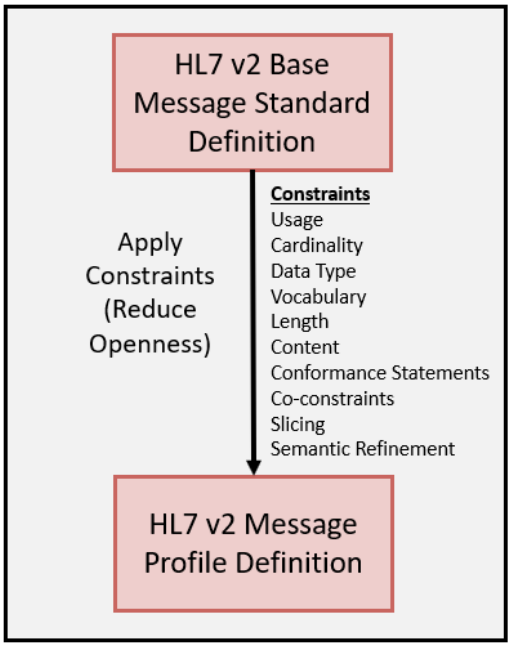

Figure 1.1: HL7 v2 Profiling Process

Figure 1.1 (HL7 v2 Profiling Process) illustrates the process by which a standard defined message event (e.g., an ADT^A04 message) is refined for a desired use. The message profile definition can be documented in a natural language or in a computable representation.

An HL7 v2 Implementation Guide provides a broader context that may contain a detailed specification for an interoperability solution. Figure 1.2 (HL7 v2 Implementation Guide Sample

Template) presents a sample outline of an implementation guide. An implementation guide can vary in the components it will contain. Some implementation guides may contain complete workflow requirements including actors and functional requirements while others will simply focus on the interoperability aspects of a set of related interactions. An implementation guide typically will contain a narrative part (upper part of Figure 1.2) and a message definition (profile) part. Section 3 provides additional insight into the content commonly given in implementation guides.

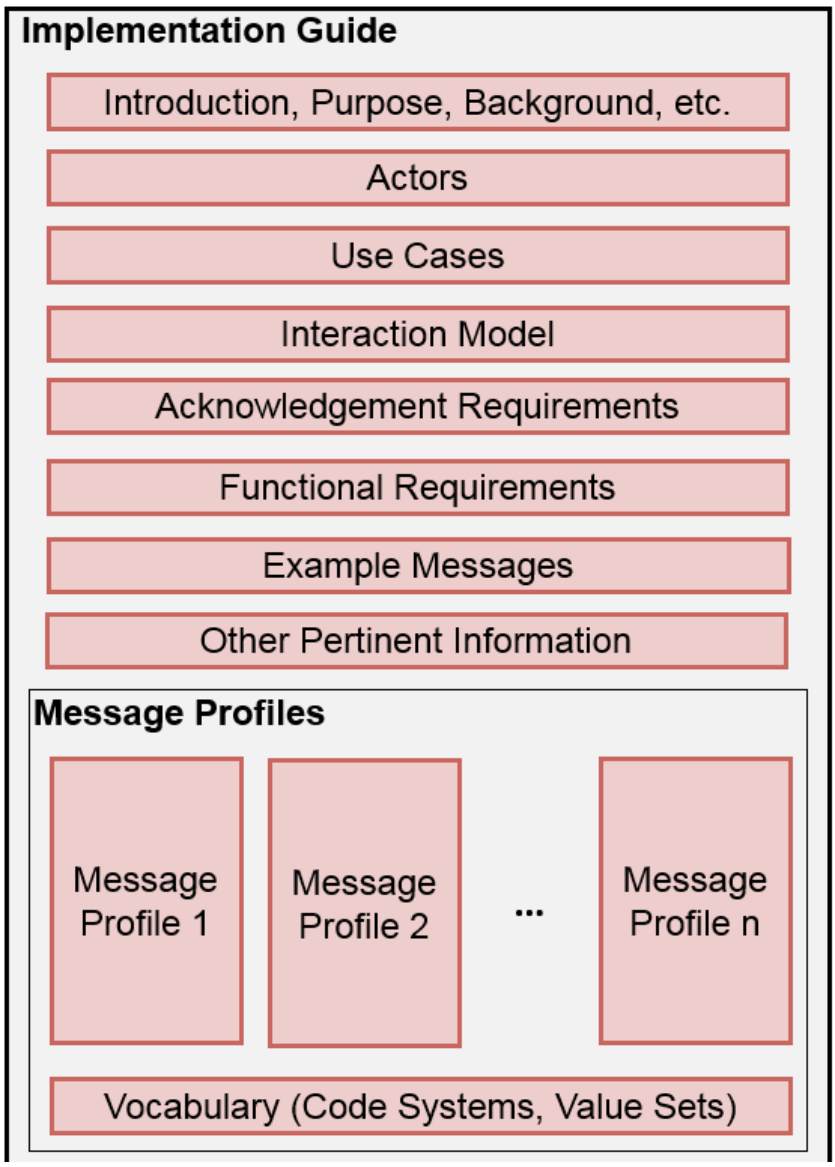

Figure 1.2: HL7 v2 Implementation Guide Sample Template

The central aspect of an implementation guide is the constrained message definitions, i.e., the message profiles. Figure 1.3 illustrates the make-up of a message profile at a high-level. The message is embodied as a structured definition. Rules for an abstract message definition are specified in the HL7 v2 message framework, which is hierarchical and consists of building blocks generically called elements. These elements are segment groups, segments, fields, and

data types (i.e., components and sub-components). The requirements for a message are defined by the message definition and the constraints placed on each element. The methods and rules for applying constraints are defined by the HL7 v2 conformance constructs (constraint mechanisms). The conformance constructs include usage, cardinality, length, vocabulary, content (non-coded), data type specialization, explicit conformance statements, co-constraints, and other constraint mechanisms. Section 5 provides definitions of the conformance constructs.

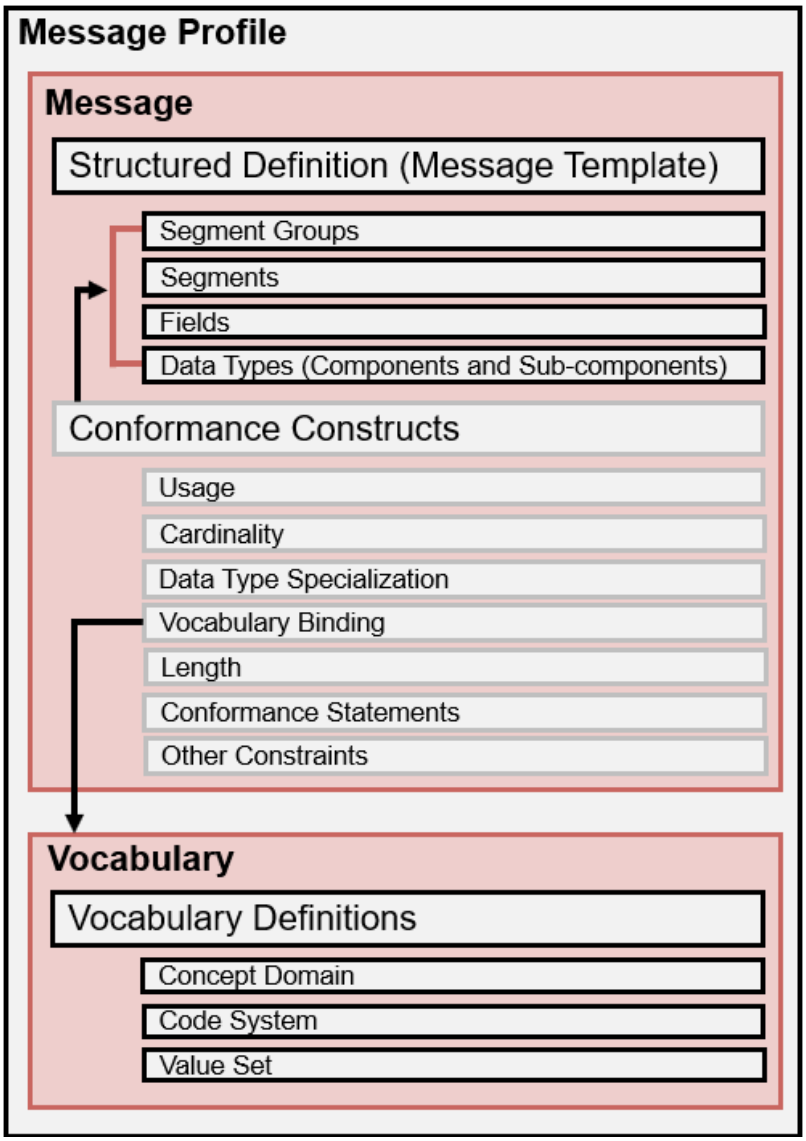

Figure 1.3: Message Profile Overview

The message profile also includes the associated vocabulary bindings and definitions. Certain message elements can contain coded data. The element in the structured definition is bound to a vocabulary definition. Depending on the profile level and the specification need, this binding can be to a concept domain, code system, or value set. The vocabulary definition provides an identifier and the set of allowable content. Section 6 defines the concepts and rules for vocabulary binding and profiling.

The goal of HL7 v2 is to provide a standardized protocol for exchanging healthcare related data between two systems. Figure 1.4 (Generic HL7 v2 Interaction Diagrams) provides the basic model upon which messages are used to exchange data. Two systems or applications (generically called Actors) exchange data; this conversation is referred to as an **interaction**. The content of what is exchanged is defined in the message profile. As such, an interaction has a one-to-one relationship with a message profile.

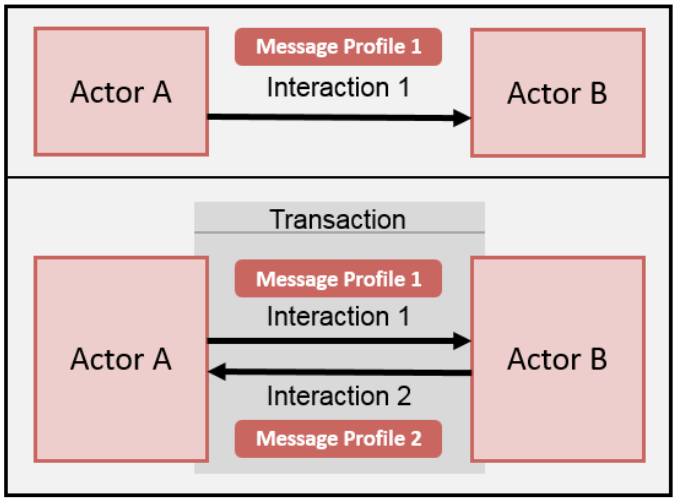

Figure 1.4: Generic HL7 v2 Interaction Diagrams

The lower part of Figure 1.4 illustrates a *roundtrip* conversation; this sequence of events is referred to as a **transaction**. As shown, for each interaction a corresponding message profile is associated. An example may be actor A sending a "Register a Patient" message to Actor B. Actor B responds with an "Acknowledgement" message. Interactions and transactions can be defined in a given context and documented in an implementation guide.

1.1 Purpose

This specification provides the rules and documentation requirements for profiling HL7 v2 base message definitions. It also provides guidance on how to assemble a set of message profiles to satisfy the requirements of a set of use cases documented in an implementation guide. A goal is to provide specifiers and implementers the *tools* to define requirements in a clear and precise manner regardless of the level of specificity they seek (e.g., national level requirements or local implementation requirements).

There are many versions of the HL7 v2 standard. An explicit conformance methodology was introduced into the standard in version 2.5 and revised in subsequent versions of the standard. The conformance methodology was part of Chapter 2 initially and later became a subchapter (Chapter 2B Conformance). This document serves to replace (override) the conformance methodology in these earlier versions and it is intended to be applied to any HL7 v2 version. That is, if a specifier is seeking to constrain a version 2.5.1 message, they must use the conformance methodology prescribed in this document and not the conformance methodology given in version 2.5.1.

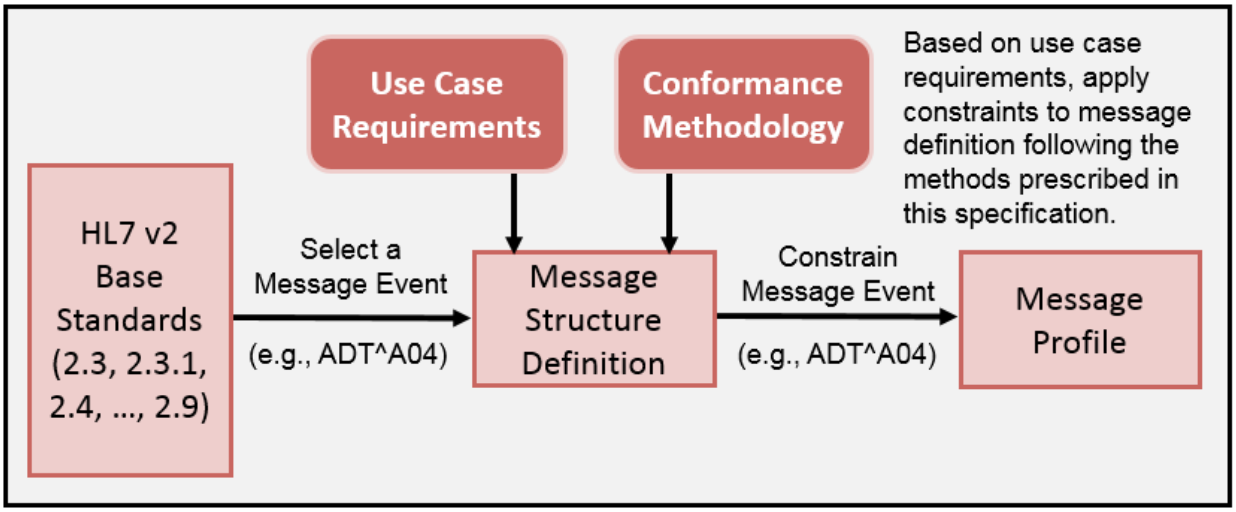

Figure 1.5: HL7 v2 Profiling Process

Figure 1.5 summarizes the profiling process and how this document is intended to be used. The specifier has at their disposal all message events for every HL7 v2 version. Based on their needs, they select a message event to constrain. Using this specification (Conformance Methodology) and use case requirements, they apply constraints to create a message profile. This document describes this process and additionally provides guidance on effective strategies for grouping message profiles for broader context and use in implementation guides.

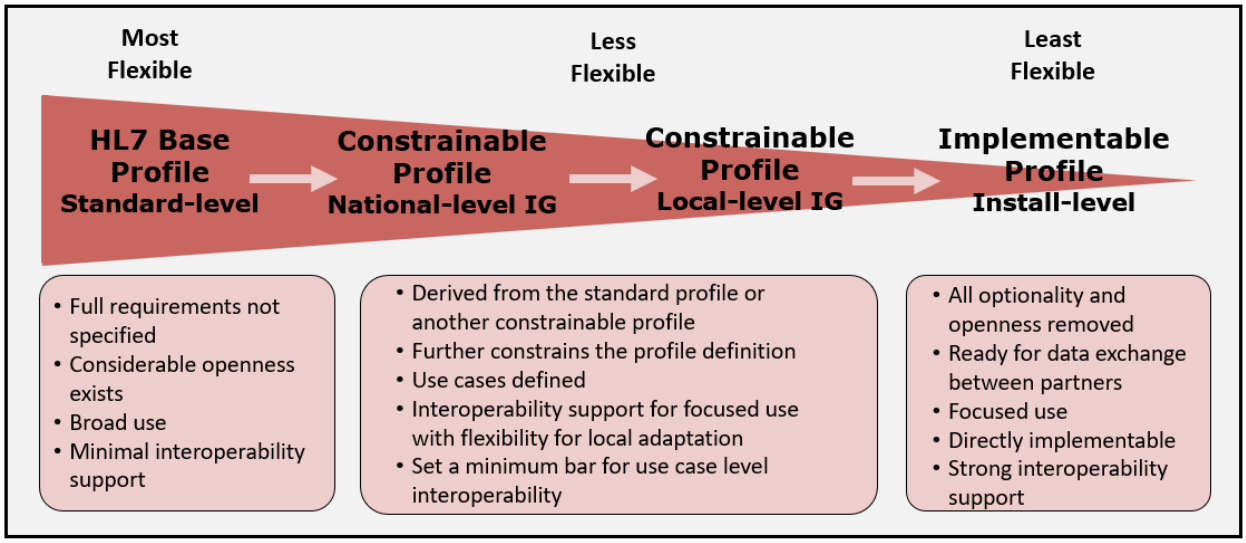

Figure 1.6: HL7 v2 Profiling Spectrum

This specification can be used in implementation guides to document constraints on messages. It is intended to be used for new interfaces and re-specification of existing interfaces. Though use of the conformance methodology in the production setting is optional, its use is recommended. For any newly balloted HL7 v2 implementation guides, however, its inclusion is required. The specifier can choose to use as many of the constructs that are defined in this specification as they

deem appropriate. Additionally, they can choose to specify profiles at a highly general (considerable openness) or very detailed level (no optionality). See Figure 1.6. The choices are dependent on the intended use of the implementation guide. Implementation guides that are too specific have reduced applicability but provide detailed requirements that allow implementations to move closer to achieving interoperability if applied faithfully. Implementation guides that are defined at a high-level with many optional aspects are more amenable for wide-spread use, but they fall far short of out-of-the-box interoperability and need further profiling for specific use. This difference presents the specifier with a tradeoff decision that is based on the intended purpose of the implementation guide. Either approach, and anything in between, is correct and appropriate.

1.1.1 RELEVANT SPECIFICATIONS

The HL7 Version 2 Conformance Methodology is intended to be used as the basis for creating implementation guides and message profiles for the following standards:

- HL7 v2.3
- HL7 v2.3.1
- HL7 v2.4
- HL7 v2.5
- HL7 v2.5.1
- HL7 v2.6
- HL7 v2.7
- HL7 v2.7.1
- HL7 v2.8
- HL7 v2.8.1
- HL7 v2.8.2
- HL7 v2.9
- Any future HL7 v2 standard releases (e.g., HL7 v2.+)

HL7 versions 2.1 and 2.2 have been excluded on purpose because the associated constructs are so immature that they are not subject to practical use of this specification. It is strongly recommended that newly created implementation guides do not use these versions. The Conformance Methodology is also applicable to the tables defined in the HL7 standard (Chapter 2C as of version 2.7) and any external vocabulary standards used in the specification of an implementation guide and message profile.

1.1.2 REQUISITE KNOWLEDGE

- HL7 v2.x Messaging Standard (www.hl7.org)

1.1.3 SUPPLEMENTAL RESOURCES

An online version of this specification, along with additional informative information, can be accessed at https://v2.hl7.org/conformance (Forthcoming). The main tabs on the web site include an exact copy of this specification. Supplemental information, such as extended examples, historical insight, and constraint assessment tables, can be found on the sub-tabs for a given section.

1.2 Conventions and Definitions

1.2.1 KEYWORDS

The terms used to express requirements in this document follow the guidelines as described in RFC 2119[1] (adapted). Table 1.1 provides a summary of keywords and how they are to be interpreted. The convention for keyword expression is to use non-bold lower case.

Table 1.1: Keywords

Keyword	Definition
must	means that the definition is an absolute requirement of the specification. Synonymous with the terms “shall” and “required”. The term “required” is reserved for use within the HL7 v2 conformance constructs and will not be used as a keyword in this specification to describe requirements, e.g., “usage” uses the term “required”.
must not	means that the definition is an absolute prohibition of the specification. Synonymous with the term “shall not”.
should	means that there is strong preference that a statement is applied. Valid reasons may exist to ignore a statement, but the full implications must be understood and carefully weighed before choosing a different course. Synonymous with the term “recommended”.
should not	means that there is a strong preference that a statement is not applied. Valid reasons may exist when a statement is acceptable or even useful, but the full implications should be understood, and the case carefully weighed. Synonymous with the term “not recommended”.
may	means that an item is truly optional. A specifier has the option to include or not to include the entity in the implementation guide or message profile. Synonymous with the terms “optional” and “permitted”. The terms “optional” and “permitted” are reserved for use within the HL7 v2 conformance constructs and will not be used as keywords in this specification to describe requirements, e.g., “usage” uses the term “optional”.

Note: the keywords presented in this section are used to state requirements in this document. A similar set of keywords that are recommended for use in an HL7 v2 Implementation Guide appear in Section 3.1.7. This document seeks to distinguish the keywords used herein from those that might be used in implementation guides, but that distinction is not always possible.

[1] http://www.ietf.org/rfc/rfc2119.txt

1.2.2 DEFINITIONS

The following terms will assist in the definitions and interpretation of requirements for the conformance constructs such as usage and conformance statements.

Value (as a verb): To place a non-empty value (noun) in an element location. To indicate that an element must be valued, the phrase “shall be valued” is used. To inquire if an element is valued, the phrase “is valued” is used (e.g., If PID-7 is valued). The equivalent inverse phrases are “shall not be valued” and “is not valued”.

Non-empty Value: A value in which at least one-character is a non-whitespace character. Two double quotes (""), which represents a “delete indicator”, qualifies as a non-empty value.

Empty Value: An element lacking content. Indicated in er7 format for a field as ||.

Known Data: Content that exists and is available for a given element.

Process: Is a general term to indicate that an action must be performed, and is an important concept for understanding receiver-side requirements. Message usage requirements alone cannot fully indicate the scope of processing requirements. Additional functional requirements for how data elements should be processed can be defined.

Exception: Is a general term to indicate that a conformance violation has occurred. The response to an exception is context-dependent.

1.2.3 ELEMENT REPRESENTATION

Message elements are addressable and represented in a standardized form and are dependent on the element context. The element representation presented here is used in this specification to identify the location of elements within a context. It is important to understand this nomenclature to interpret the representation of requirements.

Element contexts include the message, group, segment, or datatype. Depending on the context, a leaf element is referred to either by name or by a name and a position number. For message or group contexts, the leaf elements are accessed by name (e.g., MSH). For segment or datatype contexts, the leaf elements are accessed by a name and a position number (e.g., MSH-7). Table 1.2 illustrates the grammar of the representation for a given context ([] indicates optional, * indicates multiple occurrences).

Table 1.2: Element Representation Grammars

Context	Grammar
Data Type	DATATYPE[.POSITION[.POSITION]]
Segment	SEGMENT[-FIELD[.COMPONENT[.SUBCOMPONENT]]]
Group	GROUP[.GROUP]*.[SEGMENT[-FIELD[.COMPONENT[.SUBCOMPONENT]]]]
Message	MESSAGE[.GROUP]*.[SEGMENT[-FIELD[.COMPONENT[.SUBCOMPONENT]]]]

The datatype context is the name of the datatype, for example, CX. As described in the grammar, a data type element is represented by the name of the data type and position; for example, the first component of a CX data type is CX.1 (ID Number). If the data type is complex (i.e., not primitive), then the subcomponents are represented using their position number with respect to the parent (component) element. For example, CX.4 (Assigning Authority) is a

complex element. Its child elements (subcomponents) are represented as CX.4.1 (Namespace ID), CX.4.2 (Universal ID) and CX.4.3 (Universal ID Type). The location CX.4.1 refers to the first subcomponent of the fourth component of any element with a CX datatype in the message structure (i.e., the context of the element is not known).

The segment context is the name of the segment, for example, PID. As described in the grammar, a segment element and its descendants are represented by the name of the segment, field position, component position, and subcomponent position. For example, the first field of a PID segment is PID-1 (SetID). The location "PID-1" refers to the first field of any PID segment in the message structure. If the field is complex, then child (component) elements are represented using their position number with respect to the parent (field) element. For example, "PID-3" (Patient Identifier List) is a complex element. Its child elements are represented as PID-3.1 (ID Number), PID-3.2 (Check Digit), PID-3.3 (Check Digit Scheme), PID-3.4 (Assigning Authority), etc. Since PID-3.4 is a complex element, it has addressable child elements, PID-3.4.1 (Namespace ID), PID-3.4.2 (Universal ID), and PID-3.4.3 (Universal ID Type). Note that PID-3.4 is the CX data type. For the data type example of CX.4, the context was unknown. In the case of PID-3.4 the context is known at the segment level.

The group context is the name of the group, for example, PATIENT. The representation of an element from the group context is the same as the scheme for the segment context except that the group name is prepended. For example, the third field in the PID segment in the context of the PATIENT group is represented as PATIENT.PID-3. Furthermore, components and subcomponents are represented following the same pattern, for example, PATIENT.PID-3.4 and PATIENT.PID-3.4.

Nested groups are represented by stringing together the group list separated by a dot. For example, ORDER_OBSERVATION.OBSERVATION.OBX refers to the OBX segment in the context of the OBSERVATION group in the context of ORDER_OBSERVATION group; whereas

ORDER_OBSERVATION.SPECIMEN.OBX refers to the OBX segment within the context of SPECIMEN group within the context of ORDER_OBSERVATION group.

Element location is also addressable from the context of a message. The scheme used is the same as the group context, except the message structure is prepended. Examples include ORU_R01.MSH-12 and ORU_R01.PATIENT_RESULT. ORDER_OBSERVATION.OBR-4.1.

Note that in some instances there is no way to differentiate a particular occurrence of a segment when using the location notation of MESSAGE.GROUP.SEGMENT or GROUP.SEGMENT. This circumstance occurs when the same segment can be present at multiple positions under the same parent element in the message or group context location. An example is the MFN_M04 message structure in HL7 v2.8.1. In such cases, the specifier should supplement the requirement with a narrative comment.

1.3 Concepts and their Relationships

Several core concepts related to standards development and implementation are depicted in Figure 1.7. It is important to understand these concepts and their relationships to better apply the notions in this specification.

Profiling: is the process of placing additional constraints on a message definition in accordance with defined profiling compliance rules to meet requirements stated in a use case. The terms "**derived profile**" and, more generically, "**derived specification**" are used to denote a technical

documentation that is based on another technical document. Profiling is an iterative process in which more detail is provided in each derived specification.

Compliance: is the degree to which a derived specification adheres to the requirements defined in the foundational specification (standard); in other words, are the rules for adding constraints faithfully followed? Compliance rules for each of the conformance constructs are defined in subsequent sections. An example of a usage compliance rule is that a "required" element must remain "required" in any derived specifications (profiles).

Conformance: is a relationship between a specification and an implementation and is an indication of how closely the software implements the requirements stated in the specification[2]. The HL7 v2 standard and any subsequent profiling express the requirements, an implementation implements these requirements, and conformance is an objective measure of how closely an implementation satisfies the stated requirements. As such, conformance is associated with the recognition of formal testing to verify adherence to the standard. An example sequence is as follows: an element is specified as "required", an implementation either implements the requirement or not, and conformance is determined by a testing mechanism.

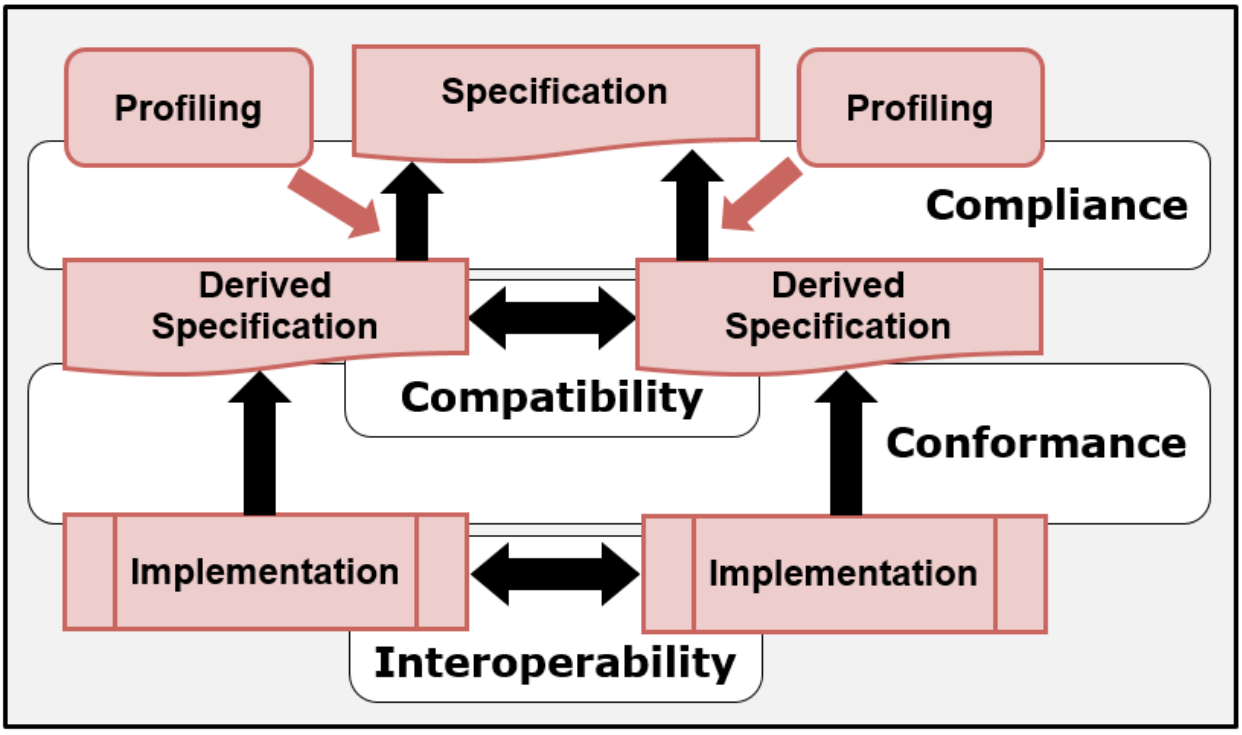

Figure 1.7: Specification and implementation relationships

Compatibility: declares whether two specifications define sets of requirements for the same use case that are harmonized with each other, allowing systems that implement them to *work* together, i.e., interoperate.

[2] In the diagram the relationship is shown to a derived specification; a derived specification is a specification.

Interoperability: is the ability of implementations to exchange data and to use that data as intended to accomplish a desired task. In theory, compatibility is a prerequisite for interoperability[3].

Figure 1.8 shows the concepts and their relationships in the context of HL7 v2. A "specifier" selects the ADT^A04 message event from HL7 v2.5.1, for example, and seeks to constrain the message definition for a given use. Using the constraint mechanisms, the "specifier" defines a message profile. The "specifier" may choose to create a single profile or create a different profile for the sender and receiver. In practice, often a single profile is defined for the sender and receiver, and if separate profiles are created, the differences usually are minor. Additionally, profiles can be created independently by separate entities, and the profiles can be compared for compatibility. Implementations will then develop to a given profile.

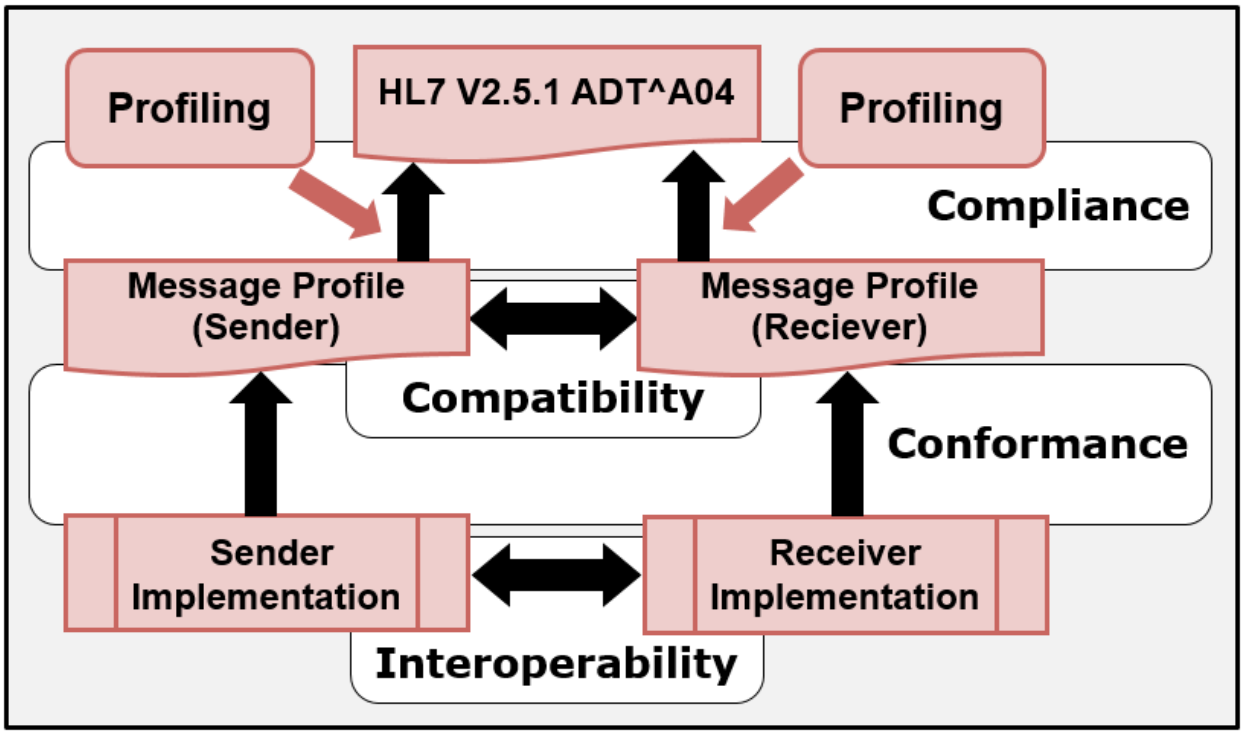

Figure 1.8: Specification and implementation relationships

1.4 Profiling and Profile Construction Overview

1.4.1 PROFILING

Profiling is a refinement to a specification in the form of constraints and extensions (See Section 4.7 for a discussion on extensions). Initially, the refinement is to the base standard and subsequently to existing profiles in a layered approach. Conformance constructs are used to specify constraints on data elements and to provide a "toolbox" for authors to specify requirements. Each conformance construct (e.g., usage) has different levels of constraint

[3] Often systems will implement to the same interface specification, and in such cases compatibility is not of concern.

specification. Profiling is the process of refining the constraint according to the rules of the conformance construct. The constraint rules provide the mechanism to make a requirement more specific (i.e., "to tighten" a requirement).

Figure 1.9 illustrates an overview of message profiling and a sample of the message constituent parts that can be constrained. A profile can be thought of as an overlay of constraints on an HL7 v2 message structure definition. On the left-hand side of the figure is a representation of a base message definition with initial element settings. On the right-hand side of the figure is a representation of the same message structure definition with additional constraints applied. Examples include changing the optional segment of NK1 to required and constraining the base standard HL70001 table to a value set and binding it to the field PID-8.

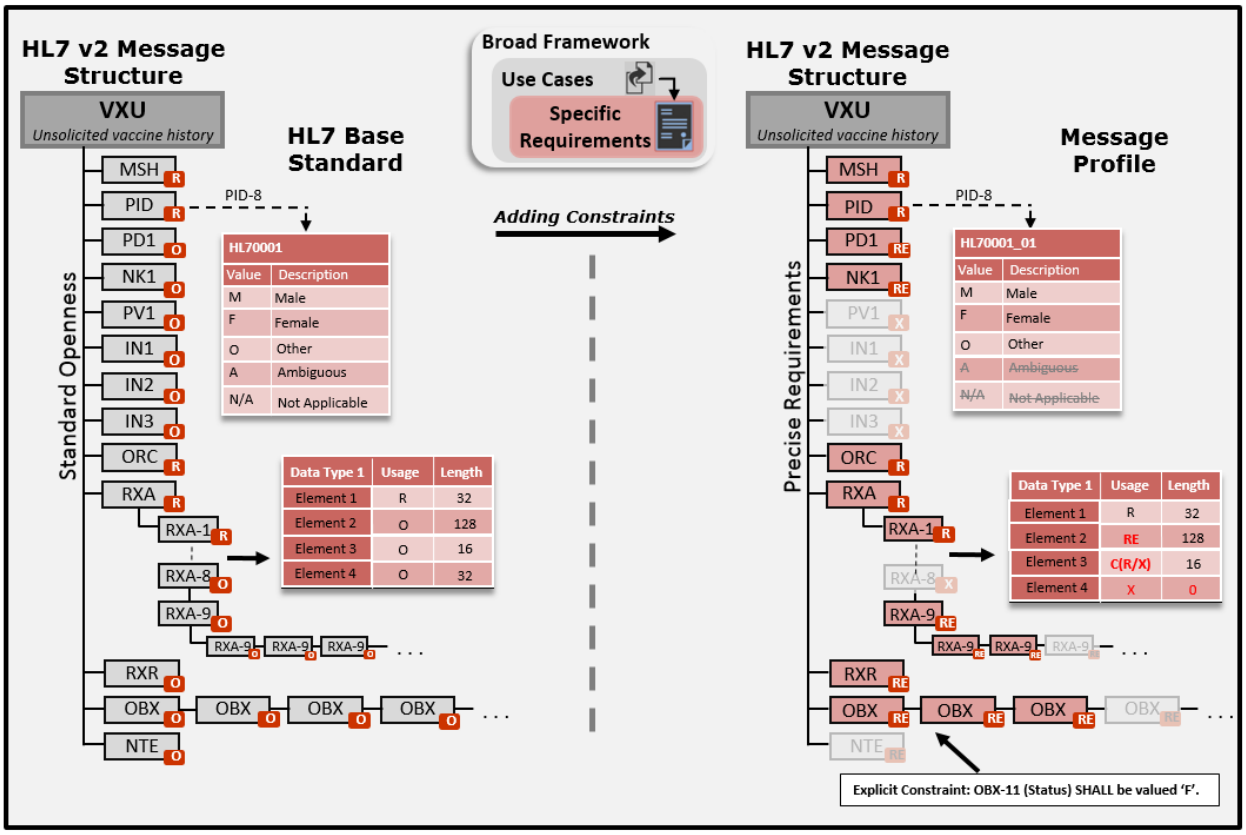

Figure 1.9: Profiling Overview Example

1.4.2 CONSTRAINTS

The key mechanisms for profiling are the conformance constructs that are used for constraining. A high-level overview of the general constraint types is given in Table 1.3. (See Section 5 for in-depth definition and use).

Note that in many ways content, conformance statements, and co-constraints are related and overlap. They provide various options to the specifier to select the *best* way to express constraints. For example, a co-constraints table is a convenient method to express numerous data

dependencies in a set and within Observations (OBX Segments) but these constraints also could have been declared in one or more complex conformance statements.

Table 1.3: General Constraint Types

Constraint Type	Description
Usage	Indicates requirements for the presence (appearance) of an element.
Cardinality	Indicates the number of occurrences for an element by specifying the minimum and maximum bounds.
Data Type	Defines the data element structure and, at the primitive level, the type of data it may contain. Constraints include type substitution and specialization (when combined with other constraint types).
Vocabulary	Defines the vocabulary binding and vocabulary definitions. Indicates the allowable content for a coded element.
Length	Defines a constraint on the number of characters that may be present in one occurrence of an element. Can specify a maximum length, a minimum and maximum length, or minimum for the maximum length supported (Conformance Length).
Content (non-coded)	Defines constraints on data (content), such as a fixed value or adherence to a specific format.
Conformance Statement	A method of expressing content constraints. An explicit statement expressed in text or a testable expression that defines a constraint.
Co-Constraint	Content constraints used to express dependencies among a set of data values.
Slicing	Allows for occurrences of a field to be defined with different constraints.
Semantic Refinement	Allows for refinement of the semantics of a data element based on the use case.

1.4.3 PROFILE CONSTRUCTION

A message profile is normally thought of as a complete message structure definition with additional constraints applied to it as a "whole". In some circumstances, however, it is convenient and efficient to employ a modular approach to profile construction. A **profile component** defines a part or a certain aspect of a message definition and is used to aggregate correlated requirements and/or to differentiate requirements from another profile. It provides a mechanism to support a set of reusable requirements. A profile component can be applied to any construct or section of a message definition. A **core profile** is used to document the common set of requirements across the set of related profiles. A **composite profile** is the composition of a profile or core profile and one or more profile components. In the end, a composite profile is a profile with the distinction that the profile was created by combining a profile or a core profile and one or more profile components. Profiles and profile components can be combined to

develop and manage other profiles. A profile component in a family of profiles can be used to identify different levels of requirements for the same use case or to identify the differences in requirements for different, but closely related, use cases.

One example is the case of national and local requirements. A typical domain for which localization is needed is public health, where a national-level profile is created and additional requirements (constraints) are specified at the state- or jurisdictional-level. A profile for the national level is created for a particular interaction (e.g., send immunization record). State-level requirements could be expressed in a profile component. When combined with the national-level profile, the result expresses the complete requirements for the state.

More discussion on profile construction and scenarios for its use is given in Section 7.1.

1.4.4 PROFILE ROLE AND COMPATIBILITY

Message profiles are used to document interface capabilities of applications. The ultimate goal is to connect applications to allow exchange of data. These interactions include a sending application and a receiving application. The requirements or expectations of each application may, and often do, differ; hence, each application will take on a certain role in the interaction, for example, the role of a sender or a receiver. As such, an interaction pair can be established in which the applications define their own message profiles depending on their role in the interaction. For a successful interface, the requirements for the sender and receiver must be compatible. Sender and receiver profiles can be paired in a variety of ways to satisfy a targeted use. The profile pairings may be closely aligned or disparate. In the case where the sender and receiver profiles are intended to be the same (i.e., the same interface requirements in terms of the expected data exchange), a common profile can be specified, and the profile role is declared as "both". In the case of a different profile definition for the sender and the receiver, compatibility must be considered.

Compatibility can be considered and assessed from different viewpoints. In a coordinated setting, such as a standard organization, a use case may be defined, and the authors of the profile ensure compatible definitions by either developing a common profile applicable to both sides of the interaction or developing compatible profiles for the sender and the receiver. Needs of both the sender and receiver are accounted for. For example, if a sender requires a certain data element to be processed by the receiver, then, as part of the interface negotiations, both the sender and receiver profiles must specify the data element as required to satisfy the use case. A determination of compatibility at the use case level is defined as **functional compatibility**. A second viewpoint is when an assessment of compatibility is made on existing profile definitions (e.g., installed interfaces). Implementers or testers may want to determine if a set of profiles (or interfaces) are compatible[4]. In this case the assessment is made without any consideration of a "higher-level" use case and is determined solely at a technical level with respect to receiver-side requirements. This type of compatibility assessment is defined as **technical compatibility**. Both viewpoints are useful for developing or comparing profiles (interfaces).

The type of profile pairing influences compatibility. In the case of closely aligned requirements in which the sender has an interest in how the data are processed on the receiver side, requirements will be closely matched Figure 1.10. In the case of disparate alignment, such as that in a broadcast scenario, the sender may have no expectations of how the data are processed on

[4] If the profiles are not compatible, the assessment is useful for determining what remedies can be applied if the parties determine to make the interfaces compatible.

the receiver side Figure 1.11. In such a case, the sender may provide data that the receiver does not need. This scenario is compatible. However, the inverse is not true. A sender that does not provide data needed by a receiver is not compatible. An example of this is shown in Figure 1.11. For Receiver 3, Element 1 is compatible because the receiver does not need the data for this element; Element 2 is compatible because the element is required for both; Element 3 is not compatible because the sender is not providing data that the receiver requires.

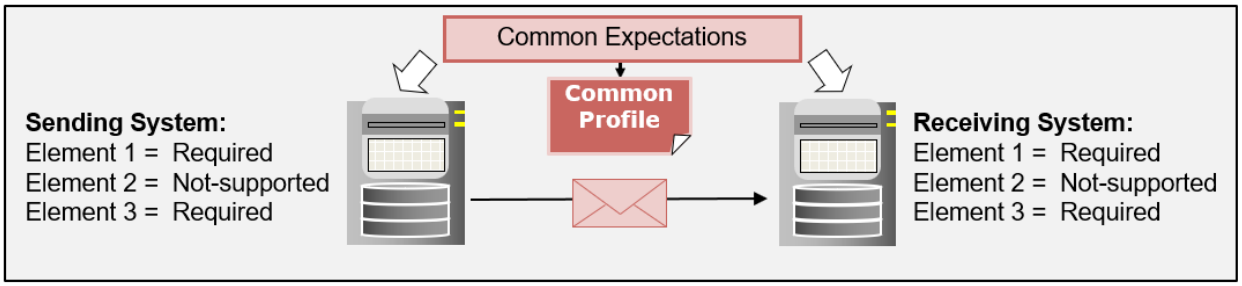

Figure 1.10: Profile Pairing (Common Expectations)

The compatibility assessment tables (for usage, cardinality, data type, length, and vocabulary) in Section 5 (Constraints) and Section 6 (Vocabulary Constraints) are based on technical compatibility since the inquiry is made on established profiles. Non-compatible definitions can be made compatible through negotiations. The technical compatibility assessment tables provide a utility to perform the analysis. Functional compatibility is achieved through deliberation either in an a priori profile-coordinated development or a post-assessment negotiations profile development. Discussion about how profiles are paired to satisfy common use case patterns is covered in Section 8 (Pairing Sender and Receiver Profile for Use).

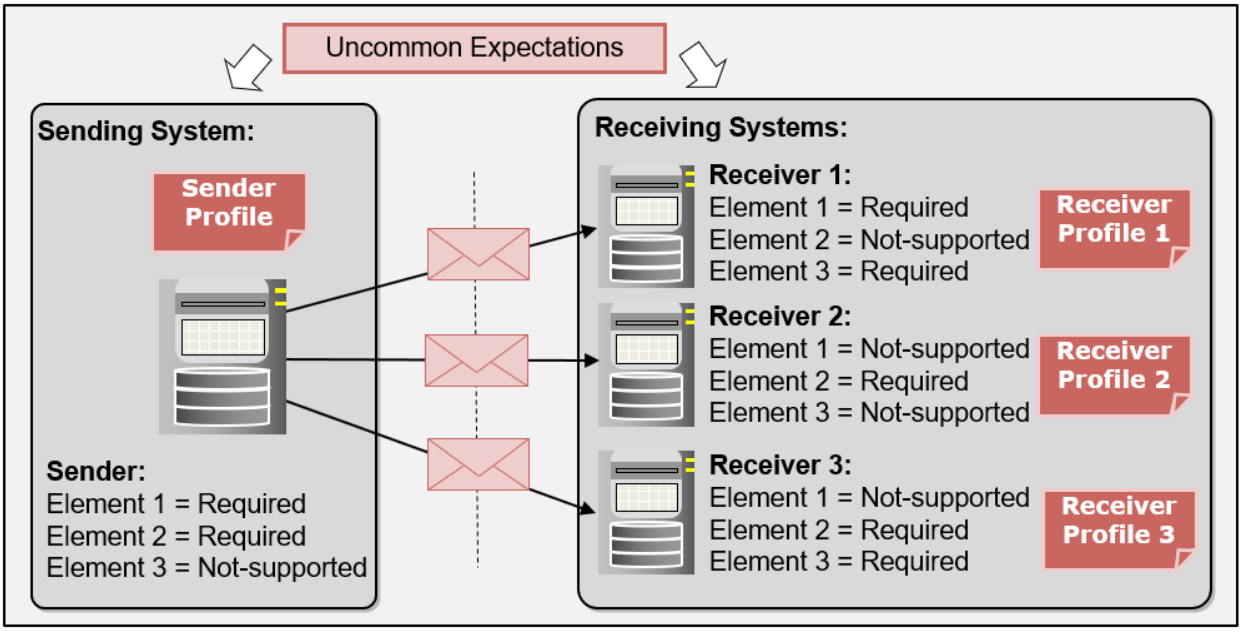

Figure 1.11: Profile Pairing (Uncommon Expectations)

1.4.5 SEGMENT FLAVORS AND DATA TYPE FLAVORS

There are a number of approaches for documenting profiling constraints. This specification recommends the use of segment flavors and data type flavors in which constraints are applied to a base element (e.g., a segment) and assigned an identifier (e.g., PID_01). Subsequently, the element is bound to the higher-level element (e.g., PID_01 is bound to the message definition where appropriate). This approach allows for multiple definitions of a base element type and reuse of those definitions, and, therefore, a precise definition of the construct can be assigned at the appropriate context. Use of this method eliminates comingled requirements commonly found in existing HL7 v2 implementation guides.

1.5 Implementation Guides in Context of Use

Implementation guides define requirements for specific use cases and are intended to be used by implementers to provide an interoperability solution. Implementation guides can also be used by testing programs to assess and certify implementations. Figure 1.12 illustrates how implementation guides (and their message profiles) can be used to facilitate implementation, implementation claims, validation, and reaction to the findings of the validation. Point (1) indicates that an implementation guide includes a conformance clause (2) that defines the requirements, criteria, or conditions that must be satisfied by the implementation to claim conformance. The conformance clause identifies what must conform and how conformance can be met. A conformance claim is a declaration by an implementer (3) of the requirements in the conformance clause that their implementation satisfies.

The conformance clause typically indicates only the high-level conformance requirements. The requirements may be stated in terms of a profile[5] or identifiable parts of a specification. Profiles are mechanisms that provide a structured approach to specifying requirements. The detailed requirements (4) are contained and delineated in the other parts of the specification (e.g., Segment definitions). For example, a conformance clause in an HL7 v2 implementation guide could indicate a set of profiles available to the implementer, and the profiles provide the map to the specific requirements.

Implementers make their conformance claims (3) based on their implementation (5). Implementations or aspects of an implementation (e.g., a message instance) can be validated (6) with respect to the declared implementer's conformance claim (7) that references the requirements set forth in (2) and (4). A Tester can analyze the results of the validation (8). In a testing program, an assessment will be made, and some level of recognition may be issued (9), e.g., a certification. In a testing program, the pass/fail criteria are rigid—either the implementation meets a requirement, or it does not. In a production environment, the response to a conformance violation is at the discretion of the receiver (10)[6]. The use of message profiles to determine conformance and the response to the conformance assessment is often misunderstood. This topic is expanded upon in the next section.

[5] Including composite profiles and profile components.

[6] In terms of how the implementation processes the data. The implementation should, however, follow any acknowledgement requirements indicated in the implementation guide.

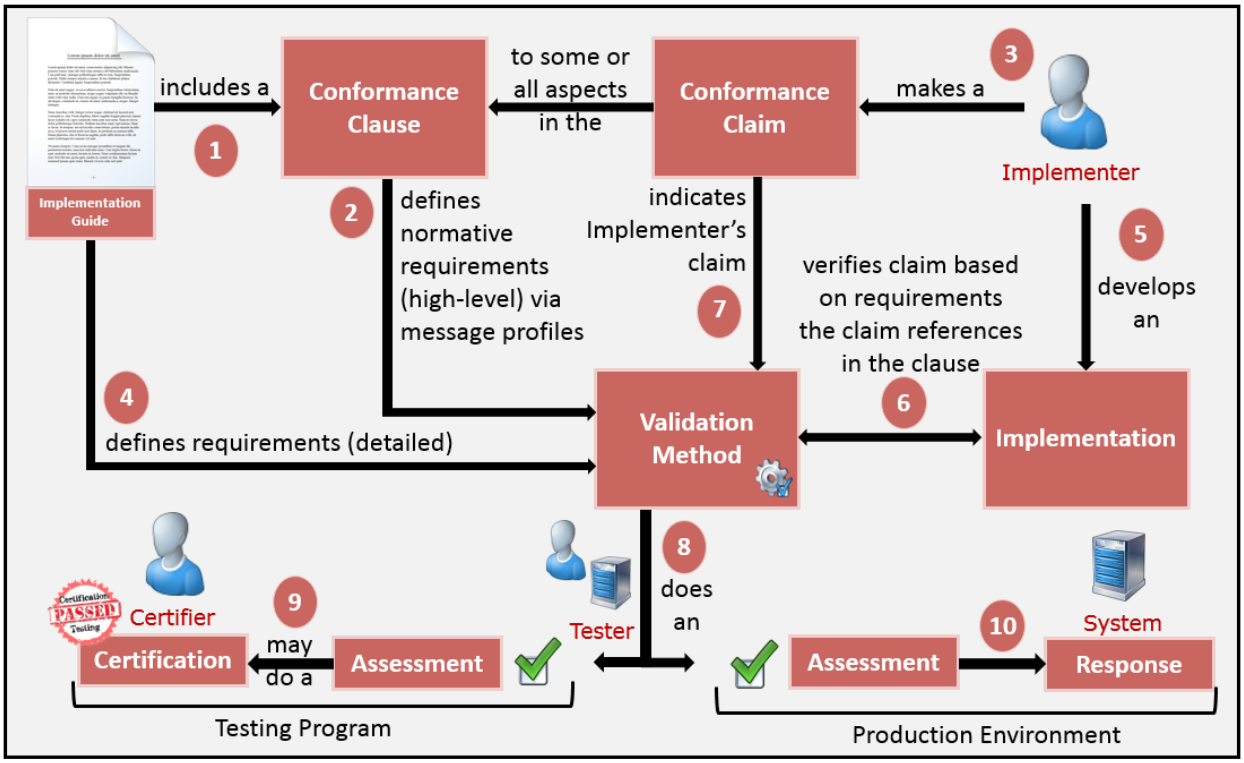

Figure 1.12: Implementation Guides in Context of Use

1.5.1 VALIDATION AND THE RESPOPNSE TO VALIDATION RESULTS

The conformance methodology specification provides the procedures and mechanisms that allow specifiers to define requirements precisely in the form of profiles. Those requirements provide the basis for evaluating implementations. A conformance violation occurs when an implementation does not satisfy a requirement; and, it must be emphasized here, the identified conformance violation applies in all circumstances. How entities react to a conformance violation is separate topic.

For the sending system, conformance validation is a strict determination of the content of a message instance against the requirements stated in the message profile. Validation is used as an indicator of the degree to which the message instance conforms to the message profile. In the conformance assessment, no allowance is made for exceptions. Therefore, any content (or absence of content) not explicitly permitted by the message profile is a conformance violation. This statement is not to be confused with receiver processing rules; i.e., the conformance assessment should influence but not necessarily dictate receiver processing behavior. For example, unexpected content such as a Z-segment (not documented in the profile, and not explicitly allowed) is a conformance violation with respect to the message profile; however, the receiver may choose to ignore the violation and process the message.

To further emphasize the point, suppose a profile specifies an address field that has components for street, city, state, and zip code. All components are required. A receiver is sent a message in which the state is omitted. The message is non-conformant; specifically, the required state component is missing, and this omission is a conformance violation. A conformance tool would determine that the message is non-conformant and report the violation. A receiver, however,

would act in accordance to its processing rules. Implementations may use conformance assessments in different ways, for example, as a mechanism to determine how the message content should be processed or whether a message should be processed at all. A viable choice in the case where the state is omitted from the address field would be to process the message and then deduce the correct state based on the zip code and other collaborating evidence. For this type of conformance violation, the receiver may or may not decide to send an acknowledgement with an exception[7] back to the sender. And if the receiver does send such an acknowledgement, that receiver may choose to apply one of several different severity levels (e.g., error, warning, or informational).

Using message profiles is a powerful tool for determining conformance of message instances; however, the appropriate reaction to the conformance assessment findings is dependent on the circumstances. In certification programs, for example, a conformance violation can and should mean a failure, whereas in live interfaces the same violation may result in a spectrum of responses, from rejecting the message to no action whatsoever. Non-conformance is not necessarily an appropriate reason for not accepting usable data. On the other hand, implementations should seek to define their interface as precisely as possible and to adhere to those rules. Well-defined specifications (message profiles) and conformance to those specifications are powerful drivers towards interoperability, which is a key goal of message profiles.

Assessing receiver-side conformance is often a more difficult task, because it is determined by the consumption of the data and the actions taken on that data. In order to test receiving systems adequately, sufficient acknowledgement and functional requirements must be specified for each element. Such specifications are often out-of-scope for HL7 v2 interface specifications. Companion edge functional requirement guides can be developed to further specify what is expected by certain receiver classes when they receive the data. For example, an operational results management application may be required to store "everything", while an analytics/clinical decision support system may opt to only store select data it needs and no more. Acknowledgement requirements may include exception responsibilities when conformance violations, such as a missing required element, are detected. Functional requirement violations can be detected by acquiring evidence that a capability is not supported by the receiving system. Capabilities may include displaying of an element, storing the data in a record, exporting the data to another system, using the data to search for a record, or some other observable processing.

[7] However, there may be explicit acknowledgement rules defined in the implementation guide for this circumstance.

2 PROFILE LEVELS

Profiles innately form a logical hierarchy as levels of constraints are applied to the base standard. That is, profiling is typically an iterative process that proceeds from the base standard to intermediate levels (e.g., a national-level profile), to local (implementation) levels. Three profile levels are defined:

- Standard Profile
- Constrainable Profile
- Implementable Profile

Figure 2.1 illustrates the basic profile-level hierarchy and the degree of constraints applied at each level. The standard-level profile (hereafter called "**standard profile**") represents the base standard definitions and constraints as-is and establishes the framework for a specific type of event (e.g., an unsolicited vaccination record update – VXU V04 message event). Treating the standard as a profile aids in related discussions and management. At this level, the overall structure ("template"), including the data type definitions, is established; however, the full declaration of requirements has yet to be specified, and, therefore, considerable openness still exists. For example, the allowable segments are defined, but the set of segments that are to be included for this use case has not yet been determined.

Additional profiles are subsequently derived from the standard profile. The standard profile can be defined more precisely by adding constraints for a desired use (e.g., the national level of requirements for the unsolicited vaccination record update message) to create a **constrainable profile**. A constrainable profile is derived from either the standard profile or another constrainable profile, and it further constrains the definition attributes in accordance to the profile compliance rules stipulated in this document. For example, a U.S. state-level constrainable profile can be derived from a national-level profile by adding more constraints for a specific state. Note, a U.S. state-level profile could have been created from the standard profile directly.

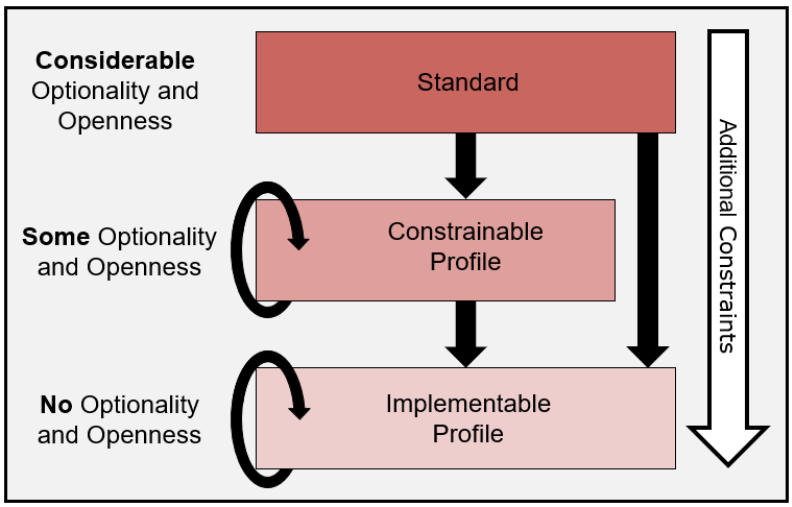

Figure 2.1: Basic Profile-Level Hierarchy

In a constrainable profile, analogous to the standard profile, not all element attributes are fully constrained. An **implementable profile** defines all elements such that all optionality and openness have been removed. All interfaces deployed in a production setting are implementable profiles whether they are documented (explicitly stated) or not documented (implicitly stated). It is recommended that interfaces be completely documented to the implementable profile level using the profiling mechanisms described in this specification.

An implementable profile may also be derived by further constraining another implementable profile. In this case, all openness has been removed, however additional constraints on attributes are applied; for example, the usage of "required, but may be empty" can be *strengthened* to "required" if the refined use case needs it to be.

Constraints are added iteratively, thereby forming a hierarchy of profile levels; but a certain set of rules must be followed. A profile can only further constrain, not relax, the requirements defined in its *parent* profile. For example, if an element (e.g., a field) is "*required*" in the parent profile, the element can't be profiled to "*optional*" in a (compliant) *child* profile, because "*optional*" would be a relaxed requirement relative to "*required*".

Conformity assessment can be conducted for each profile level. A complete assessment of the interface declaring conformance to an implementation profile can be determined. For standard-level and constrainable-level profiles, not all aspects can be determined. Declaring conformance to the base standard is generally of limited use to perspective stakeholders.

2.1 Profiles in Use

Two typical real-world scenarios demonstrating the use of the profile level model are presented in Figure 2.2. In the first case, a national-level constrainable profile is developed. A hospital (group) adopts and refines the national-level guidance provided in the realm-specific constrainable profile. The hospital procures a vendor's product that can be configured to satisfy the requirements. The hospital and the vendor finalize the requirements and the software is installed. The resultant interface is documented as an implementable profile. Alternatively, the hospital could have provided their desired implementable profile directly to the vendor.

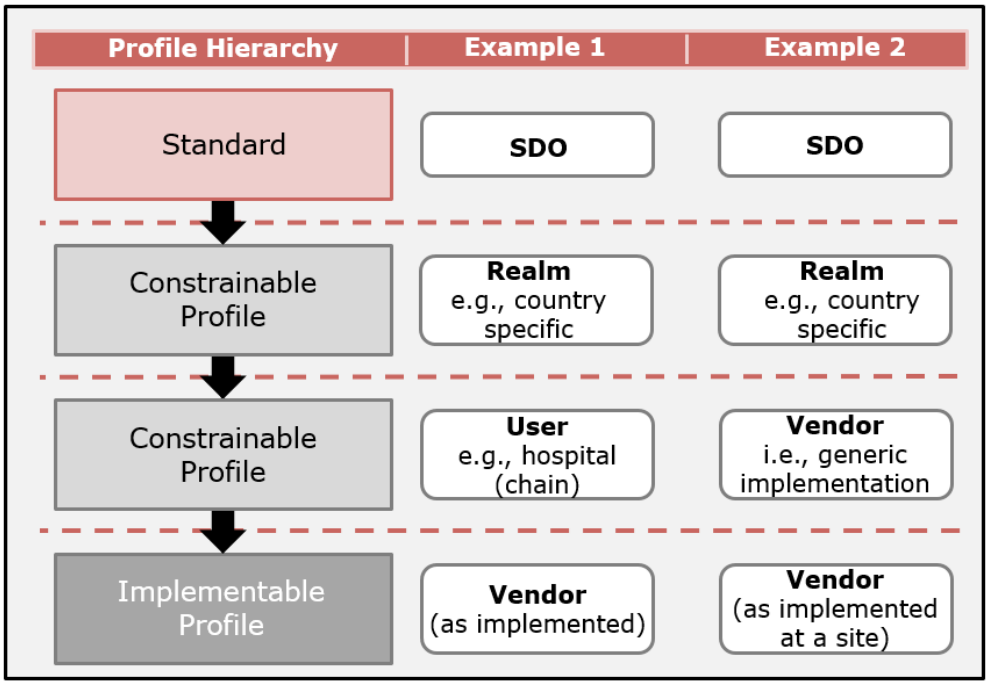

Figure 2.2: Example Profile Hierarchy

In the second case, a vendor refines the national-level guidance profile and provides a generic implementation based on this constrainable profile. When working with clients for whom this profile closely satisfies their requirements, a final refinement is made at the specific sites. The vendor often will (or should) provide the documentation for the installed interface in the form of an implementable profile. The examples shown in Figure 2.2 can be nested and refined to any depth as appropriate.

3 HL7 V2 IMPLEMENTATION GUIDES

HL7 v2 messages are not used in a vacuum; rather, they are part of a larger system designed to satisfy one or more use cases. An HL7 v2 implementation guide is a specification that describes this larger context. An implementation guide[8] is created to organize a collection of message profiles for specifying a set of related interactions described in a use case or use cases. Implementation guides may describe broader conformance requirements such as application functionality. Such requirements may include how a set of messages are to be used to enact certain application functionality among applications (actors).

Broadly speaking, implementation guides are *containers* for message profiles. In other words, implementation guides describe the use case(s), the applications (actors) involved, the message profile(s), and the interaction among the actors; and they also can specify functional requirements. A message profile in the context of a messaging standard describes the requirements for a single interaction (e.g., Send an Unsolicited Immunization VXU V04 message).

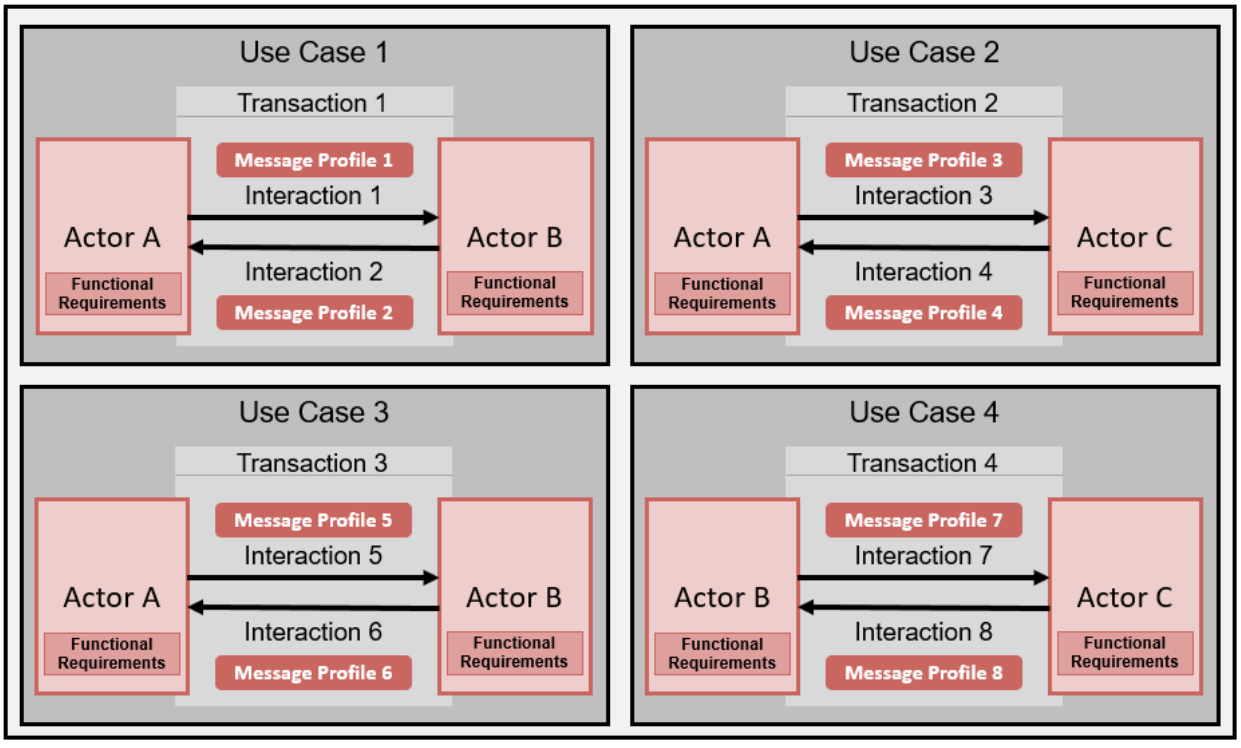

Figure 3.1: Generic Implementation Guide Organization[9]

Figure 3.1 illustrates a generic logical organization of an Implementation Guide. There are no definitive rules for exactly what content is contained in an implementation guide. For example, some implementation guides will focus on interoperability definitions and include limited

[8] IHE (Integrating the Healthcare Enterprises) uses the term Integration Profile

[9] Note the left-hand side of the diagram indicates the case where the Actors are the same but for a different Transaction.

application functional requirements (mostly implied); whereas, others will include functional requirements to a greater degree.

Table 3.1: Immunization Implementation Guide List of Message Profiles

Profile Identifier[10]	Message Definition	Profile	Related Profiles
Z22[11]	VXU V04	Send Immunization History	Z23
Z23	ACK V04	Acknowledgement	Z22
Z24	ADT A31	Send Demographics Only Content	Z23
Z34	QBP Q11	Request Immunization History	Z31 Z32 Z33
Z44	QBP Q11	Request Evaluated History and Forecast	Z42 Z33
Z31	RSP K11	Return a list of Candidates Profile	Z34
Z32	RSP K11	Return an Immunization History Profile	Z34
Z42	RSP K11	Return Evaluated History and Forecast	Z44
Z33	RSP K11	Acknowledgement with No Person Records	Z34 Z44

The "*HL7 Version 2.8.2 IG: Immunization Messaging*" implementation guide is an example of a guide that describes a set of relevant message profiles for a specific domain. Table 3.1 lists the base message definitions and derived profiles included in that guide. The left column of this table indicates the identifier of the profile and the column on the right side lists the valid profile transaction pair(s) for each profile. For example, the profile Z22 – Send Immunization History defines (refines) the requirements for a VXU V04 message **interaction**. When it is paired with the Acknowledgment profile (Z23), a **transaction** is defined. The implementation guide provides use cases, actor definitions, transactions, and limited functional requirements. Each refinement of a message definition (e.g., VXU V04) specifies a **message profile** (e.g., Z22) and corresponds to an interaction. The collection of message profiles and associated use case information define the implementation guide. A message profile identifier or collection of message profile identifiers can be declared in a conformance clause for an application's or product's claim of conformance (e.g., a claim of conformance to a message profile). This claim is indicated in the message instance via the MSH-21 (Message Profile Identifier) element.

Figure 3.2 illustrates a representative implementation guide for a set of related immunization use cases. This diagram depicts a use case (e.g., Send Immunization), the actors involved (i.e., EHR-S and IIS-Immunization Information System), and a set of relevant transactions (e.g., Z22/Z23) and interactions (e.g., Z22-VXU).

[10] The identifier is the HL7 v2.x conformance profile identifier. The CDC implementation guide uses a text identifier.

[11] The letter "Z" in the profile identifier does not imply a "Z" event.

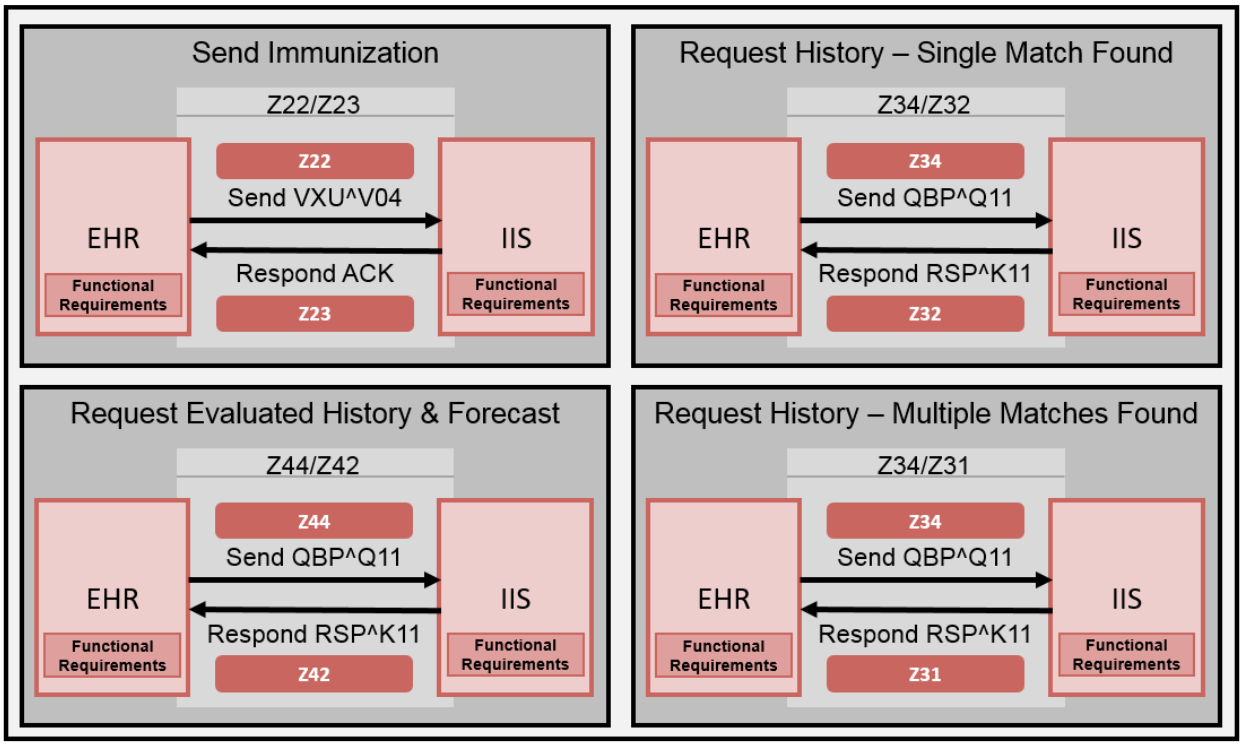

Figure 3.2: Representative Immunization Implementation Guide[12]

As stated in the Conformance Clause (Section 3.1.8), the implementation guide is informative within the context of this specification but highly recommended for documenting an interoperability solution. In addition, certain aspects in an instance of an implementation guide can be made normative by the implementation guide author. For example, an implementation guide can contain functional requirements and declare those as normative requirements; however, the implementation need not contain functional requirements at all.

The remainder of this section provides a sample outline of an HL7 v2 implementation guide. The manner in which the sample implementation guide is organized and a list of what parts it may contain are provided for guidance.

3.1 Background

An implementation guide may contain various sections that provide background information to inform the reader of the overall purpose, scope, conventions, conformance clause, and any other facts the author thinks are pertinent for the use and application of the implementation guide. The following sub-sections provide a list and descriptions of common background sections.

3.1.1 INTRODUCTION

This section is the beginning of the technical content. The Introduction provides basic background information about the implementation guide, including the problem statement, anticipated stakeholders, sponsors, and scope. Additional sub-sections (listed below) may include

[12] This diagram does not indicate all possible use cases and transactions described in the implementation guide. A sample of the most relevant use cases is provided to illustrate the organization of the guide.

the purpose, intended audience, organization of the guide, reference profiles, scope, and key technical decisions or conventions.

3.1.2 PURPOSE

The purpose can describe the intended use of the implementation guide.

3.1.3 INTENDED AUDIENCE

This section provides information about the stakeholders who may have an interest in the implementation guide. This audience can include implementers, providers, hospitals, and records offices, such as public health registries. The section may indicate prerequisite knowledge requirements. Additionally, it may indicate any relationship to a regulation or certification program.

3.1.4 SCOPE

The scope provides, from a high-level perspective, the use cases the implementation guide is covering and the boundaries within the use cases. This section could include an "extended" workflow and indicate the portions of the workflow covered in the implementation guide ("in-scope"). The section also can include aspects that are not covered in the implementation guide ("out-of-scope"). The scope is intended to give the reader the appropriate context for interpreting the requirements.

3.1.5 ORGANIZATION OF THIS GUIDE

This section informs the reader about the layout and design of the implementation guide. It is intended to help the reader navigate the document. Implementation guides may present the profiles in different ways. One way may be to define the building block components, such as the data type and value set definitions, separately and then describe the message definitions that reference these components. Alternatively, a message profile section contains all of the components relevant for its definition. Using this method, components typically are duplicated in the implementation guide. Other formats may exist, for example, specifying a differential profile while referencing the base profile.

3.1.6 REFERENCE SPECIFICATIONS—ANTECEDENTS

This section refers to and/or describes implementation guides or other specifications or artifacts related to this implementation guide, including previous versions or publication history of this implementation guide where applicable. This section may also point to regulations that influence the implementation guide.

3.1.7 CONVENTIONS

An implementation guide should identify the mechanisms used to specify the included requirements. This document serves as the template for creation of the requirements, as most of the requirements must be specified via the conformance constructs; however, additional narrative-type requirements regarding the implementation guide, profiles, or functional requirements may need to be defined. In such instances, a set of standardized conformance keywords (also referred to as conformance verbs) must be used. The conformance keywords help define the requirements and testable criteria more precisely.

It is recommended that specifications adopt, in principle, the *Key words for use in RFCs to Indicate Requirement Levels* (commonly known as RFC 2119). Depending on the type of specification and its purpose, a subset of the keywords may be appropriate. Below is an example of a subset of the keyword definitions extracted from RFC 2119:

> **KEYWORDS:**
>
> The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in RFC 2119. The following definitions are excerpted from the RFC:
>
> **MUST** or the terms "**REQUIRED**" or "**SHALL**" means that the definition is an absolute requirement of the specification.
>
> **MUST NOT** or the phrase "**SHALL NOT**" means that the definition is an absolute prohibition of the specification.
>
> **SHOULD** or the adjective "**RECOMMENDED**" means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
>
> **SHOULD NOT** or the phrase "**NOT RECOMMENDED**" means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood, and the case carefully weighed, before implementing any behavior described with this label.
>
> **MAY** or the adjective "**OPTIONAL**" means that an item is truly optional. One software supplier may choose to include the item to enable certain capabilities while another software supplier may omit the same item. In either case, the communication partner cannot be expected to either provide it (sender) or process it (receiver) without clear and voluntary agreement between the partners.

3.1.8 CONFORMANCE CLAUSE

An implementation guide should contain a conformance clause that defines the requirements, criteria, or conditions that must be satisfied by an implementation to claim conformance to the specification. The conformance clause identifies what must conform and how conformance can be met. This information typically is provided via a list of profiles[13] or use cases that contain the profiles.

The conformance clause also can describe options and levels. The conformance clause will state which profiles *must* be implemented and which profiles *may* be implemented. The option to implement the same requirements but at different difficulty levels also may be allowed, which is useful when the specifiers want to ease the community into adoption of a difficult requirement by providing a preview of the sought-after implementation without mandating its immediate adoption. Implementers can use the conformance clause to make a claim related to the options and levels they selected.

The conformance clause will indicate the normative and informative parts of the specification. Normative parts of the implementation guide contain requirements that must be implemented (and, therefore, be verifiable). Informative parts of an implementation guide provide guidance

[13] Including core profiles, profile components, and composite profiles.

that aids in implementation or provide supplementary information. No provisions in the informative part of a specification need to be implemented.

3.2 Actors

An **Actor** is an information system or a component of an information system that produces, manages, or acts on information associated with operational activities in an organization. An actor has a specific role, performs specific functions, and is defined in an abstract manner. Actors exchange information through standard HL7 v2 messages and can be an initiator or a responder. An **Initiator** is an entity that starts an action and is also referred to as a **Sender, Producer,** or **Client**. A **Responder** is an entity that is reacting to the action of the Initiator. A responder is also referred to as a **Receiver, Consumer,** or **Server**.

3.3 Use Case

A **Use Case** describes a system capability and the associated sequences of actions that a system performs to achieve an intended outcome. The use case can include the actors involved (including the sending and receiving systems); the interactions (including the message exchanges) between the actors; system functional requirements, acknowledgement requirements, and pre- and post-conditions; and any other pertinent information that aids in the understanding of an implementer, such as example messages or message excerpts.

The use case describes messaging requirements at a high-level and references detailed requirements specified in the Messaging Infrastructure. An implementation guide will typically contain more than one use case. A use case can be considered a *container* that organizes the pieces needed to specify a system capability. The following sub-subsections provide a list of candidate content of a use case definition.

3.3.1 INTERACTION SPECIFICATION

The Interaction Specification describes the dynamic aspects of a system. In HL7 v2, an interaction specification typically will include the sequence of trigger events, the resulting message flows between two or more actors (Interaction Model), and any acknowledgement responsibilities. The Interaction Specification can be represented in literal or graphical form. Graphical form can include standard UML interaction diagrams (i.e., sequence and collaboration diagrams) and/or activity diagrams. An interaction specification must be defined in an implementation guide for relevant uses cases and must include an interaction model and acknowledgement responsibilities; however, this specification does not prescribe a specific method or format for doing so.

3.3.1.1 INTERACTION MODEL

An interaction model in the context of an HL7 v2 system shows the actors and the interactions (messages) between them. Figure 3.3 shows a simple sequence diagram of the message flow for the Send Immunization Event use case. The initiating system creates immunization messages and sends one or more immunization events in a VXU message. The VXU message declares conformance to the message profile definition indentified as ‘Z22’. The responding system accepts the message, processes it, and returns an acknowledgement message with a profile definition indicated by the ‘Z23’ identifier.

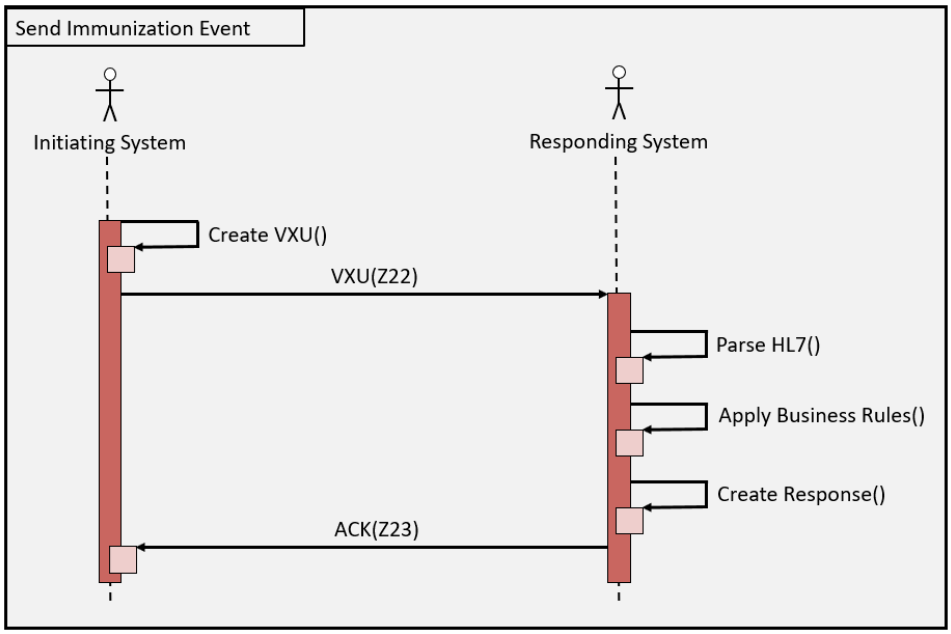

Figure 3.3: Example Send Immunization Event Sequence Diagram

3.3.1.2 ACKNOWLEDGEMENT RESPONSIBILITIES

For each interaction, the specific HL7 v2 acknowledgements required and/or allowed for use with a message profile must be defined. Specifically, the acknowledgement responsibilites must identify whether an **accept** and/or **application** level acknowledgement is allowed or required.

The acknowledgement responsibilites must define the conditions under which an accept and/or application level acknowledgement is expected.

Allowed conditions include:

- Always
- Never
- Only on success
- Only on error

For a given interaction, multiple acknowledgement outcomes may be defined. Figure 3.4 shows an activity diagram for the acknowledgement responsibilities of the receiver in reaction to the processing of the Send Immunization Event.

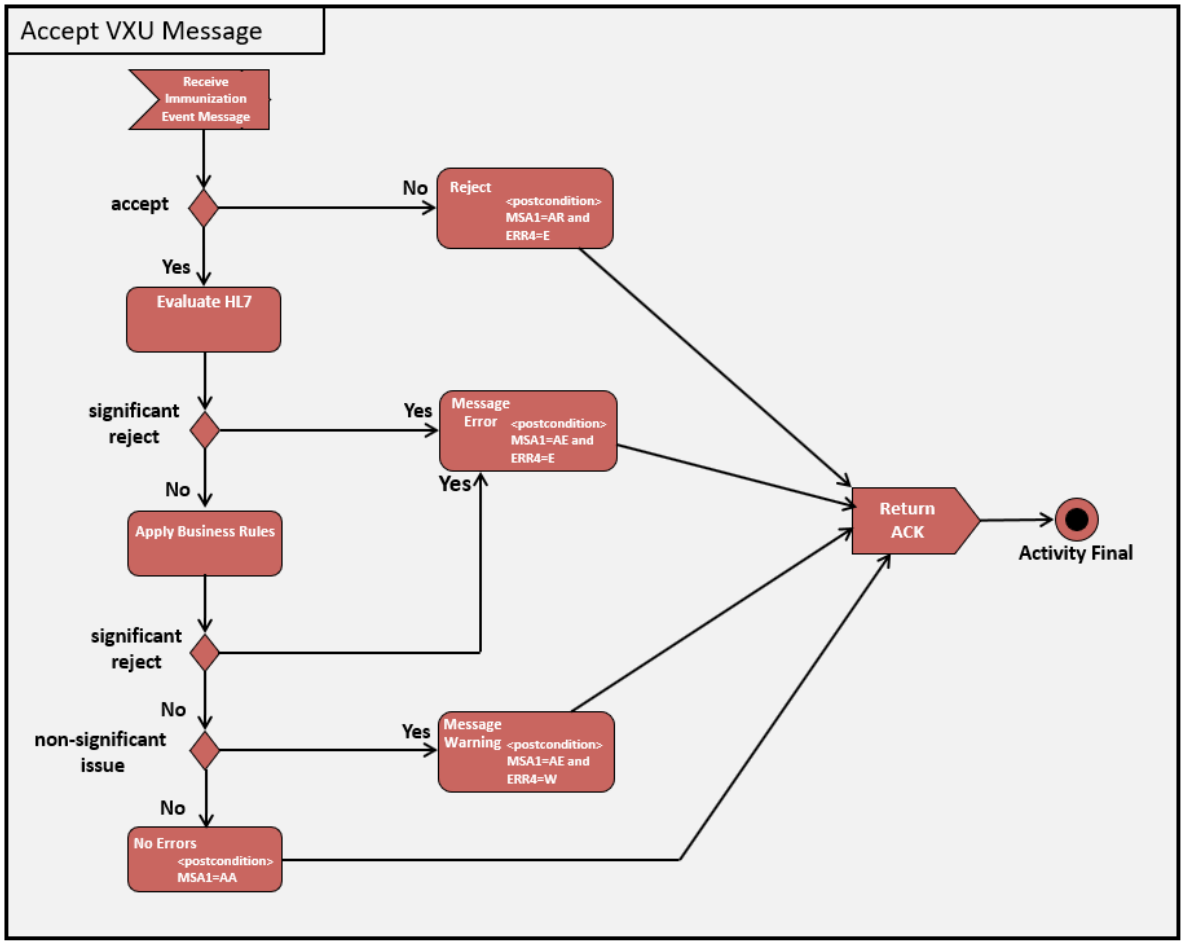

Figure 3.4: Acknowledgement Responsibilities Example

3.3.2 FUNCTIONAL REQUIREMENTS

Functional requirements are a set of requirements that describes the functions and operations of an application (or actor). They provide a defined set of capabilities of the application and are also referred to as business rules. Functional requirements may be included in HL7 v2 implementation guides and may be defined for each element or a collection of elements necessary to conduct a given operation. The extent to which functional requirements are included in implementation guides varies; typically, their inclusion is, and should be, minimal. A separate functional requirements specification allows for a protocol-independent specification that can change irrespective of the interface specification.

3.3.3 EXAMPLE MESSAGES

Example messages, or excerpts of them, may be included in implementation guides to provide insight into and expected outcomes of the message profile for a specific use case interaction. Implementers can refer to the example messages to gain understanding, but they should not infer that the examples are comprehensive and use them as the sole basis for implementation.

3.4 Messaging Infrastructure

The messaging infrastructure part of an implementation guide provides the specification of the messaging requirements. Messaging requirements are defined in the message profiles that include the message, segment, data type, and vocabulary definitions. An HL7 v2 implementation guide must include the messaging infrastructure.

3.5 Other Relevant Information

An implementation guide may contain any other relevant information to aid developers in implementations. This information may include:

- known reference implementation
- conformance test tools
- relationship to certification programs
- related standards

4 MESSAGE PROFILES

A message profile is based on a message structure defined in the HL7 v2 standard and specifies refined requirements according to its intended context of use. The message profile identifies the message code (e.g., ADT), trigger event (e.g., A04), message structure (e.g., ADT_A01), and other meta data as shown in Table 10.2. Constraints are placed on objects at various levels in the message template. Table 4.1 summarizes the types of constraints and the message objects to which they apply. In the sub-sections to follow, the constraint types that apply at each level of profiling are explained. In Section 5 (Constraints), each constraint type is described in detail.

Table 4.1: Summary of Constraint Types and Where They Apply

Constraint	M	SG	S	F		C		SC
				Cx	P	Cx	P	P
Usage		√	√	√	√	√	√	√
Cardinality		√	√	√	√			
Data Type Specialization				√	√*	√	√*	√*
Vocabulary				√	√	√	√	√
Length					√		√	√
Conformance Length					√		√	√
Content					√		√	√
Conformance Statement	√	√	√	√	√	√	√	√
Co-constraints			√					
Slicing				√				
Semantic Refinement	√	√	√	√	√	√	√	√

Key				
	M: Message	SG: Segment Group	S: Segment	F: Field
	C: Component	SC: Sub-component	Cx: Complex	P: Primitive

*Typically (and in most cases) a data type specialization is not made for primitive elements (fields, components, sub-components). However, data types like DTM that have specialization "encoding" of the date/time format YYYYMMDDHHSS.SSSS+-ZZZZ are the exceptions.

Although all elements allow for semantic refinement, caution should be used in certain circumstances. For example, components within a data type generally have very specific meaning and rarely should be a candidate for semantic refinement.

Components and sub-components are not directly constrained, rather, they are constrained within the context of a data type flavor. Vocabulary bindings to elements within a data type can be assigned in the context of use. Vocabulary constraints only apply to elements that have data types that have coded data types. Fields and components can be complex or primitive elements. Sub-components are always primitive elements. As shown in Table 4.1, some of the concepts are the same depending on the context. For example, fields and components are primitive elements in some circumstances.

Segments are profiled at the message level. Fields are profiled at the segment level. For data types, components and sub-components are profiled in the context of data type flavors.

4.1 Message Profile Identification

To simplify profile references and claims of conformance, an identification mechanism for HL7 v2 is provided by the message profile identifier. The message profile identifier is referenced in two places that provide the bridge between the message profile and the message instance:

- In the message profile meta data, indicated by the MessageProfileIdentifier element
- In the message instance, indicated by the MSH-21 field (Message Profile Identifier)

The MessageProfileIdentifer element provides the identifier for the message profile that can be referenced in a message instance. This reference in the message instance is a claim by the sending system of conformance to the message profile it references. The receiver, via the conformance claim, is made aware of the expectant message content (as defined in the message profile). The receiver may validate the message content based on the requirements given in the message profile and make processing decisions based on the outcome of that validation. Additionally, validation tools conduct conformance testing based on the message instance and the conformance claim indicated by the message profile identifier. A receiver can publish its claim of conformance to the message profile in its interface documentation or other capability statement.

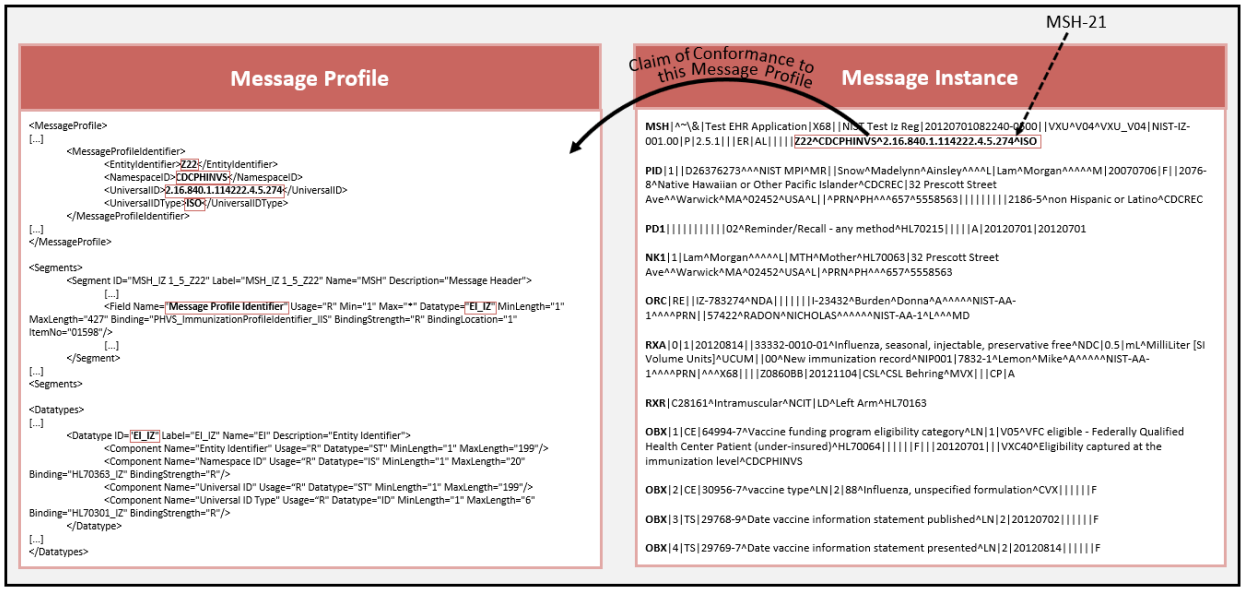

Figure 4.1: Message Profile Identifier Mechanism

The left-hand side of Figure 4.1 shows the message profile identifier as meta data that models the Entity Identifier (EI) data type. The field definition for MSH-21 is contained in the body of the message profile; the definitions in the meta data and the field must be compatible. The right-hand side of Figure 4.1 shows a message instance in which MSH-21 is claiming conformance to the message profile definition shown on the left-hand side of Figure 4.1.

The message profile identifier is not limited to just the message profile; it can reference any of the following profile building blocks:

- message profile (including a pre-coordinated profile identifier)

- profile component (including the core profile and differential component)
- composite profile
- value set library

In principle, all of these items can be managed the same way. The message profile identifier can be a list of any of these profile building blocks. For example, the specifier may wish to separate the message profile and the value set library. The message profile identifier in this case will contain two occurrences. The totality of both occurrences indicates the claim of conformance. Likewise, a complete profile can be described as a collection of profile components. For example, three profile components can be indicated in which one of the components is the core profile and the others are profile components. See Section 7.1 for more details on how profiles can be constructed.

The EI data type consists of four components. EI.1 indicates the identifier of the profile artifact. EI.2 or (EI.3 and EI.4) establishes the assigning authority for the identifier given in EI.1. This scheme gives the identifier uniqueness. EI.2 describes the common name of what is defined by EI.3 and EI.4 (and vice-versa). EI.2 alone, or EI.3 and EI.4, or all three elements can be used to specify the assigning authority of the identifier. If all three are defined, then EI.2 and the combination of EI.3 and EI.4 must refer to an equivalent concept. Typically, EI.3 is an OID that defines the assigning authority of the identifier.

Below is an example that shows the message profile identifier definition for a profile that consists of a national level profile with a separate identifier for the value set library. Together the two identifiers indicate the national-level requirements for a message. Additionally, the example shows that a state (local) entity further constrained the profile for their refined use case. Together, all three identifiers indicate the complete set of requirements. The example also shows how a system could value MSH-21 in the message instance to convey the content of the message.

Profile Definition:

```
<MessageProfile>
  <CoreProfileID>
    <EntityIdentifier>Profile_National_Level</EntityIdentifier>
    <NamespaceID>Domain_XYZ</NamespaceID>
    <UniversalID>1.2.3.4.5</UniversalID>
    <UniversalIDType>ISO</UniversalIDType>
  </CoreProfileID>
  <ValueSetLibraryID>
    <EntityIdentifier>VS_Library</EntityIdentifier>
    <NamespaceID>Domain_XYZ</NamespaceID>
    <UniversalID>1.2.3.4.5</UniversalID>
    <UniversalIDType>ISO</UniversalIDType>
  </ValueSetLibraryID>
  <ProfileComponentID>
    <EntityIdentifier>State_Profile _Component</EntityIdentifier>
    <NamespaceID>Domain_ABC</NamespaceID>
    <UniversalID>1.2.3.4.6</UniversalID>
    <UniversalIDType>ISO</UniversalIDType>
  </ProfileComponentID>
```

```
</MessageProfile>
```

Message Instance:

```
Profile_National_Level^Domain_XYZ^1.2.3.4.5^ISO~VS_Library^Domai
n_XYZ^1.2.3.4.5^ISO~State_Profile_Component^Domain_ABC^1.2.3.4.6
^ISO
```

The above example illustrates the grouping of three profile building blocks with an implied AND connector. The message profile identifier mechanism also supports an implied OR connector with the PreCoordinatedProfile element. For example, the above illustration also could have defined a single pre-coordinated identifier that refers to these three profile building blocks. See the illustration below.

Profile Definition:

```
<MessageProfile>
  <PreCoordinatedProfile>
    <PreCoordinatedProfileID>
      <EntityIdentifier>State_Profile</EntityIdentifier>
      <NamespaceID>Domain_ABC</NamespaceID>
      <UniversalID>1.2.3.4.6</UniversalID>
      <UniversalIDType>ISO</UniversalIDType>
    </PreCoordinatedProfileID>
    <CoreProfileID>
      <EntityIdentifier>Profile_National_Level</EntityIdentifier>
      <NamespaceID>Domain_XYZ</NamespaceID>
      <UniversalID>1.2.3.4.5</UniversalID>
      <UniversalIDType>ISO</UniversalIDType>
    </CoreProfileID>
    <ValueSetLibraryID>
      <EntityIdentifier>VS_Library</EntityIdentifier>
      <NamespaceID>Domain_XYZ</NamespaceID>
      <UniversalID>1.2.3.4.5</UniversalID>
      <UniversalIDType>ISO</UniversalIDType>
    </ValueSetLibraryID>
    <ProfileComponentID>
      <EntityIdentifier>State_Profile _Component</EntityIdentifier>
      <NamespaceID>Domain_ABC</NamespaceID>
      <UniversalID>1.2.3.4.6</UniversalID>
      <UniversalIDType>ISO</UniversalIDType>
    </ProfileComponentID>
 <PreCoordinatedProfile>
</MessageProfile>
```

Message Instance:

```
State_Profile^Domain_ABC^1.2.3.4.6^ISO
```

OR

```
Profile_National_Level^Domain_XYZ^1.2.3.4.5^ISO~VS_Library^Domai
n_XYZ^1.2.3.4.5^ISO~State_Profile_Component^Domain_ABC^1.2.3.4.6
```

```
^ISO
```

This example shows a definition of a pre-coordinated profile and its constituent parts. Systems can convey the message content in MSH-21 by either indicating the pre-coordinated identifier or the three identifiers for the parts of the profile. Combining profile parts can continue indefinitely to a practical limit. For example, a profile component can be added to the pre-coordinated profile in the previous example to address a different, but closely related, use case. The identifiers can be specified separately, or another pre-coordinated identifier could be created.

4.2 Message Level Profiling

The profiling operations at the message level include:

- The HL7 abstract message syntax must be used to document the message-structured definition. The message-structured definition must include the list of segments as given in the base standard with their associated optionality ([]) and repeatability ({}) symbols.
- A list of segment flavors may be specified with their modified associated optionality ([]) and repeatability ({}) symbols,
- For each segment in the message-structured definition, the usage must be defined using a usage code as specfied in Section 5.1. For any segment defined with a declared conditional usage, an explicit condition predicate must be defined.
- For each segment group (begin) in the message-structured definition, the usage must be defined using a usage code as specfied in Section 5.1. For any segment group (begin) defined with a declared conditional usage, an explicit condition predicate must be defined.
- For each segment in the message-structured definition, the cardinality must be defined using the cardinality rules as specfied in Section 5.2.
- For each segment group (begin) in the message-structured definition, the cardinality must be defined using the cardinality rules as specfied in Section 5.2.
- Message-level conformance statements may be specified and, if specified, must be defined using the conformance statements rules as specified in Section 5.7 and should use the language defined in Appendix B.

Table 4.2 illustrates an abbreviated specification of a message definition as given in the base standard. Table 4.3 shows an example of a profiled version of that message definition.

Table 4.2: Example of Base Standard Message Level Definition

Segments	Description	Status	Chapter
MSH	Message Header Segment		2
[{ SFT }]	Software		2
[UAC]	User Authentication Credential		2
PID	Patient Identification Segment		3

Segments	Description	Status	Chapter
[PD1]	Additional Demographics		3
[{ NK1 }]	Next of Kin/Associated Parties		3
...			
[{	--- ORDER begin		
ORC	Common Order		4
[{PRT}]	Participation (for ORC)		7
[{	--- TIMING begin		
TQ1	Timing/Quantity		4
[{ TQ2 }]	Timing/Quantity Order Sequence		4
}]	--- TIMING end		
RXA	Pharmacy Administration Segment		4A
[RXR]	Pharmacy Route		4A
[{	--- OBSERVATION begin		
OBX	Observation/Result		7
[{ PRT }]	Participation (for Observation)		7
[{ NTE }]	Notes (Regarding Immunization)		2
}]	--- OBSERVATION end		
}]	--- ORDER end		

The **Segment column** represents the abstract message definition as defined in the base standard not the profiled definition. Specifiers may choose to represent the segment columns differently but must maintain the original list of segments. A second segment column may be added to indicate that a segment flavor has been defined. The specifier may choose to mark in bold text segments that need to be supported. A segment flavor indicates that the segment definition has been constrained. The segment-flavor identifier provides a convenient mechanism to manage and reference segments. If the segment was not constrained, then the base standard segment identifier is given.

The **Usage** column must be included and reflects the usage of the segment or segment group for this message-structured definition.

The **Cardinality** column must be included and reflects the minimum and maximum number of occurrences allowed for the segment or segment group for the message-structured definition.

Additional column headings, such as Referenced Chapter or Comments, may be specified. Conformance Statements and Condition Predicates (if any) also may be included as a column heading or be specified as separate tables immediately following the message-structured defintion table. Table 4.4 shows conformance statement defintions.

Table 4.3: Example of Profiled Message Level Definition

Segment	Segment Flavor	Description	Usage	Cardinality	Chapter
MSH	**MSH_IZ**	Message Header Segment	R	[1..1]	2
[{SFT}]	SFT	Software	X	[0..0]	2
[UAC]	UAC	User Authentication Credential	O	[0..1]	2
PID	**PID_IZ**	Patient Identification Segment	R	[1..1]	3
[PD1]	**PD1_IZ_01**	Additional Demographics	RE	[0..1]	3
[{NK1}]	**NK1_IZ**	Next of Kin/Associated Parties	RE	[0..1]	3
...					
[{		--- ORDER begin	R	[1..*]	7
ORC	**ORC_IZ_01**	Common Order	R	[1..1]	2
[{PRT}]	PRT	Participation (for ORC)	O	[0..1]	3
[{		--- TIMING begin	O	[0..1]	3
TQ1	TQ1	Timing/Quantity	R	[1..1]	7
[{TQ2}]	TQ2	Timing/Quantity Order Sequence	O	[0..*]	7
}]		--- TIMING end			
RXA	**RXA_IZ_01**	Pharmacy Administration Segment	R	[1..1]	4
[RXR]	**RXR_IZ**	Pharmacy Route	RE	[0..1]	4
[{		--- OBSERVATION begin	RE	[0..*]	
OBX	**OBX_IZ_02**	Observation/Result	R	[1..1]	7
[{PRT}]	PRT	Participation (for Observation)	O	[0..*]	7
[{NTE}]	NTE	Notes (Regarding Immunization)	O	[0..1]	3
}]		--- OBSERVATION end			
}]		--- ORDER end			

Table 4.4: Example of Profiled Conformance Statement – Message Level

Message Level Conformance Statements	
IZ-205	If OBX-3.1 (Identifier) contains the value '59785-6' (Indication for Immunization) then RXA-20 (Completion Status) in the same Order group SHALL contain one of the values in the list: {CP, PA}.
IZ-206	If OBX-3.1 (Identifier) contains the value '30945-0' (Vaccination contraindication/precaution) then RXA-20 (Completion Status) in the same Order group SHALL contain the value 'NA'.

4.3 Segment Level Profiling

Segment-level profiling constrains the field elements within a segment. When profiling at the segment-level, the structured definition must be documented using an HL7 segment attribute table format. For each field in the segment definition, the attributes shown in the list below must or can be constrained, and, if constrained, must be defined using the rules as specified in Section 5.

- Must contain the field sequence identifier.
- Must contain the list (name) of fields as given in the base standard.
- Must contain the usage of the field. For any field defined with a declared conditional usage, an explicit condition predicate must be defined.
- Must contain the minimum and maximum cardinality of the field.
- Must contain the base data type or a data type specialization of the field. May be ommitted for non-supported elements.
- Must contain a vocabulary specification for fields with coded element data types. It is recommended that base-level vocabulary bindings remain as is for optional (O) elements and be removed for unsupported (X) elements.
- May contain the minimum and maximum length for a primitive field.
- May contain the conformance length for a primitive field.
- Must not contain both a conformance length and a minimum and maximum length pair for a primitive field.
- Must not specify a minimum and maximum length for a complex field.
- Must not specify the conformance length for a complex field.
- May specify a content constraint (Fixed Value, Pattern Restricted, Arbitrary Data Values) for a primitive field.
- May specify conformance statements and associate to a field.
- May specify a set of co-constraint associated with the segment (for dependent fields).
- May specify a slicing definition for a field.
- May refine the semantic definition of a field.
- If a segment occurs multiple times in an abstract message definition, it may be represented by different segment profiles (i.e., segment flavors). Each segment flavor must be explicitly associated in the message level definition (profile).

Table 4.5 shows the RXA segment definition as it appears in the HL7 v2.8.2 standard. Table 4.6 shows an example of how the profiled segment definition (RXA_IZ_01) can be displayed. The segment flavor (specialization) identifier (RXA_IZ_01) is used in the message definition to indicate that this specialization of the RXA segment is to be used.

Table 4.5: Example Base Segment Definition – RXA Segment

SEQ	LEN	C.LEN	DT	OPT	RP/#	TBL#	ITEM #	ELEMENT NAME
1		4=	NM	R			00342	Give Sub-ID Counter
2		4=	NM	R			00344	Administration Sub-ID Counter
3			DTM	R			00345	Date/Time Start of Administration
4			DTM	R			00346	Date/Time End of Administration
5			CWE	R		0292	00347	Administered Code
6			NM	R			00348	Administered Amount
7			CWE	C		9999	00349	Administered Units
8			CWE	O		9999	00350	Administered Dosage Form
9			CWE	O	Y	9999	00351	Administration Notes
10			XCN	B	Y		00352	Administering Provider
11				W			00353	Administered-at Location
12		20=	ST	C			00354	Administered Per (Time Unit)
13			NM	O			01134	Administered Strength
14			CWE	O		9999	01135	Administered Strength Units
15		20=	ST	O	Y		01129	Substance Lot Number
16			DTM	O	Y		01130	Substance Expiration Date
17			CWE	O	Y	0227	01131	Substance Manufacturer Name
18			CWE	O	Y	9999	01136	Substance/Treatment Refusal Reason
19			CWE	O	Y	9999	01123	Indication
20	2..2		ID	O		0322	01223	Completion Status
21	1..1		ID	O		0206	01224	Action Code – RXA
22			DTM	O			01225	System Entry Date/Time
23		5=	NM	O			01696	Administered Drug Strength Volume
24			CWE	O		9999	01697	Administered Drug Strength Volume Units
25			CWE	O		9999	01698	Administered Barcode Identifier
26	1..1		ID	O		0480	01699	Pharmacy Order Type
27			PL	O			02264	Administer-at
28			XAD	O			02265	Administered-at Address

The specifier has chosen to present the segment flavor definition with a field sequence identifier, element name, data type (that includes any flavor), usage, cardinality, vocabulary binding (including concept domain[14], code system[15], or value set binding), minimum and maximum length, conformance length, and comments. If content constraints (e.g., fixed values) existed, a column for them could have been added.

Table 4.6: Sample Profile Segment Definition – RXA Segment (RXA_IZ)

SEQ	ELEMENT NAME	Data Type	Usage	Card.	Vocab	LEN	C.LEN	Comments
1	Give Sub-ID Counter	NM	R	1..1			1	
2	Administration Sub-ID Counter	NM	R	1..1			1	
3	Date/Time Start of Administration	DTM_IZ02	R	1..1				
4	Date/Time End of	DTM_IZ02	R	1..1				

[14] Only applicable for standard and constrainable profile levels.

[15] Only applicable for standard and constrainable profile levels.

SEQ	ELEMENT NAME	Data Type	Usage	Card.	Vocab	LEN	C.LEN	Comments
	Administration							
5	Administered Code	CWE_IZ01	R	1..1	CVX_01			
6	Administered Amount	NM	R	1..1		1..16		
7	Administered Units	CWE_IZ01	C(R/X)	0..1	UCUM_01			
8	Administered Dosage Form	CWE	O	0..1				
9	Administration Notes	CWE_IZ01	C(R/O)	0..1	NIP001_01			
10	Administering Provider	XCN_IZ01	C(RE/O)	0..1				
11	Administered-at Location		X	0..0		0..0		
12	Administered Per (Time Unit)	ST	C	0..1				
13	Administered Strength	NM	O	0..1				
14	Administered Strength Units	CWE	O	0..1	9999			
15	Substance Lot Number	ST	C(R/O)	0..*			20=	
16	Substance Expiration Date	DTM_IZ03	C(RE/O)	0..1				
17	Substance Manufacturer Name	CWE_IZ01	C(R/O)	0..1	MVX			
18	Substance/Treatment Refusal Reason	CWE_IZ01	C(R/X)	0..*	9999			
19	Indication	CWE	O	0..1	9999			
20	Completion Status	ID	RE	0..1	0322	2..2		
21	Action Code – RXA	ID	C(R/X)	0..1	0206	1..1		
22	System Entry Date/Time	DTM	O	0..1				
23	Administered Drug Strength Volume	NM	O	0..1				
24	Administered Drug Strength Volume Units	CWE	O	0..1	9999			
25	Administered Barcode Identifier	CWE	O	0..1	9999			
26	Pharmacy Order Type	ID	O	0..1	0480			
27	Administer-at	PL_IZ01	C(RE/O)	0..1				
28	Administered-at Address	XAD	O	0..1				
29	Administered Tag Identifier	EI	O	0..1				

Table 4.7 shows the list of condition predicates associated with fields with declared conditional usage. The table indicates the location of the conditional usage, the usage, and the predicate. An example (and recommended) predicate language is given in Appendix A.

Table 4.7: Example of Profiled Condition Predicate – Segment Level

Condition Predicates		
Location	**Usage**	**Predicate**
RXA-7	C(R/X)	If RXA-6(Administered Amount) does not contain the value '999'.
RXA-9	C(R/O)	If RXA-20(Completion Status) contains one of the values in the list: {CP,PA}.
RXA-10	C(RE/O)	If RXA-9.1(Identifier) contains the value '00' AND RXA-20(Completion Status) contains one of the values in the list: {CP,PA}.

Condition Predicates		
RXA-15	C(R/O)	If RXA-9.1(Identifier) contains the value '00' AND RXA-20(Completion Status) contains one of the values in the list: {CP,PA}.
RXA-16	C(RE/O)	If RXA-9.1(Identifier) contains the value '00' AND RXA-20(Completion Status) contains one of the values in the list: {CP,PA}.
RXA-17	C(R/O)	If RXA-9.1(Identifier) contains the value '00' AND RXA-20(Completion Status) contains one of the values in the list: {CP,PA}.
RXA-18	C(R/X)	If RXA-20(Completion Status) contains the value 'RE'.
RXA-21	C(R/X)	If RXA-5.1(Identifier) does not contain the value '998'.
RXA-27	C(RE/O)	If RXA-9.1(Identifier) does contain the value '00' AND RXA-20(Completion Status) does contain one of the values in the list: {CP,PA}.

Table 4.8 shows the list of conformance statements associated with the segment. Appendix B provides an example (and recommended) conformance statement language.

Table 4.8: Example of Profiled Conformance Statement – Segment Level

Conformance Statements	
IZ-207	If RXA-20 (Completion Status) contains the value 'RE' then RXA-5.1 (Identifier) SHALL NOT contain the value '998'.
IZ-30	RXA-4 (Date/Time End of Administration) SHALL be identical RXA-3(Date/Time Start of Administration).
IZ-32	If RXA-18 (Substance/Treatment Refusal Reason) is valued then RXA-20 (Completion Status) SHALL contain the value 'RE'.
IZ-48	If RXA-20 (Completion Status) contains the value 'RE' then RXA-6 (Administered Amount) SHALL contain the value '999'.
IZ-49	If RXA-5.1 (Identifier) contains the value '998' then RXA-6 (Administered Amount) SHALL contain the value '999'.
RXA_IZ_01-1	RXA-1(Give Sub-ID Counter) SHALL contain the value '0'.
RXA_IZ_01-2	RXA-2(Administration Sub-ID Counter) SHALL contain the value '1'.

Semantic redefinement clarifies the use of the field within the context of the use case. Best practice is to provide further insight and not to repeat the definition in the base standard. Below is an excerpt of an example semantic refinement.

…

RXA-3: Date/Time Start of Administration (DTM_IZ02)

The date this vaccination occurred. In the case of a contraindication, refusal, or deferral, this is the date the action occurred.

RXA-4: Date/Time End of Administration (DTM_IZ02)

This field is specified as required in the HL7 base standard. An immunization is given at a point in time, and, in the context of immunization, the End date/time is equivalent to the Start date/time. For this reason, this document has required this field to be equal to RXA-3.

RXA-5: Administered Code (CWE_IZ01)

This field identifies the medical substance administered. If the substance administered is a vaccine, CVX and NDC codes typically are used for historical and new administrations respectively. The second set of three components may be used to represent the same vaccine using a different coding system.

RXA-6: Administered Amount (NM)

This field records the amount of vaccine administered. The units are expressed in the next field, RXA-7. When the administered amount is unknown, this field should be populated with the value "999".

RXA-9: Administration Notes (CWE_IZ01)

This field is used to indicate whether this immunization record is based on a historical record or is being reported by the administering provider.

…

Other constraint types, such as co-constraint and slicing, can be specifed if applicable. The specifier has the latitude to express the requirements in a manner best suited to their circumstance.

4.4 Data Type Profiling

Components and sub-components are constrained in the context of data type specializations; that is, a component or sub-component should not be constrained directly in their context of use. Rather, they are constrained in the context of a data type specialization, and that specialization is used in context, either at the field or complex data type level.

Data type profiling entails constraining the components of a data type. The two cases where this profiling applies are related to primitive components and complex components:

- For each primitive component in a data type, the component must be defined using the rules for Primitive Element Profiling (Section 4.5).
- For each complex component in a data type, a data type specialization may be assigned. In this nested circumstance, the data type specialization is composed of primitive elements. The data type specialization assigned would have profiled its components (subcomponents when the data type is nested) previously following the rules of primitive element profiling.

The data type specification is displayed in a table form. Table 4.9 shows a data type definition (XON) as given in the base standard. Table 4.10 shows an example of the data type specialization (XON_IZ01) of the XON data type.

Table 4.9: XON Data Type Definition in Base Standard

XON DATA TYPE								
SEQ	LEN	C.LEN	DT	OPT	TBL#	Name	Comments	SEC.REF.
1		50#	ST	O		Organization Name		2.A.76
2			CWE	O	0204	Organization Name Type Code		2.A.36
3				W		ID Number	Withdrawn as of v2.7.	
4				W		Identifier Check Digit	Withdrawn as of v2.7.	
5				W		Check Digit Scheme	Withdrawn as of v2.7.	
6			HD	O	0363	Assigning Authority		2.A.33
7	2..5		ID	O	0203	Identifier Type Code		2.A.35
8			HD	O		Assigning Facility		2.A.33
9	1..1		ID	O	0465	Name Representation Code		2.A.35
10			ST	O		Organization Identifier		2.A.76

As shown, the specifier did not include all columns as given in the base standard. If constraints are unspecified in the profile, then the requirements, or associated information of the profile from which it was derived, apply (in this case, the base standard).

Table 4.10: XON Data Type Definition Constrained in Profile

XON_IZ01 DATA TYPE				
SEQ	Name	DT	Usage	Vocab
1	Organization Name	ST	RE	
2	Organization Name Type Code	CWE	O	
3	ID Number		X	
4	Identifier Check Digit		X	
5	Check Digit Scheme		X	
6	Assigning Authority	HD_IZ01	C(R/X)	
7	Identifier Type Code	ID	C(R/X)	0203_02
8	Assigning Facility	HD	O	
9	Name Representation Code	ID	O	
10	Organization Identifier	ST	C(R/RE)	

Table 4.11 shows the list of condition predicates for the data type specialization. The implementation guide and the message profile can define multiple data type specializations for the same data type.

Table 4.11: XON Data Type Definition Constrained in Profile

Condition Predicates		
Location	**Usage**	**Predicate**
XON.6	C(R/X)	If XON.10(Organization Identifier) is valued.
XON.7	C(R/X)	If XON.10(Organization Identifier) is valued.
XON.10	C(R/RE)	If XON.1(Organization Name) is not valued.

4.5 Primitive Element Profiling

Primitive elements are those elements that are leaf nodes (i.e., they carry the data). Primitive data elements that appear at a field, component, or sub-component level can apply the following constraints:

- Usage
- Cardinality (for fields)
- Data Type Specialization (where applicable, e.g., DTM)
- Vocabulary (for coded elements)
- Length
- Conformance Length
- Content (Fixed Value, Pattern Restricted, Arbitrary Data Values)
- Conformance Statement
- Sematic Refinement

4.6 Differential Profiles

An alternative form of documenting a profile is to include only the differential between the constrained profile and a baseline (starting) profile. The baseline profile may be the HL7 v2 base standard or a message profile. For example, a local jurisdiction profile (e.g., state of Maryland) can be created from a national-level profile (US Realm profile). The profile for the state of Maryland could be expressed as a differential to the US Realm profile. The same profiling mechanisms expressed in the earlier section can be used but would only express the elements that differ from the baseline profile. For example, if only two fields were modified in a segment definition, then only those two fields (rows) would be expressed in the segment table definition. A differential profile may be represented as a profile component. The complete, i.e., the composite profile, can be created by “overlaying” the differential profile “on” the baseline profile. Having tooling that the supports expressing both the differential and composite views is optimal for end users.

4.7 Extensions

Extensions provide a mechanism for specifying a concept (e.g., a data element) that is not expressed in the base standard but is needed for a particular use case. This mechanism allows for new concepts to be specified in an "as needed" manner to support particular implementations. HL7 v2 supports extensions via the "Z" mechanism in which Z messages, Z segments, Z fields, and Z data types can be created. Creation of locally defined vocabulary is supported by the base standard natively. The use of Z elements in message profile definitions must be specified using the constraint mechanisms defined in this document.

When using extensions, the data elements should be described and documented sufficiently in a profile such that trading partners can reach a common understanding. Extensions should not be used in cases where a concept already exists in the base standard. The base standard provides a common method for defining extensions, thus ensuring consistent application of the mechanism; these rules must be adhered to in the message profile. If a given concept is found to be specified frequently as an extension in industry profiles, then such extensions should be proposed to become a formal HL7 construct in the base standard.

5 CONSTRAINTS

This section presents the constraints that can be applied during the profiling process. The types of constraints that can be applied to the various parts of a profile were given in the previous section. For each of these constraints, this section provides a description and definition, and, additionally, the compliance and compatibility rules are presented.

For each of the conformance constructs described, the standard defines a set of "allowable constraints" (i.e., the rules for profile compliance) that can be applied. Allowable constraints are a means of restricting requirements in the base standard (or another profile) to make those requirements more specific. As such, elements that are "Required" cannot be relaxed, for example, changed to "Optional".

5.1 Usage

Usage rules govern the expected behavior of the sending application and the receiving application with respect to an element. The usage indicators expand/clarify the optionality indicators defined in the HL7 base standard. Usage determines, from an implementation perspective, whether an element must be supported. Additionally, from an operational perspective, usage determines whether the element must be present, can be present, or must not be present in a message instance for the sender. For the receiver, usage influences the processing of the element.

Optionality and Usage: Optionality and Usage often refer to the same concept. Optionality is a term that has been used historically in the HL7 v2 base standard. Usage is the term that is used in the HL7 v2 conformance specification.

Table 5.1 provides an overview of the allowable usage indicators in message profiles (i.e., for constrainable and implementable profiles). While the base standard allows for all optionality indicators in some capacity, message-profile usage indicators have restricted use for a profile level.

Table 5.1: Conformance Usage Indicators and Definitions

Indicator	Name	Description
R	Required	The element is required and must be present in the message instance.
RE	Required, but may be Empty	The element is required to be supported and may be present in the message instance.
C	Undeclared Conditional	The usage of the element is conditional and is based on the outcome of a predicate that may not be defined initially. The usage indicator does not define an exact usage for the true and false outcomes and does not define an explicit predicate. Although the definition may be informative, the definition is incomplete and must be further defined. The undeclared conditional usage is the predominate form of conditional usage in the base standard. At this level, the specific context may not be known completely; therefore, precise constraints can't be specified. In constrainable message profiles, a "C" usage may be used and is interpreted as a "passthrough" from the base standard (i.e., the specifier

Indicator	Name	Description
		is indifferent). The "C" usage indicates implementation requirements are yet to be determined and, in this sense, behaves much like the "O" usage indicator, except it is known that a condition (details to be determined or removed) is associated with the element. For an implementation profile, all elements designated as undeclared conditional usage must be constrained to R, RE, C(a/b), or X.
C(a/b)	Declared Conditional	The usage of the element is conditional and is based on the outcome of a predicate. The usage indicator defines exactly the usage for the true and false outcomes and defines an explicit predicate that determines the true and false outcomes. The declared conditional defines explicit implementation requirements. The true and false outcomes must be constrained to R, RE, O, or X (therefore, nesting is not possible).
O	Optional	The usage requirements for the element have not been defined at this stage of specification. For an implementation profile, all elements designated as optional usage must be constrained to R, RE, C(a/b), or X.
X	Not-supported	The element is not supported and must not be present in the message instance.
B	Backward Compatible	The element has been designated for removal from a future version of the standard, and current use is discouraged. For an implementation profile, all elements designated as backward compatible usage must be constrained to R, RE, C(a/b), or X.

Conditional usage in the base standard often is under-specified in terms of explicitly defining a predicate and the true and false outcomes. Additionally, in many cases such a complete declaration is not possible, because the specific use requirements are unknown at the base standard level; only at the profile level can they be fully determined. Observing this reality, the conformance methodology specification henceforth recognizes this unadorned conditional usage designation as an undeclared conditional usage that can be used in constrainable message profiles. Undeclared conditional usage is distinguished from a declared conditional usage that is fully defined. The introduction of the undeclared conditional usage along with the declared conditional usage allow for multiple types of conditional usage to co-exist in message profiles without ambiguity. Undeclared conditional usage asserts no implementation requirements, but declared conditional usage does.

C(a/b) usage in the base standard: C(a/b) optionality indicator[16] replaced C optionality indicator in the base standard (Chapter 2: Optionality Section) in version 2.8 and beyond. However, the construct is rarely used as intended, and, in many cases, it is not possible for it to be used at the base standard level, because the complete set of requirements are unknown. In nearly all cases, the unadorned C usage indicator is used for specifying conditional elements and not C(a/b). This specification considers elements specified as conditional in the base standard *in essence* as undeclared conditional elements and may be constrained as an undeclared conditional

[16] The base standard refers to the "appearance" indicator as optionality, the conformance section of the standard refers to the "appearance" indicator as usage.

(C) or a declared conditional C(a/b) in message profiles.

5.1.1 USAGE REQUIREMENTS FOR SENDING APPLICATIONS

Table 5.2 shows the usage rules requirements for a sending application. These requirements are expressed from the perspective of implementation and operational requirements. Usage implementation requirements indicate whether the application must support the element. Usage operational requirements indicate whether the application must provide a value for an element in a message instance. Whether or not a value is provided is sometimes dependent on conditions and data availability. Table 5.2 indicates what must be supported by the implementation and what must be provided in the message instance, not how it is accomplished.

Table 5.2: Usage Requirements Sending Applications

Indicator	Description	Implementation Requirement	Operational Requirement
R	Required	The application must support an element with an “R” usage designation.	The application must value an element with an “R” usage designation.
RE	Required, but may be empty	The application must support an element with an “RE” usage designation.	The application must value an element with an “RE” designation if data is known for that element.
C	Undeclared Conditional	There are no implementation requirements. The “C” usage designation is a placeholder indicating that the usage for this element has not yet been specified.	Not Applicable.
C(*a*/*b*)	Declared Conditional	The application must support the implementation requirements as indicated by the true (“a”) outcome and by the false (“b”) outcome usage indicators in the declared conditional definition.	The operational usage designation for the element is determined based on the outcome of an associated predicate at runtime. If the predicate associated with the element is true, follow the usage rule requirements for “***a***”, which must be one of “R”, “RE”, “O”, or X”: If the predicate associated with the element is false, follow the usage rule requirements for “***b***”, which must be one of “R”, “RE”, “O”, or X”.
X	Not supported	There are no implementation requirements.	The application must not value an element with an “X” usage designation.
O	Optional	There are no implementation requirements. The “O” usage designation is a placeholder indicating that the usage for this element has not yet been specified.	Not Applicable.

Indicator	Description	Implementation Requirement	Operational Requirement
B	Backwards Compatible	There are no implementation requirements. The "B" usage indicates that the element is retained for backwards compatibility of the element. Another usage indicator may be assigned in a derived profile.	Not Applicable.

The use of the RE usage code is qualified with the "if data is known" clause. The sender must interpret the clause as "the capability must always be supported, and data must always be sent if known". To clarify, the sender does not determine whether the data should be sent; to be conformant to the rule, the data must be sent. There is a misconception where the RE usage is interpreted as "the capability must always be supported, and data may or may not be sent even when known based on a condition external to the profile specification". If there are valid external conditions, then the profile does not describe the use case accurately, and the profile needs to be modified accordingly, or possibly another profile needs to be created. This is not to say that the sender doesn't control what data they send in production systems, but it is an indication that the sender is not conformant to the profile to which they are claiming conformance. The consequence of non-conformity created by not sending known data for an RE element is out of scope for this specification. See Section 1.5.1 for insight into how installed systems can handle non-conformities.

5.1.2 USAGE REQUIREMENTS FOR RECEIVING APPLICATIONS

Table 5.3 shows the usage rules requirements for a receiving application. These requirements are expressed from the perspective of implementation and operational requirements. Usage implementation requirements indicate whether the application must support the element. Usage operational requirements indicate whether the application must process the value for the element in a message instance. The concept of "must process in a meaningful way" is intentionally vague at the interface level specification. Exactly what this means can be further defined in the profile definition in the context of a use case or in a functional requirements specification. The phrase "process in a meaningful way" can mean many things, as illustrated in the simple examples given below:

- Commit the data element to a data base
- Use the data element for some task (e.g., patient matching) and then discard the data
- Display the data element in the context of a report
- Evaluate the data element and, based on that evaluation, perform a certain task
- Compare the data to existing data base content and, if different, decide which data to keep based on source reliability or other factors

Table 5.3 indicates what must be supported by the implementation and what must be processed by the implementation, but not how the processing is to be accomplished.

Table 5.3: Usage Rules Requirements for a Receiving Application

Indicator	Description	Implementation Requirement	Operational Requirement
R	Required	The application must support an element with an "R" usage designation.	The receiving application must process in a meaningful way the information conveyed by an element with an "R" usage designation. A receiving application must raise an exception due to the absence of a required element. A receiving application must not raise an exception due to the presence of a required element[17].
RE	Required, but may be empty	The application must support an element with an "RE" usage designation.	The receiving application must process in a meaningful way the information conveyed by an element with an "RE" usage designation. The receiving application must process the message if the element is omitted (that is, an exception must not be raised because the element is missing). A receiving application must not raise an exception due to the presence of a required element[18].
C	Undeclared Conditional	There are no implementation requirements. The "C" usage designation is a placeholder indicating that the usage for this element has not yet been specified.	Not Applicable.
C(*a*/*b*)	Declared Conditional	The application must support the implementation requirements as indicated by the true ("a") outcome and by the false ("b") outcome usage indicators in the declared conditional definition.	The operational usage designation for the element is determined based on the outcome of an associated predicate at runtime. If the predicate associated with the element is true, follow the usage rule requirements for "***a***", which must be one of "R", "RE", "O", or X": If the predicate associated with the element is false, follow the usage rule requirements for "***b***", which must be one of "R", "RE", "O", or X".

[17] A receiving application may, however, raise an exception due to invalid content of an element that is required based on semantic or business requirements.

[18] A receiving application may, however, raise an exception due to invalid content of an element that is required based on semantic or business requirements.

Indicator	Description	Implementation Requirement	Operational Requirement
X	Not supported	There are no implementation requirements.	None if the element is not present. If the element is present, the receiving application may process the message but must ignore the element content and may raise an exception. The receiving application must not process the information conveyed in an element with an "X" usage designation.
O	Optional	There are no implementation requirements. The "O" usage designation is a placeholder indicating that the usage for this element has not yet been specified.	Not Applicable. The receiver in an implementation profile or implementation makes a choice of an allowable usage indicator and operates based on that usage.
B	Backwards Compatible	There are no implementation requirements. The "B" usage indicates that the element is retained for backwards compatibility of the element. Another usage indicator may be assigned in a derived profile.	Not Applicable.

5.1.3 CONDITIONAL USAGE

Conditional usage has several facets that necessitate further explanation, including formal definitions of undeclared and declared conditional usage, factoring conditional usage to a non-conditional usage, and predicate definition.

5.1.3.1 UNDECLARED CONDITIONAL USAGE

Undeclared conditional usage is a designation that the element is conditional on another element or elements, but the specific requirement can't be determined at the current level of use case knowledge. Additionally, the conditional usage may be out of scope for the use case, but the specifier chooses to be silent on the element requirements; that is, the specifier neither fully defines the conditional usage nor eliminates its potential use in a derived profile. The undeclared conditional usage can be thought of as a "passthrough" in a message profile; and it is analogous to an optional element in the base standard, remaining as an optional element in a constrainable profile.

Although an undeclared conditional usage isn't fully or formally defined, that fact does not imply that any requirements specified are not obligatory in a derived profile if "activated". A requirement defined in the undeclared conditional usage must be preserved in any subsequent definition if the underlying condition hasn't changed.

5.1.3.2 DECLARED CONDITIONAL USAGE

The Declared Conditional usage is a fully-specified declaration of the true and false outcome usages based on an explicitly defined predicate. The formal definition is:

C(*a*/*b*) with an associated predicate where "***a***" and "***b***" in the expression are placeholders for usage codes representing the true ("***a***") predicate outcome and the false ("***b***") predicate outcome of the condition. The condition is expressed by a conditional predicate associated with the element. "***a***" and "***b***" must be one of "R", "RE", "O", and/or "X". As such, the conditional usage construct cannot be nested. The values of "a" and "b" cannot be the same; logically, if this is the case, the usage indicator resolves to a single non-conditional usage.

The example C(R/RE) is interpreted as follows: if the predicate associated with the element is true, then the usage for the element is R-Required; if the condition predicate associated with the element is false, then the usage for the element is RE-Required but may be empty.

5.1.3.3 FACTORING CONDITIONAL USAGE TO NON-CONDITIONAL USAGE

Depending on the profiling of an element with a conditional usage, the conditional designation may be replaced with a non-conditional designation. If a profile constrains an element that is the object of the predicate to a usage that renders the predicate inconsequential, then the conditionality can be removed.

For example, a specification defines the usage of an element (E1) as "C", and the condition predicate is dependent on the presence or non-presence of another element (E2). The conditional usage is defined as C(R/X). The profile may constrain the element (E2) that the condition is dependent on to X (not-supported)[19]; in such a circumstance the condition for the element (E1) would always evaluate to false, and, therefore, the condition for the element (E1) resolves to C(not possible/X). The only possible outcome is usage "X" for the element (E1) that was previously defined with a conditional usage. The predicate and conditional usage become inconsequential and thus can be removed and a non-conditional usage can be specified (X in this example). This provides the rationale by which a conditional usage in both forms can be profiled to a single usage indicator (R, RE, or X).

The conditional usage that resolves to a single usage indicator is limited to R, RE, or X. Usage of O is not possible. If the usage was originally C, then keeping the usage indicator as C is appropriate if the element is still in consideration for an implementation requirement. See column DTF1 in Table 5.4 for more details. Using C instead of O is best practice. In this example, C is behaving like an O, but the informational aspect of the element originally dependent on another element is maintained.

C and CE Conformance Usage: Conditional (C) and Conditional, but maybe Empty (CE) usage indicators were introduced in the conformance specification in version 2.5 and deprecated in version 2.7.1. In version 2.7.1, C(a/b) was introduced, which subsumed C and CE. C equates to C(R/X) and CE equates to C(RE/X). Note, that the definition of "C" optionality in the base standard does not match the definition of "C" usage in various versions of conformance section/chapter.

[19] Note: if Element E2 is valued, it is an error and not a candidate to affect the predicate outcome.

Table 5.4 describes various use cases for profiling conditional usage. The examples employ an excerpt of the CWE data type (as of version 2.7). CWE.1 (Code) is the code identifier, CWE.2 (Text) is the text that describes the code concept, CWE.3 (Code System) is the name of the code system of which the code (CWE.1) is a member, and CWE.14 (Code System OID) is the OID of the code system of which the code (CWE.1) is a member. The BASE column indicates the specification as given in the base standard, in which the Code and Text are optional and the Code System and Code System OID are conditional. The conditions and relationships for the CWE definition as defined in the base standard are that if a code is provided, then either the code system or the code system OID must also be provided.

Table 5.4: Examples of Conditional Usage

Element	Name	Base	DTF1	DTF2	DTF3	DTF4	DTF4 Predicate
CWE.1	Code	**O**	**R**	**X**	**R**	**RE**	
CWE.2	Text	**O**	**RE**	**R**	**RE**	**C(R/RE)**	If CWE.1 is not valued.
CWE.3	Code System	**C**	**R**	**X**	**X**	**C(R/X)**	If CWE.1 is valued.
CWE.14	Code System OID	**C**	**C**	**X**	**R**	**C(O/X)**	If CWE.1 is valued.

Four profiling examples of the CWE data types that represent a data type flavor (DTF1, DTF2, DTF3, DTF4) are explained below, with references to information in Table 5.4:

DTF1: The specifier is requiring a code and the associated code system. Additionally, the specifier is making no statement on the requirement for CWE.14. This requirement designation is left to be specified in a derived profile. The usage of “C” is used instead of “O” because of the dependency of the element. Since the original condition has been satisfied by requiring CWE.3, CWE.14 can be profiled to one of R, RE, or X in a derived profile. There is no need to specify an explicit predicate since the condition is an undeclared conditional usage.

DTF2: The specifier is indicating that a code is not to be supported, therefore, CWE.3 and CWE.14 must also not be supported. The text component in this case is required. Since the original predicate will always resolve to false, the specification of a predicate is not necessary, and a non-conditional usage indicator can be specified (“X” in this example).

DTF3: The specifier is indicating that a code and the associated code system OID are required. The code system name is not to be supported. Since the original condition is satisfied by requiring CWE.14, the specifier has the option to constrain CWE.3 to R, RE, X, or leave it to be determined in a derived profile (usage = C). In this case the specifier decided not to support (X).

DTF4: The specifier is constraining the data type flavor such that a code must be supported, and, if available, it must be valued (CWE.1). If the code is not available, then text describing the concept must be valued (CWE.2). If a code is provided, then the code system name (CWE.3) must be provided as well. The specifier is leaving it up to the specifier of a derived profile as to whether the code system OID (CWE.14) must be

supported. Since a declared conditional usage is specified, an explicit condition predicate is given (Column DTF4 Predicate).

5.1.3.4 PREDICATE DEFINITION

If the usage code of an element is a declared conditional (i.e., C(a/b), then an explicit conditionality predicate must be associated with this element. The result of the conditional computation determines the usage code as indicated by the true and false usage outcomes. The predicate must be testable and based on other values within the message. In the message profile specification, the condition predicate must be indicated as a formal declaration that associates the element (i.e., element location) to the computable condition predicate. Additionally, the condition predicate must be defined based on the context in which it applies:

Data Types: the condition predicate must be defined in the data type context

Fields: the condition predicate should be defined in the same segment context

Segments/Groups: the condition predicate must be defined in the message context

See Section 5.1.3.3 (Table 5.4) for examples of the context of a data type. The conforming sending and receiving applications must both evaluate the predicate. The condition predicate is only specified for the declared conditional usage in message profiles.

Conditional Usage on the Event Type (MSH-9.2): Although not explicitly prohibited, the use of the event type as the basis of the conditional usage is not recommended. Specification of individual profiles is the preferred approach.

The predicate may be expressed in a computable language (recommended) or in plain text. This specification does not prescribe a specific method for expressing a predicate. However, a pseudo language (See Appendix A) has been developed to concisely and consistently express conditional usage predicates. The language is specifically designed for HL7 v2 in terms of relatability and ease of use. When possible, use of this language is recommended.

5.1.4 USAGE COMPLIANCE

Usage compliance indicates the allowable paths by which element optionality is removed. Optionality lessens as the profiling proceeds from the base standard to an implementation profile. Usage requirements can only be strengthened, they must not be relaxed. For example, an optional element can be made required in a derived profile, however, a required element can not be made optional in a derived profile.

Table 5.5 presents the allowable transitions for each Usage code at each possible profile-level transition. The “Usage Code” column indicates the usage code for the “starting” profile level. The columns to the right of the Usage Code column indicate the possible usage code for the “ending” profile level. The “starting” profile level is indicated by the top line in the column heading, and the “ending” profile level is indicated by the bottom line in the column heading. For example, in the first row of the Usage Code column, the usage code is R, which is the usage of the element as defined in the Base standard. This usage code can only transition to R in a

constrainable profile, as indicated by the Usage Code = R and Base-to-Constrainable coordinates.

Table 5.5: Usage Compliance

Usage Code	Base-to-Constrainable	Constrainable-to-Constrainable	Base-to-Implementation	Constrainable-to-Implementation	Implementation-to-Implementation
R	R	R	R	R	R
RE	R, RE	R, RE	R, RE	R, RE	R, RE
O	R, RE, C(a,b), O, X	R, RE, C(a,b), O, X	R, RE, C(a,b), X	R, RE, C(a,b), X	N/A
C	R, RE, C, C(a,b), X	R, RE, C, C(a,b), X	R, RE, C(a,b), X	R, RE, C(a,b), X	N/A
C(a/b)	R, RE, C(a,b), C(a'/b'), X	R, RE, C(a,b), C(a'/b'), X	R, RE, C(a,b), C(a'/b'), X	R, RE, C(a,b), C(a'/b'), X	R, C(a,b), C(a'/b')
X	X	X	X	X	X
B	R, RE, C(a,b), O, X, B	R, RE, C(a,b), O, X	R, RE, C(a,b), X	R, RE, C(a,b), X	N/A
W	X	N/A	X	N/A	N/A

In an implementable profile, ultimately only two possibilities are allowed: either a specific element is supported ("R or RE") or it is not ("X"). For a conditional usage, the true and false outcomes must also be defined only as R, RE, or X.

An element with a usage of withdrawn is not to be used. The usage code “W” can only be profiled to “X”, and use of “X” is required in the profiles. The usage code “B” in a constrainable profile can be profiled to another usage indicator; however, this approach is not recommended (unless it is being profiled to "X"), since the intent of the authors of the standard is that this element not be used in the future.

5.1.4.1 CONDITIONAL USAGE COMPLIANCE

Table 5.6 indicates the circumstances in which an undeclared conditional usage remains an undeclared conditional usage and the implications of that situation; and the circumstances in which an undeclared conditional is constrained to a declared conditional usage and the implications of that transition. Note, the conditional usage in the base standard is not referred to as an undeclared conditional usage.

Table 5.6: Use of Undeclared Conditional Usage

Indicator	Transition	Resulting Indicator	Comments
C	Base-to-Constrainable Constrainable-to-Constrainable	**C**	The message profile does not further specify the element definition beyond what is defined in the base standard. Since the message profile does not declare an explicit definition of the conditional usage and condition predicate, there are no implementation requirements. The specifier does not make an explicit declaration on the inclusion of the element. Therefore, the “C” usage is treated the same as the “O” usage indicator in

Indicator	Transition	Resulting Indicator	Comments
			terms of implementation requirements (i.e., the implementation requirements are determined in a derived profile).
C	Base-to-Constrainable Constrainable-to-Constrainable Base-to-Implementable Constrainable-to-Implementable	**C(a/b) with predicate**	The message profile explicitly defines the true (a) and false (b) usage outcomes for the conditional usage based on the predicate statement. A predicate statement is required. This declaration defines explicit requirements for implementation.

Elements with a conditional usage indicator require a separate examination, because a specialization allows for different combinations depending on the characteristics of the constraint.

Compliance assessment for elements with conditional usage (i.e., C(a/b)) is dependent on the respective true and false usage code specification. For example, if conditional usage for an element is specified as C(RE/O), the true usage code "RE" can be profiled to "RE" or "R" in a derived profile. The false usage code of "O" can be profiled to "R", "RE", "O"[20], or "X". The conditional usage codes may collapse to a single non-conditional code if the usage codes are profiled to the same code. For example, if the "RE" and "O" usage are both profiled to "R", then the usage code can be specified simply as "R" and not C(R/R). Table 5.7 summarizes the possible constraints applicable to conditional usage.

Table 5.7: Summary of Compliance Rules for Constraining Conditional Usage

Base Profile	Derived Profile	Comment
C(a/b)	C(a/b)	The derivation remains unchanged.
C(a/b)	C(a' / b')	*a'* is a valid specialization (constraint) of *a*, and *b'* is a valid specialization of *b*; *a* and *b* can be constrained individually, the condition remains unchanged.
	a	If *b'* is a valid specialization of *b*, and this is equal to *a*. For example, the usage is C(R/O), and because of the specific use case being profiled the specifier wants to further constrain the false outcome to R. Therefore, the conditional is C(R/R), which resolves to R.
	b	If *a'* is a valid specialization of *a*, and this is equal to *b*. For example, the usage is C(O/X), and because of the specific use case being profiled the specifier wants to constrain the true outcome to never allow that element. Therefore, the conditional is C(X/X), which resolves to X.

[20] In a constrainable profile.

Base Profile	Derived Profile	Comment
C(a/b)	a	If the condition is always met in a specific use case.
C(a/b)	b	If the condition is never met in a specific use case.

The condition in the derived profile shall not modify the condition in the profile from which it was derived. The exception is a removal of the condition in a derived profile if it can be evaluated consistently to either true or false and then be replaced by the appropriate usage code.

A reasonable question is: should the condition predicate itself be allowed to change?

The authors are unaware of any standard that provides guidance on this question. An example situation could be changing the condition from "if the patient is male" to "if the patient is male and older than 18 years". Any change to a condition changes the result set as well. In the modified version of the condition in this example, some patients would be excluded because they are too young. In principle, such a change causes the application to evaluate the data in a different way, but it does not change the related usage of this element and, therefore, does not change the handling of this element. It is unclear whether this kind of change should be an allowable "constraint". When changing a condition, careful consideration should be given to the potential impact on implementations.

5.1.5 USAGE COMPATIBILITY

Table 5.8 provides a technical compatibility analysis of an element's usage in a sender profile to an element's usage in a receiver profile. For a pair of profiles to be compatible, all element pairs in the profiles must adhere to the profile compatibility rules. For example, if the sender profile specifies an element as required and the receiver profile also specifies the corresponding element as required, then the profiles are compatible for that element. If, however, the sender profile specifies an element as not-supported and the receiver profile specifies the corresponding element as required, then the profiles are not compatible for that element, since the receiver is expecting data that the sender will never provide.

Table 5.8 addresses implementable profiles where each element must be profiled; that is, no elements can be optional. Table 5.9 addresses additional optionality choices available for constrainable profiles. Tables 5.8 and 5.9 do not take conditional usage into account; an analogous analysis can be performed for each true and false outcome of the conditional usage.

Optional elements apply only to constrainable profiles. Often specifiers develop constrainable-level profiles for national specifications. Their goal is to specify elements that are needed to meet their use case requirements. Beyond that, they allow trading partners to negotiate among themselves regarding local customization of the remaining un-profiled (or optional) elements.

Assessing technical compatibility among applications requires a comparison of the capabilities of the sending side to the capabilities of the receiving side through the means of profiles. For example, the assignment of "R" usage for an element on the sending side expresses the fact (or the intent) that this element is always valued in every message instance that is sent. "RE" usage

expresses the intent that data will be present in messages if the data are entered into the system or are made available in some other way. In other words, these usage requirements identify what a receiver can expect in messages being sent to them. Statements for a receiver are obligatory expressions of their requirements. Therefore, a required element ("R" usage) indicates that the receiver must get this information in order to be able to process the message or a specific part of the message. A pairing of X → R is deemed not compatible. However, the pairing "R → X" is deemed compatible since technical compatibility is assessed on receiver side requirements and in this case the receiver does not need the data to fulfill its use case. However, if the sender had an expectation that the element is processed in some manner, then a resolution would be decided in the functional compatibility negotiations.

Table 5.8: Sender/Receiver Pair Profile Compatibility Rules

Sender	Receiver	Compatible	Comment
R	R	Yes	Sender and Receiver have the same expectations.
R	RE	Yes	Receiver supports this element but is not always expecting it.
R	X	Yes	Receiver doesn't support this element.
RE	R	No	Receiver is not guaranteed to get required data.
RE	RE	Yes	Sender and Receiver have the same expectations.
RE	X	Yes	Receiver doesn't support this element.
X	R	No	Receiver will not get required data.
X	RE[21]	No	Receiver will not get the data it needs for certain use cases. Note: a data value must be needed in at least one instance; otherwise, the element should not be profiled to RE. On the other hand, RE is the only construct that expresses the capability of the receiving system to handle the data.
		Yes	The element is not necessary for operation.
X	X	Yes	Sender and Receiver have the same expectations.

The analysis of optional elements for profile compatibility provides guidance for pairing potential implementable profiles derived from constrainable profiles. A definitive assessment of profile compatibility can't be made until implementation profiles are developed; however, the guidance provided here will aid in the specification of constrainable profiles. As is to be expected, profile compatibility of constrainable profiles is directly linked to the requirements of the compatibility rules of implementable profiles.

[21] In this combination, compatibility depends on the use case. If the data are important in order to perform the use case, then this combination is not compatible. But if RE only declares the capability of the system, then this pair is compatible.

Table 5.9: Compatibility Analysis for Optional Elements

Sender	Receiver	Compatible	Comment
R	O	Yes	The receiver does not need data for this element for its current use case. Derived compatible receiver profile settings include R, RE, or X.
RE	O	Only RE, X	The receiver does not need data for this element for its current use case. Derived compatible receiver profile settings include RE or X.
X	O	Only X	The receiver does not need data for this element for its current use case. A derived compatible receiver profile setting can only be achieved if the usage for the element is constrained to X.
O	R	Only R	The receiver needs data for this element for its use case. The only derived compatible sender profile setting is R.
O	RE	Only R, RE	The receiver needs data for this element for its use case. The derived compatible sender profile setting are R and RE.
O	X	Yes	The receiver does not need data for this element for its use case. A derived compatible sender profile setting are R, RE, and X.
O	O	Possible	Compatibility can be achieved by following the rules for Implementation profiles as given in Table 5.8.

5.1.6 USAGE AND CONFORMANCE

The base standard allows broad flexibility for the message structures that HL7-conformant applications must be able to receive without failing; but, while the base standard allows messages to be missing data elements or to contain extra data elements, no inference should be made from these declarations that such messages are conformant in the context of a message profile. In fact, the usage codes specified in a message profile impose strict conformance requirements on the behavior of the application. For example, the presence of unexpected content for an unspecified element in the message profile is a conformance violation. A case in point would be where data are present for a fourth component of a data type, but the message profile only defines three components for the data type.

5.2 Cardinality

A data element often must occur more than once in the instance of a message. Cardinality is the conformance construct that is used to indicate this requirement. Cardinality controls the number of occurrences of an element an implementation must support and the number of instances of an element that can appear in a message. Some elements shall always be present (e.g., cardinality

[1..1], [1..n]). Others shall never be present (i.e., cardinality [0..0]). Still other elements may or may not appear in the message instance, with zero or more occurrences (e.g., cardinality [0..n]). In certain circumstances, the maximum number of occurrences may have no specified/practical limit. In this case, it is identified with "*" (e.g., [1..*]).

Cardinality identifies both the minimum and maximum number of occurrences that a message element must have in a conformant message and is expressed as an interval of a minimum-maximum pair of non-negative integers. A conformant message must always contain at least the minimum number of occurrences and shall not contain more than the maximum number of occurrences. As an example, a cardinality of "[0..5]" would indicate that the element need not occur in the instance at all or it may occur up to five times; an implementation must support up to five occurrences of an element. If the minimum number of occurrences is 0, the element may be omitted from a message.

Table 5.10: Cardinality

Cardinality	Interpretation	Valid Usage
[0..0]	Element is never present.	X
[0..1]	Element may be omitted, and it can have at most one occurrence.	RE, C(a/b), O
[1..1]	Element must have exactly one occurrence.	R
[0..n]	Element may be omitted or may have up to ***n*** occurrences.	RE, C(a/b), O
[1..n]	Element must appear at least once and may have up to ***n*** occurrences.	R
[0..*]	Element may be omitted or may have an unlimited number of occurrences.	RE, C(a/b), O
[1..*]	Element must appear at least once and may have an unlimited number of occurrences.	R
[m..n]	Element must have at least ***m*** occurrences and may have at most ***n*** occurrences. When the usage indicator is RE, the element may be omitted ("zero occurrences"). ***m*** must be greater than 1 and ***n*** must be greater than or equal to ***m***; the case where ***m*** equals 1 is addressed separately.	R, RE
[m..*]	Element must have at least "m" occurrences and may have an unlimited number of occurrences. When the usage indicator is RE, the element may be omitted ("zero occurrences"). ***m*** must be greater than 1; the case where ***m*** equals 1 is addressed separately.	R, RE

Cardinality boundaries apply both to primitive data of a simple data type and a collection of data contained in a complex data type. An explicit cardinality range is required for segment groups, segments, and field elements. Component and sub-component elements do not explicitly include a cardinality range, but a cardinality range is implicitly associated with each component and sub-component element. The associated cardinality depends on the element's *usage* code. For

components and sub-components with a usage code of *R*, the associated cardinality range is [1..1]; for all elements with a usage code of *RE* or *O*, the associated cardinality is [0..1]; for all elements with a usage code of C(a/b), the associated cardinality is determined by the resultant usage based on the evaluation of the condition predicate; and for all elements with an *X* usage code, the associated cardinality is [0..0].

5.2.1 RELATIONSHIP BETWEEN USAGE AND CARDINALITY

A relationship exists between the usage and cardinality constructs. The Cardinality column in Table 5.11 shows the valid cardinality values and the Usage column indicates the usage codes that can be used with the corresponding cardinality range. The following information summarizes the constraints on the allowed combinations:

- If the usage of an element is Required (R), the minimum cardinality for the element shall be equal to or greater than 1.
- If the usage of an element is not Required (R) (i.e., any code other than 'R'), the minimum cardinality shall be 0 except in the following condition:

  If the profile author wishes to express a circumstance where an element will not always be present, but when it is present it must have a minimum number of occurrences greater than one, they may document this rule by specifying the RE usage code with the minimum cardinality representing the minimum number of occurrences when the element is present. This expression would be in the form [m..n], indicating that permitted occurrences are either zero or the range of m through n.

Table 5.11: Example Cardinality-Usage Combinations

Cardinality	Usage	Interpretation
[1..1]	R	There will always be exactly 1 occurrence.
[1..5]	R	There will be 1 to 5 occurrences inclusive.
[0..1]	RE	The element must be supported, but may not always be present.
[0..5]	C(R/X)	If the condition predicate is true, there will be 1 to 5 occurrences inclusive. If the condition predicate is false, there will be 0 occurrences.
[3..5]	RE	If any values for the element are valued, there must be at least 3 and no more than 5 occurrences. However, the element may be absent (0 occurrences).
[3..5]	R	There will be 3 to 5 occurrences inclusive.

5.2.2 REPETITION, OCCURRENCE, AND CARDINALITY

Historically, the base standard has used the term “repetition” to indicate the upper limit and not the actual number of times an element may repeat (in contrast to its well-established dictionary

definition). The terms “maximum occurrences” or “maximum cardinality” are the preferred terms when describing the number of times an element may appear in a message. It is important to note that if the upper boundary for repetition is *n* it means n occurrences and not *n* repetitions (i.e., *n+1* occurrences). This document uses the terms occurrences and cardinality.

5.2.3 CARDINALITY COMPLIANCE

Table 5.12 lists the rules for constraining cardinality. The left-most column indicates the cardinality for an element as defined in the “Parent Profile” (e.g., the base standard). The combination of the information in the “Derived Profile” column (always m..n) and the “Valid Compliance Rule” column indicates possible modifications of the cardinality constraint. The associated “Example(s)” column provides valid instances. Likewise, the information in the “Derived Profile” column (always m..n) and the “Invalid Compliance Rule” column indicate possible modifications (invalid in this case) of the cardinality constraint. The associated “Example(s)” column provides non-valid instances.

For instance, a cardinality defined in the parent profile as [0..0] and then *constrained*[22] to [0..0] (m=0 and n= 0) is a valid constraint (row 1 – valid column); however, if the cardinality is *constrained* to [1..4], it is invalid (row 1 – invalid column). In Table 5.12 it is assumed that “*m*” is always less than or equal to “*n*”. Generally speaking, the cardinality range must be constrained by increasing the lower boundary and decreasing the upper boundary. The minimum cardinality has to be less than or equal to the maximum cardinality. Additionally, for some of the examples listed in Table 5.12 a specific value for a variable is used to facilitate the explanation.

Table 5.12: Compliance Assessment for Constraining Cardinality

Parent Profile	Derived Profile	Valid Compliance		Invalid Compliance	
		Rule[23]	Example(s)	Rule	Example(s)
[0..0]	[m..n]	m=0 and n=0	[0..0]	m≠0 or n≠0	[0..1], [1..4]
[0..1]	[m..n]	m≤1 and n≤1	[0..0], [1..1]	m>1 or n>1	[0..3], [1..2]
[0..x]	[m..n]	n≤x	x=3, [0..0], [0..3]	m>x or n>x	x=3, [4..6], [0..4]
[0..*]	[m..n]	m≤n	[0..200], [2..40]	m>n	[1..0], [5..1]
[1..1]	[m..n]	m=1 and n=1	[1..1]	m≠1 or n≠1	[0..1], [1..2]
[1..x]	[m..n]	m≥1 and n≤x	x=3, [1..3], [2..2]	m<1 or n>x	x=3, [0..3], [1..5]
[1..*]	[m..n]	m≥1 and n≥1	[1..1], [2..200]	m<1 or n<1	[0..0], [0..200]
[x..x]	[m..n]	m=x and n=x	x=3, [3..3]	m≠x or n≠x	x=3, [0..3], [3..4]
[x..y]	[m..n]	m≥x and n≤y	x=3, y=5, [4..5]	m<x or y<n	x=3, y=5, [3..6]
[x..*]	[m..n]	m≥x and n≥x	x=3, [3..3], [4..5]	m<x or n<x	x=3, [2..2], [2..6]

[22] Or not modified in this case.

[23] In addition, “*m*” has to be less than or equal “*n*”. As mentioned, this assumption applies to all cases in this analysis.

It is important to note that this analysis does not account for the dependencies that an associated usage constraint places on cardinality constraints. In Table 5.12, the usage reference point is O-optional. For a cardinality of [0..1], a valid constraint is [0..0] as indicated in row 2 – Valid Compliance column; additionally, [1..1] is also a valid constraint. In the context of a related usage, such constraints may not be valid. For example, for a cardinality of [0..1] and an associated usage of "RE-required, but may be empty", the maximum cardinality must be 1 (allowing for appearance of the element). Likewise, if the usage is "X-not-supported", then a cardinality setting of [0..0] is the only valid constraint of [0..1]. See Section 5.2.1 for additional information on the dependencies between usage and cardinality.

Note that the set of constraints given in Table 5.12 can be applied to any profile level in the hierarchy (i.e., the constraint table can be applied *recursively*). Additionally, once a constraint has been applied, only further (or the same) constraints can be specified in the (another) derived profile. For example, if the base profile "A" has a cardinality of [0..1] that is constrained to [1..1] in a derived profile "B", then in derived profile "C" the cardinality constraint must remain as [1..1] (as indicated in row 5 of the Table). On the other hand, if the base profile "A" cardinality of [0..*] is constrained to [1..5] in a derived profile "B", then in derived profile "C" the cardinality can be further constrained to [2..4]. In this example, the constraints transitioned from [0..*] (see row 4 of the Table) to [1..x] (see row 6 of the Table), and finally to [x..y] (see row 9 of the Table).

5.2.4 CARDINALITY COMPATIBILITY

Compatibility is a measure that indicates whether two specifications (or systems that implement the same specification) have harmonized requirements. Compatibility is determined from the perspective of a receiver. Table 5.13 presents an analysis of compatibility for a set of sender/receiver cardinality pairs. The analysis is a direct assessment of the sender/receiver pairs indicated and does not consider what pairs might actually be enacted for an element in a particular use case. For example, in the case of a sender cardinality of [1..1] and a receiver cardinality of [0..0], the receiver doesn't need the data element to be valued in order to operate. If the sender has an expectation that the receiver must process the data element, however, then this requirement is indicated in the use case, and, as such, the receiver must set the cardinality for the element to [1..1], which would be part of the profile/interface negotiations.

Table 5.13: Compatibility Analysis for Cardinality

Sender	Receiver	Compatible	Comment
[0..0]	[0..0]	Yes	Sender and Receiver have the same expectations.
[0..0]	[0..m]	Yes	Receiver can process data but can also handle absence of data.
[0..0]	[n..m]	No	Receiver will not get required data if n>0.
[0..1]	[0..0]	Yes	Receiver has no expectations[24].
[0..1]	[0..1]	Yes	Sender and Receiver have the same expectations.

[24] This case and similar cases are analogous to the sender usage of "R" and the receiver usage of "X". If the use case dictates that the sender expects the element to be handled, then the element must be profiled as "R" for both the sender and the receiver.

Sender	Receiver	Compatible	Comment
[0..1]	[0..m]	Yes	Receiver supports more than the sender.
[0..1]	[n..m]	No	Receiver will not get required data if n>0.
[1..1]	[0..0]	Yes	Receiver has no expectations[25].
[1..1]	[0..1]	Yes	Receiver processes the data.
[1..1]	[1..1]	Yes	Sender and Receiver have the same expectations.
[1..1]	[1..m]	Yes	Receiver supports more than the sender.
[1..1]	[n..m]	No	Receiver will not get required data if n>1.
[x..y]	[n..m]	Yes	If m<x.
[x..y]	[n..m]	No	If n>y.

5.3 Data Type Specialization (Flavor)

Data types and specializations of data types are core aspects of the HL7 v2.x standard and implementation guides. The HL7 v2.x standard provides a set of data types (referred to as the "base" data types). Typically, the base data types are not used "as-is" in implementation guides; instead they are constrained for a particular need in the profiling process. The specialization of the data type is referred to as a data type "flavor". Each element in the data type can be constrained following the rules defined by the conformance constructs. The data type flavor is assigned a new identifier that can be referenced by other elements as part of the profiling process.

An example of data type specialization was given in Section 4.4. Additionally, a detailed discussion on data type specializations and on the HL7 v2 standardized data type library can be found in the HL7 Version 2 Specification: Data Type Specializations, Release 1 (currently in ballot resolution) and on the web site https:/v2.hl7.org/datatypeflavors (forthcoming). The goal of standardized data type flavors is to promote consistency and reuse of commonly used data type specializations.

5.3.1 ROOT DATA TYPE SUBSTITUTION

Under certain circumstances, a use case may require a data type substitution at the root level. One example would be when an IS data type is replaced with a CWE data type. Root data type substitutions are promotions, that is, more functionality is specified. The following compliance and compatibility rules apply:

- A primitive data type can replace a primitive data type at any level.
- A primitive data type must not replace a complex data type.
- A complex data type can replace a primitive data type but only at the field level.
- A complex data type can replace a complex data type at any level.

[25] see previous footnote. For this case, cardinality of the receiver must be [1..1] based on use case requirements of the sender.

- A data type substitution must follow the conformance constructs compliance rules for all the constitute parts of the data type.
- The data type substitution must not reduce capabilities of the data type it is replacing
- The data type substitution must be backwards compatible with the data type it is replacing

When a substitution is made, a data type in which all constituent parts and attributes are compatible must be specified. Many substitutions are possible, and specifiers should exercise caution when changing the root data type, as detecting non-compliance can be challenging. Table 5.14 provides a few examples of root data type substitutions and an analysis of the substitution.

Table 5.14: Root Data Type Substitution

Original	Replaced By	Analysis	Comments
IS	**CWE**	Valid	1st component of CWE is compatible with "IS" data.
CWE	**IS**	Invalid	Data type demotion; fewer capabilities.
TX	**FT**	Valid	Primitive data types; expands capabilities of text.
FT	**TX**	Invalid	Reduces the capabilities of the element.
CWE	**CNE**	Valid	Same structure, added usage constraint; in essence CNE is a data type flavor of CWE.
ST	**TX**	Valid	Contains the same sort of data. Allowable content differs.

5.3.2 DATA TYPE COMPATIBILITY

Data Type Flavors can be replaced in derived profiles; however, requirements can only be strengthened. In general, the compatibility rules apply for each element in the data type flavor. For example, if a data type flavor specifies an element with R usage, the replacing data type flavor must also specify that element with R usage.

5.4 Content

Content constraints limit the allowed values of an element. Content constraints include coded values, fixed values, pattern restrictions, arbitrary data values, and element relationships.

Content is restricted based on a vocabulary definition that is bound to an element. The vocabulary definition may include a set of codes that can be used to populate an element in a message instance. The code set is associated with a specific element (which is known as "binding"), and the bound vocabulary should only contain codes that are meaningful for that element in a given context (use case). The base standard does not and cannot provide the necessary granularity or depth of specification, because it is designed to have broad utility. Therefore, the base standard provides a starting point, which includes normative (HL7) tables or informative (User) tables or may merely include a placeholder (i.e., a concept domain). The process of profiling elements that are bound to code values includes providing the value set definition, the binding to the element, and the strength of that binding.

Appropriate code set constraints should define and bind a code set containing only the values suitable for a particular data element; often, however, this rule is not followed. A common issue found in specifications is a single code set (e.g., an HL7 table) being applied to multiple elements for the sake of convenience, even though not all values in the value set are meaningful (suitable) for every element. The implementer is left to figure out which values in the given value set are appropriate for the data element in their particular use case.

Vocabulary constraints (coded values) are described in detail in Section 6.

Fixed Value: A fixed value constrains the content to a single value, for example, "A08" that indicates an update event type in ADT messages. The more detailed and specific a certain data exchange specification is, especially in the case of specific use cases, the fewer possible options are valid for a data element. In some cases, the options for an element are reduced to a single value, thus providing a fixed (constant) value. A Fixed Value is represented in the message profile by the fixed value constraint attribute or can be specified by a conformance statement (See Section 5.7).

Pattern-Restricted Value: A pattern-restricted value constrains the content of the element based on the specified pattern matching algorithm (e.g., a Medical Record Number format). A pattern-restricted value is specified using the conformance statement mechanism (See Section 5.7).

Data Values: A list set of non-coded (example) data values can be specified for a given element. Data values may only be specified for elements that represent primitive data types; that is, they have no components or sub-components.

Element Relationships: Elements may have relationship constraints that must be maintained. This includes dependency of data values. Element relationships are supported by the co-constraint construct (see Section 5.8) and conformance statements (See Section 5.7).

5.5 Length

Length is defined as a constraint on the number of characters that may be present in the data value of one occurrence of a data element (object). The definition is system-independent; the number of characters is what is important at the application level. The application must be designed to ensure that storage space is adequate to suit the defined length, even if more bytes are necessary to physically store the data.

The definition and requirements for length in the base standard have changed over various versions. Although specific requirements are not always clear in the base standard, specifiers have the opportunity in message profiles to be definitive in specifying length requirements. Specification of length requirements is optional in message profiles. Additionally, length requirements can be specified selectively for certain elements where length requirements are of concern. Those length requirements that are specified using the methods in this document are deemed to be normative. Length requirements are only applicable to primitive data elements. Length constraints include Minimum and Maximum Length, Conformance Length, and Truncation Indicators.

5.5.1 MINIMUM AND MAXIMUM LENGTH

Length is expressed as either a minimum and maximum pair (e.g. 1..20) and indicates the number of characters an occurrence of an element may have. A constraint on length restricts the range and thus reduces the capabilities, for example, [1..2048] to [1..1024]. In some sense the length constraint is a bit counterintuitive when compared to usage. Usage requires that an application add capabilities, whereas a constraint on length may mean reducing an application's current capability (as configured). The specific needs of a use case will dictate whether length constraints must be applied.

Length is specified using the following syntax: "m..n", where m and n are non-negative integers designating the minimum and maximum number of characters the element may have, respectively, and where n ≥ m. When an upper limit for length cannot be determined in advance, the asterisk character, "*", may be used as a place-holder for the maximum value, so that, in addition to the above syntax where m and n are integer values, a constraint of "m..*" may be used to indicate that the maximum length constraint is unknown. Table 5.15 gives the possible length definitions and their interpretation.

Table 5.15: Minimum and Maximum Length

Length	Interpretation
0..0	Not supported element; minimum and maximum set to 0[26].
1..1	Element must have exactly one character.
1..n	Element may have up to n characters.
n..n	Element must have exactly "n" characters.
1..*	Element may have any length.
m..*	Element may have any length which is greater than or equal to "m", where "m" is greater than or equal to 1.
m..n	Element must have a minimum length of "m" and a maximum length of "n" where "m" is less than or equal to "n" and "m" is greater than or equal to 1.

Length is interpreted as a restriction on an element's data value, not on the presence or absence of the element. Whether or not an element is valued is controlled by cardinality. But if the element is valued with a non-empty value, the minimum and maximum length requirements must be adhered to. The delete indicator (i.e., two double quotes) is not considered to be a value with applicable length information. If the delete indicator is represented by transmitting "", length conforms to any minimum and maximum length specification. A required element can have a delete value, since this still means there is a data value encoded for the element in the message. If an element is empty or not present in the message, i.e., there is no data value encoded for the element in the message, then length restrictions do not apply because there is nothing to restrict and no length constraint that can be violated.

[26] The specifier may choose not to indicate a length at all when the cardinality of the element is 0..0.

Length must not be specified for composite elements. In these cases, the actual minimum and maximum lengths can be very difficult to determine due to the interdependencies on the component content, and the specification of actual lengths is not useful either.

Length compatibility (Table 5.16) is based on a single criterion, that the receiver's capabilities encompass the sender's capabilities.

Table 5.16: Testing Possible Combinations of Implemented Length

	Sender		Receiver	Compatible
Implemented Minimum Length	s_min_len	<	r_min_len	No
	s_min_len	=	r_min_len	Yes
	s_min_len	>	r_min_len	Yes
Implemented Maximum Length	s_max_len	<	r_max_len	Yes
	s_max_len	=	r_max_len	Yes
	s_max_len	>	r_max_len	No

5.5.2 CONFORMANCE LENGTH

Constrainable profile specifications also may specify a conformance length. Conformance length is a constraint that applies to both the sender and the receiver. Conformance length is a constraint on the number of characters that needs to be supported, not the number of characters that need to be sent. Conformance length can be thought of as the minimum for a maximum length. As such, conformance length sets the minimum number of characters that an implementation must support for an element. For example, if a constrainable profile specifies a conformance length of 200, no other profile may assert compliance to the constrainable profile unless its maximum length is 200 or greater. Conformance length is a redundant concept in implementation profiles (it is the maximum length) and must not be specified.

Either the minimum/maximum length or the conformance length can be specified in a profile, but not both. In a derived profile in which a conformance length is defined and changed to a minimum/maximum length pair, the maximum length has to be equal to or greater than the conformance length.

5.5.3 TRUNCATION

The conformance length or maximum length can be indicated with truncation behavior requirements. In the base standard, "=" denotes that the content of the element must not be truncated, and the "#" denotes that the content of the element may be truncated, and, if truncated, the rules for indicating data truncation must be followed.

Message profiles support the concept of truncation with a Boolean truncation flag. In constrainable profiles, the truncation flag can be specified for either the conformance length or the maximum length. In an implementation profile, the truncation flag can be specified for the maximum length. The truncation flag can be set to true or false, or left as unspecified. False signifies that the element must not be truncated (equivalent to the "=" designation), while true

means that the value may be truncated (equivalent to the "#" designation). If unspecified, the default behavior is that truncation is allowed. If a profile sets truncation to false, no other further-constraining profile may mark this value as true. If the value is set to true, other additional-constraining profiles may mark it as true or false. Although the truncation pattern was only defined in v2.7, the behavior may be adopted for previous versions of HL7 in message profiles. As best practice, implementations should not truncate an element even when truncation is allowed. Truncation should only occur when the underlying implementation does not have the capability to support the full length of the element. Allowing truncation of clinical data for any reason can be risky for patients and clinical practice. If the underlying implementation does not have the capability to support the full length of the element, then an evaluation of the risk related to allowing truncation of the data used to populate the element must be performed. Depending on the outcome of this assessment, the capability of the underlying implementation would need to be enhanced to support the clinical requirements for the data. Table 5.17 shows the allowable transition settings for the truncation flag in the process of profiling.

Table 5.17: Truncation Flag Setting Allowable Transitions

Indicator	Transition	Allowed?	Comments
#	#	Yes	Same requirement
#	=	Yes	Requirement Strengthen
=	#	No	Requirement Relaxed
=	=	Yes	Same Requirement

5.5.4 GENERAL LENGTH CONFORMANCE RULES

The following list provides a general set of length rules and observations that need to be considered when applying length constraints in message profiles:

- Length constraint specification is optional in message profiles, either entirely or at specific data elements. If not specified, base standard requirements apply.
- If specified, the following length rules apply:
  - Length constraints can only be applied to primitive data elements.
  - Both the Minimum Length and Maximum Length must be specified.
  - The Minimum Length and Maximum Length must be specified as a range of two non-negative integers in which the Maximum Length must be equal to or greater than the Minimum Length; an exception to these rules applies:
    - In a constrainable profile, the Maximum Length may be unknown and, therefore, may not be specified; in this case, the Maximum Length must be represented as '*", e.g., 1..*.
  - Minimum Length must be 1 or greater (except for elements with X usage).
    - 0..0 would be the length of an element with a usage of X if specified (best practice is to not specify a length).
  - In a derived profile, the Minimum Length must remain the same or can increase (given that the increased value is not greater that the Maximum Length).

- Whereas the base standard (versions 2.3 to 2.6 inclusive) specifies the Maximum Length as normative, it is not considered as a constraint in a message profile. There are contradicting requirements. The specifier, at their discretion, may specify any length constraint that satisfies their use case requirements.
- Whereas earlier versions of the base standard specify the Maximum Length for composite data elements, a message profile must not.
- The Conformance Length constraint is not applicable in an implementation profile and must not be specified.
    - For conformance testing of a message that claims conformance to a constrainable profile, the Conformance Length will be treated as a Maximum Length constraint.
- Whereas Conformance Length is informative in the base standard, Conformance Length is normative in a constrainable profile when explicitly specified in a message profile.
- Whereas the base standard is not prescriptive on the strength of the truncation indicator, the conformance methodology (this specification) is prescriptive and asserts it as normative.

5.6 Slicing

Slicing is a concept that allows for occurrences of a field to be defined with different constraints. A given instance of that field in a message can conform to one of the constraint sets defined for the field. The slicing mechanism has parameters to control which constraints apply to the field instance. In earlier versions of HL7 v2, a field definition was restricted to a single definition. That is, a single set of constraints applied to all instances of a field that might be present in a message. Consequently, the single field definition would include requirements that typically consisted of the superset of possibilities for field instances. Slicing, by allowing multiple definitions of the field, enables a more precise and granular specification of a field. The set of constraints for each "*slice"* of the field definition is realized as a data type flavor. Slicing is an optional mechanism and only applies to field elements.

Slicing addresses specification requirements such as "If patient address type is "home", then use the data type flavor constrained for home address, otherwise use the default data type flavor for address". Another specification may pertain to the order sequence in which various types of addresses are sent, for example, the home address in the first occurrence and the office address in the second occurrence. The specifier will determine, based on the use case requirements, the type of slicing that is needed or if slicing is needed at all. This document provides the concepts, methodology, specification, and profile representation for slicing.

Slicing allows each occurrence of a field to be constrained with a different set of constraints. Individual field occurrence requirements are realized as a data type flavor. The slicing mechanism is limited to fields with a complex data type.

There are three ways to specify the slicing method:

1. Discriminate
2. Ordered
3. Non-selective

The first method uses a discriminator to determine the data type flavor that is used; for example, for a field with an XAD data type the discriminator might be the address type. The second method defines specific requirements based on the order sequence in which the field instance appears. As best practice, use of the discriminator method is preferred, if possible, over the ordered method. The third method is not based on a discriminator or order sequence but provides a list of data type flavors to which a given instance would have to conform.

Figure 5.1 depicts the basic mechanics for slicing a field. The figure shows a field that is specified with a cardinality of 0..* and with a base data type of XAD. It also indicates that the field is "sliced", meaning the field is defined with a set of constraints for each field occurrence. The type of slicing determines which slice of the field is applicable to a given instance of the field in a message. Without the use of slicing, each occurrence of the field is restricted to the same set of constraints.

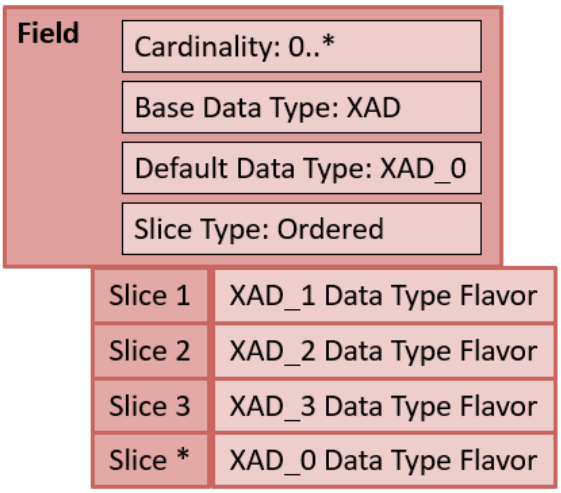

Figure 5.1: Overview of Slicing Concept

In this basic example shown in Figure 5.1, the slice type is *ordered* (to be explained in detail in Section 5.6.1.2), which indicates that occurrences in the message for this field must appear in the order that conforms to the requirements defined in the data type flavor indicated by the field slice. For example, the first field occurrence must follow the requirements specified in the XAD_1 data type flavor. Likewise, the second occurrence must adhere to the requirements defined by XAD_2, and the third occurrence must adhere to the requirements defined by XAD_3. Since no upper limit is defined for this field at this level of profiling, a default set of requirements (XAD_0 data type flavor) is specified for any remaining occurrences of the field that may appear. Note that the slicing does not alter the existing requirements at the field level. That is, the cardinality of the field is 0..*, so a message could omit the field instance and still be conformant. However, if the message did include an instance, the first instance must conform to the constraints indicated by the XAD_1 data type flavor. The sections to follow will explain the slicing types that are supported, and the parameters associated with each type.

5.6.1.1 SLICING USING A DISCRIMINATOR

The slicing type "discriminator" assigns a data type flavor to the field slice (occurrence) based on an evaluation of a discriminator value. The discriminator must be a primitive component[27] within the field. For example, when PID-11.7 (address type) = 'H' then the data type flavor XAD_1 is assigned to the field slice (PID-11). PID-11.7 (address type) is the discriminator and 'H' is the discriminator value. The discriminator value can have three evaluation types:

1. Fixed Value, i.e., a literal value (e.g., PID-11.7 = 'H')
2. Exists, i.e., a Boolean value (Present or Non-presence)
3. Pattern, i.e., a Regular Expression

5.6.1.1.1 FIXED VALUE DISCRIMINATOR SPECIFICATION AND EXAMPLES

Table 5.18 and Table 5.19 show an example of slicing for the Patient Address field using the Fixed Value discriminator method. Table 5.18 shows the field definition for PID-11 (Patient Address). The cardinality is defined as 0..*. The base data type is XAD and the default slice data type is set to the XAD_0 data type flavor. The discriminator is 7 (Address Type), which indicates a component position within the field.

Table 5.18: Field Definition of Patient Address (PID-11): Slice Type = Discriminator Fixed

Field	Min Cardinality	Max Cardinality	Base Datatype	Default Datatype	Discriminator (Position)
PID-11 (Patient Address)	1	*	XAD	XAD_0	Address Type (7)

Table 5.19 shows the definition for each slice. The discriminator indicates the actual value that triggers and defines a "slice". The first row in the table indicates that when (if) the discriminator contains the value of "H" then the field must conform to the requirements defined by the XAD_1 data type flavor. The field slice with these characteristics must appear in the message instance and only can appear once in the message instance. The requirements that dictate whether the field slice must appear are specified by the slice minimum and slice maximum parameters. Likewise, if the discriminator value contains "M" then the field instance follows the requirement indicated by the XAD_2 data type flavor. The field slice with this definition need not appear in the message (Slice Min = 0) or can appear more than once (Slice Max = *). Other discriminator values can also apply, but, since they are not explicitly listed, the default data type flavor defines the requirement for such field instances; in this case, XAD_0 is the data type flavor (see Table 5.19).

[27] A complex component or one of its sub-components cannot be a discriminator; this is a design decision. Future enhancements to the construct will support this capability.

Table 5.19: Slice Definitions

Discriminator Value	Slice Datatype	Slice Min	Slice Max	Comment
“H” (Home)	XAD_1	1	1	If PID-11.7 = “H”, then the requirements for this field slice are defined by XAD_1 data type flavor. A field of this definition must appear once and only once.
“M” (Mailing)	XAD_2	0	*	If PID-11.7 = “M”, then the requirements for this field are defined by the XAD_2 data type flavor. The field slice need not appear or may appear an unlimited number of times.

Slicing is used most often for a field that has multiple occurrences. However, the discriminator method can be used for a field with a maximum cardinality of one. In this case, the field instance requirements vary based on the discriminator value, even though there can be only one instance of the field. For example, the discriminator could be the address type as in the previous example. If the address type is “H”, then the field instance would follow the requirements specified in the XAD_1 data type flavor. Likewise, if the address type is “M”, then the field instance would follow the requirements specified in the XAD_2 data type flavor. So, the field would follow one or the other definition based on the address type value (discriminator value) and in aggregate can only appear once, since the cardinality of the field is 1..1. Note, for this example, the default data type is not defined by specifier choice. Table 5.20 and Table 5.21 show the specification of this example.

Table 5.20: Field Definition of Patient Address (PID-11): Slice Type = Discriminator Fixed

Field	Min Cardinality	Max Cardinality		Base Datatype	Default Datatype	Discriminator (Position)
PID-11 (Patient Address)	1	1		XAD	NA	Address Type (7)

Table 5.21: Slice Definitions

Discriminator Value	Slice Datatype	Slice Min	Slice Max	Comment
“H” (Home)	XAD_1	0	1	If PID-11.7 = “H”, then the requirements for this field slice are defined by XAD_1 data type flavor. A field of this definition need not appear and at

Discriminator Value	Slice Datatype	Slice Min	Slice Max	Comment
				most will appear once.
"M" (Mailing)	XAD_2	0	1	If PID-11.7 = "M", then the requirements for this field are defined by the XAD_2 data type flavor. A field of this definition need not appear and at most will appear once.

5.6.1.1.2 EXISTS DISCRIMINATOR SPECIFICATION AND EXAMPLES

The "Exists" discriminator methods work much like the "Fixed Value" discriminator method except that the discriminator value[28] can be either "valued" (True) or "not-valued" (False). Only those two possibilities exist. Table 5.22 and Table 5.23 provide examples. The field definition defines the minimum and maximum cardinality for the field and the discriminator. However, there is no default data type flavor because the slicing definition (Table 5.23) covers all possibilities.

Table 5.22: Field Definition of Patient Address (PID-11): Slice Type = Discriminator Exists

Field	Min Cardinality	Max Cardinality	Base Datatype	Default Datatype	Discriminator (Position)
PID-11 (Patient Address)	1	*	XAD	NA	Address Type (7)

If only one of the discriminator values is specified, then a default data type flavor is necessary (in essence, the default is specifying the requirements for the slice in which the discriminator value was indicated).

Table 5.23: Slice Definitions

Discriminator Value	Slice Datatype	Slice Min	Slice Max	Comment
True	XAD_1	0	*	If PID-11.7 is valued, then the requirements for this field slice are defined by XAD_1 data type flavor. The field slice need not appear or may appear an unlimited number of times.
False	XAD_2	0	1	If PID-11.7 is not valued, then the requirements for

[28] Note: the component usage cannot be R or X for the Exists slicing type.

				this field are defined by the XAD_2 data type flavor. A field of this definition need not appear and at most will appear once.

5.6.1.1.3 PATTERN DISCRIMINATOR SPECIFICATION AND EXAMPLES

The "Pattern" discriminator method uses a pattern to determine the slice that is triggered. Table 5.24 and Table 5.25 illustrate an example. The use case in this scenario is that if a postal code is valid for the United States or Canada, then specific constraints are specified for the field instance. If the postal code is neither, then a default set of constraints are applied. Note that for the United States, two types of postal codes are acceptable, hence there are two separate slices that refer to the same data type flavor (XAD_1). The discriminator values are represented as a pattern, and the discriminator (position) is PID-11.5 (ZIP or Postal Code).

Table 5.24: Field Definition of Patient Address (PID-11): Slice Type = Discriminator Pattern

Field	Min Cardinality	Max Cardinality	Base Datatype	Default Datatype	Discriminator (Position)
PID-11 (Patient Address)	1	*	XAD	XAD_0	ZIP or Postal Code (5)

The above example shows how an application can handle different requirements of XAD based on the postal code. In most cases, a message is likely to contain only one type or the other, but both could be supported in the same message for a patient. This specification supports either configuration. Note that a similar result could be accomplished using conditional usage.

Table 5.25: Slice Definitions

Discriminator Value	Slice Datatype	Slice Min	Slice Max	Comment
[0-9]{5}	XAD_1	0	*	If PID-11.5 matches the five-digit pattern for United States zip code, then the requirements for this field slice are defined by XAD_1 data type flavor. The field slice need not appear or may appear an unlimited number of times.
[0-9]{5}-[0-9]{4}	XAD_1	0	*	If PID-11.5 matches the nine-digit pattern for United States zip code, then the requirements for this field slice are defined by XAD_1 data type

Discriminator Value	Slice Datatype	Slice Min	Slice Max	Comment
				flavor. The field slice need not appear or may appear an unlimited number of times.
(?!.*[DFIOQU])[A-VXY][0-9][A-Z] ?[0-9][A-Z][0-9]	XAD_2	0	*	If PID-11.5 matches the pattern for Canada postal code, then the requirements for this field are defined by the XAD_2 data type flavor. The field slice need not appear or may appear an unlimited number of times.

5.6.1.2 SLICING USING ORDERING

Ordered slicing defines specific requirements based on the order sequence in which the given field instance appears. That is, the order sequence in which the field instance appears is significant for the use case. Table 5.26 illustrates a field definition for Patient Name (PID-5). Table 5.27 defines the requirements (data type flavor) for the associated slice occurrence. For the first occurrence of PID-5 the data type definition is XPN_1, for the second occurrence of PID-5 the data type definition is XPN_2, and for any other occurrence of PID-5 the data type definition is XPN_0 (the default data type flavor assigned).

Table 5.26: Field Definition of Patient Name (PID-5): Slice Type = Ordered

Field	Min Cardinality	Max Cardinality	Base Datatype	Default Datatype
PID-5 (Patient Name)	1	*	XPN	XPN_0

This strategy can be combined with the data type flavors and constant values to specify particular occurrences of the type of patient name. For example, if it were necessary to constrain the Patient Name field tightly such that the first occurrence of the field is the legal name (PID-5.7 = ‘L’), then the data type flavor, XPN_1 in this example, can set the XPN.7 (Name Type Code) to ‘L’ as a constant value.

Table 5.27: PID-5 Ordered Slicing Definition

Occurrence	Slice Datatype
1	XPN_1
2	XPN_2

The occurrence value is not required to be specified for each field instance nor is an order sequence required. For example, if a specifier only wanted to assign a specific data type flavor to the second occurrence, then only the second occurrence would be specified. All other occurrences, including occurrence one, would follow the requirements defined by the default data type flavor for Patient Name.

5.6.1.3 NON-SELECTIVE SLICING

The "Non-selective" slicing method can be employed in cases where the specifier wishes the field instances to follow the constraints of one of a number of constraint sets (data type flavors). Table 5.28 and Table 5.29 illustrate the use case.

Table 5.28: Field Definition of Patient Address (PID-11): Slice Type = Non-selective

Field	Min Cardinality	Max Cardinality	Base Datatype
PID-11	0	*	XAD

In this specification, the PID-11 can occur an unlimited number of times. A field occurrence must follow the constraints of any one of the three data type flavors (XAD_1, XAD_2, XAD_3)

Table 5.29: PID-5 Ordered Slicing Definition

Slice Datatype List	Comment
XAD_1, XAD_2, XAD_3	The field instance must adhere to the constraints specified in one of the specified data type flavors (XAD_1, XAD_2 or XAD_3).

5.7 Conformance Statement

A conformance statement[29] is a mechanism to express a constraint using a natural or computable language. An example is:

> "IF RXA-18 (Substance/Treatment Refusal Reason) is valued THEN RXA-20 (Completion Status) SHALL contain the value 'RE' (Refused)".

Conformance statements provide a catch-all mechanism for expressing a constraint that can't be defined by the constraint types provided by the other conformance constructs (e.g., usage). Conformance statements should not be used in place of the conformance constructs; that is, do not use a conformance statement to constrain the usage of an element (e.g., "PID-8 (Administrative Sex) is Required.").

[29] Conformance statement is an overloaded term used in other contexts to indicate, from a high level, the capabilities of an implementation or system (i.e., a set of things that the implementation or system is claiming it can do). In the context of this section, conformance statement simply refers to a statement of a requirement.

The forms in which a conformance statement can be expressed vary and can appear as a text description or a computable expression. In both forms a specific syntax should be used to ensure that the requirement details are complete and to improve readability for the implementer. This specification does not require a specific method or language. However, a pseudo language specifically for HL7 v2 has been developed and is the recommended method for expressing conformance statements (See Appendix B).

5.8 Co-Constraints

The co-constraint construct is a related constraint concept that is used to express dependencies among a set of data values. An HL7 v2 message in which observations are conveyed is one example of how co-constraints are used. Such constraints typically follow the form of "if OBX-3.1 = LOINC code *XXXXX-X* then OBX-2 SHALL BE "CE" and OBX-5.1 SHALL contain a value from the value set *YYY*" or a similar form. A co-constraint is not limited to use with observations, but it appears most frequently with observations. A convenient way to illustrate a set of co-constraints is to present them in a table format. Table 5.30 illustrates an example for a set of immunization observations.

Table 5.30: Excerpt of Co-Constraints of Immunization Observations

LOINC (OBX-3)	Description	Data Type (OBX-2)	Data Type Flavor (OBX-2)	Value Set (OBX-5)	Units (OBX-6)
30973-2	Dose number in series	NM	NM	Not applicable	"NA" from HL70353
59782-3	Number of doses in series	NM	NM	Not applicable	"NA" from HL70353
59783-1	Status in Immunization series	CE	CE_01	Local Value Set	
30956-7	Vaccine Type	CE	CE_01	CVX - Vaccine Group	
30980-7	Date next dose is due	DT	DT_D	Not applicable	
59779-9	Immunization Schedule	CE	CE_01	ScheduleIdentifier	
30963-3	Vaccine funding source	CE	CE_01	FundingSource	

Usually the OBX-3 (Observation Identifier) is the key upon which data dependencies are based. For example, when OBX-3 contains the LOINC code 30956-7 (for Vaccine Type), the data type in OBX-2 must be "CE" (for coded element), and OBX-5 (the Observation Value) must contain a code from the CVX value set (specifically from the Vaccine Group). The data type representation for OBX-5 is given by the data type flavor CE_01 definition. This example shows how co-constraints are typically expressed in HL7 v2. Other mechanisms can also be used. Additionally, co-constraints can be grouped or predicated on a condition. For example, a group of OBX

segments with interdependences may be required predicated on a certain value in the segment group (e.g., OBR-4).

5.9 Semantic Refinement

Every element has associated data semantics that define the sort of data the element is carrying and how those data should be interpreted. That is, the data element is bound to a concept (logical meaning). The base standard provides these element definitions. In some instances, the definitions are broad, because the context of a given use case is unknown. Profiles, therefore, can refine the semantics of a data element based on the use case. This refinement is in prose, and hence not conducive to computerized understanding; that is, implementers will have to read and adjust implementation accordingly. Best practice is to provide only refined element semantics in profiles, and not to repeat existing element definitions from the base standard. HL7 v2 profiles have an Annotation mechanism that supports documentation of semantic refinements in a structured way.

5.9.1 ANNOTATIONS

An annotation is descriptive text that accompanies a standard element definition or concept and provides additional information pertaining to the use of the element (i.e., it is an elaboration of the concept as it relates to the use case to which it is being applied). An annotation may be associated at any defined element level in a message profile (e.g., a field). HL7 v2 supports a number of pre-defined annotation types:

- Definition: An explanation of the meaning of the element.
- Description: An explanation of the associated element. This may contain formatting markup.
- Design Comment: Internal development note about why particular design decisions were made, outstanding issues and remaining work. They may contain formatting markup. Not intended for external publication.
- Implementation Note: Provides a general description about how the element is intended to be used, as well as hints on using or interpreting it.
- Other Annotation: Additional content related to the element.
- Example: An example instance.

Annotations do not introduce new requirements in the form of constraints or extensions; they are informative and can be used to supplement the understanding of the element.

6 VOCABULARY CONSTRAINTS

The purpose of vocabulary profiling is to narrow the allowable content of data elements that are bound to coded data types such as ID, IS, CE, CWE, and CNE[30]. Vocabulary is a term used to describe, in general, the mechanics of defining, maintaining, and using codes to represent concepts in healthcare data exchange. An understanding of the vocabulary terms and principles is necessary before describing vocabulary profiling. A number of key concepts are defined:

Coded Data Element: is an element with a data type definition that supports coded concepts. Examples include IS, ID, CE, CWE, CNE, and any data type flavors of coded data type definitions (e.g., CWE_01).

Code System: is a managed collection of codes that represent concepts used in a particular business or technical area and in which often there are relationships between the coded concepts. Code systems are developed to provide a set of coded concepts for a particular domain, and they are designed for one or many specific or general business uses. A code system may be a simple list of coded concepts, or it may be designed with one or more explicit relationships between the coded concepts (at least hierarchical, and often many other types of relationships in a multi-dimensional fashion). As an example, a concept domain of "Administrative Gender" can contain a concept "male" and a code "M" that represents the concept "male" within the context of the code system.

Value Set: is a collection of codes targeted for a specific use. A value set draws codes from one (usually) or more code systems; the result is a set of codes that constrains the possible content of a data element. A key distinction between a value set and a code system is that a code system is used as a reference source of coded meaning whereas a value set is a specific constraint for a set of explicit business uses. Ultimately, binding of data elements to value sets is required for implementation.

Value Set Examples: The Logical Observation Identifiers Names and Codes (LOINC) is a code system that has tens of thousands of observations. It is widely used in healthcare data exchange standards to specify all existing laboratory tests in current use internationally as well as the most common clinical observations. Two examples of value sets that draw from the LOINC code system are the set of codes that specify radiology study types and the set of codes that just specify laboratory observations. Another example value set that draws from the LOINC code system is the NIP003 value set that contains immunization observations such as "30963-3" for "Vaccine Funding Source". The NIP003 value set includes only a few dozen codes out of the tens of thousands of LOINC codes.

Concept Domain: is an abstract notion that refers to a set of related ideas (concepts) that serve to help define the meaning of a particular data element (over and above just the name of the data element). A concept domain does not directly define a particular set of concepts and is independent from a specific and explicit association to a particular vocabulary (code system or value set). The concept domain of postal code (ZIP code in the US) is a good example.

[30] However, in message profiles non-coded data types such as ST (string) can be bound to a vocabulary, e.g., PID-23 (Birth Place).

Tables: HL7 v2 defines a set of pre-defined tables that contain identifiers and may also contain codes. A table sometimes acts like a code system, value set, and/or a concept domain depending on how it is defined and used. Tables in HL7 v2 have two types: *HL7 Table* and *User Table*. An *HL7 Table* includes values that must be used if the concept to be conveyed is contained in the table. *HL7 Tables* can be extended for inclusion of concepts not pre-defined in the table. A table could also be categorized as a *User Table*; this kind of table provides a list of suggested values that could be used, or, alternatively, a different set of values could be defined and used. The *User Table* could be empty with no values suggested, which indicates that this entire set of values must be defined by a specifier. Tables have a pre-defined identifier—the table number—that helps in managing the tables and in binding them to a data element.

In the context of profiling, the pre-defined set of tables in HL7 v2 can be used as a starting point for defining value sets. In different cases, a Table might be constrained (like a code system), extended, replaced (acting like a concept domain), or used exactly as-is (acting like a value set).

6.1 Vocabulary Profiling Mechanics

At a high level, the process of defining vocabulary requirements can be divided into two distinct aspects: (1) Binding and (2) Vocabulary Definition.

Binding consists of the following parts:

1. Binding: Associates a data element to a vocabulary (Concept Domain, Code System, or Value Set).
2. Binding Strength: Indicates whether the binding is required or suggested.
3. Binding Parameters: Defines allowable options for value set modifications in derived profiles. Binding parameters include extensibility and stability.
4. Binding Location: For complex coded data types, this parameter indicates the specific location to which the binding refers (e.g., the first triplet in a CWE data type).

The Vocabulary Definition consists of the following parts:

1. Vocabulary Definition: Metadata describing the vocabulary specification.
2. Vocabulary Codes: List of coded concepts (enumerated or expanded by algorithm).
3. Vocabulary Code Usage: Defines the use of an individual code.

In a specification, not all aspects of vocabulary constraints need to be defined, and often their definition will depend on the profile level. For example, a constrainable profile can bind an element to a concept domain. It is expected that a derived profile would further define the vocabulary constraints.

Binding: is the association of a coded data element to a vocabulary. Depending on the level of the specification, the binding can be to a concept domain, code system, or value set. The HL7 v2 tables can represent any of these three vocabulary types depending on how the table is defined and used. At each specification (profile) level, the binding becomes increasingly specific, refining the data semantics of the element by limiting the vocabulary's content to a particular set of coded values. For example, at the base standard level just a concept domain may be specified, and at the profile level a value set containing explicit codes for an implementation may be specified. In profiling, the HL7 v2 tables often are the starting point for value set creation.

Table 6.1 illustrates a binding of the coded data element PID-8 (Administrative Sex) to the vocabulary identifier by "HL70001_EX". The vocabulary definition referenced by "HL70001_EX" reveals the details of the binding. Likewise, PID-10 ("Race") is bound to the vocabulary definition identified by "HL70005". In these examples, "HL70001_EX" is a

modified set of codes that is based on the standard set of codes specified in the HL70001 table, and “HL70005” is used as-is.

Table 6.1: Binding Examples

Patient Identification (PID) Segment Definition Excerpt				
SEQ	**Name**	**DT**	**Usage**	**Vocabulary**
…	…	…	…	…
7	Date/Time of Birth	TS	R	
8	Administrative Sex	IS	R	HL70001_EX
9	Patient Alias		X	
10	Race	CE	RE	HL70005
…	…	…	…	…

Binding Strength: indicates the conformance of the binding, that is, whether the vocabulary[31] must be used or not. There are two possible values: Required (R) and Suggested (S) (Recommended). All bindings are eventually required in implementations, and a suggested binding can be specified in a constrainable profile (i.e., the specification needs to be further constrained before implementation). The bound vocabulary definition with a binding strength of “Suggested” can be used as-is, modified, or replaced by an alternative code set in derived profiles[32]. Bindings to concept domains and code systems[33] always have a binding strength of “Suggested”[34], because at the implementation level, all vocabulary bindings are to value sets (i.e., a definitive set of codes have been selected). Table 6.2 shows a required binding strength for HL70001_EX to PID-8 and a suggested binding strength for HL70005 for PID-10.

Table 6.2: Binding Examples with Binding Strength

Patient Identification (PID) Segment Definition Excerpt					
SEQ	**Name**	**DT**	**Usage**	**Vocabulary**	**Strength**
…	…	…	…	…	
7	Date/Time of Birth	TS	R		
8	Administrative Sex	IS	R	HL70001_EX	R
9	Patient Alias		X		
10	Race	CE	RE	HL70005	S
…	…	…	…	…	

[31] In this context, vocabulary is used to refer to concept domain, code system, or value set.

[32] However, if the binding from the base standard is to an HL7 table, then concepts from that table definition must be used if needed in the profile.

[33] However, a value set can use an entire code system.

[34] This does not mean the semantics of the concept domain can change; that is, any set of codes can be specified but the semantics of those codes must be in line with the defined concept domain.

In constrainable profiles, the binding strength can initially be left unspecified (in essence, a binding strength of suggested) and then set once the use case or implementation requirements are better known. For implementation profiles, the binding strength must be set to required.

It is important in profiles to set the binding strength explicitly to required when the intent is to require a set of codes for use. This declaration is necessary to avoid confusion related to the implications of the underlying data type of the element with which the binding is associated. For example, the "IS" data type[35] indicates that the element is associated with a *User Table*, but this association is not the desired specification in the profile.

No aspects of binding should be specified for coded elements with a usage of X (Cardinality of 0..0), as they are of no consequence.

Binding Parameters: define allowable options for code set modifications in derived profiles. Binding parameters include **extensibility** and **stability**.

Extensibility: indicates whether the value set definition can be extended or not in a derived (profiled) version of the specification. The extensibility of a value set definition can be either *open* or *closed*. Open means that more values can be added to the value set in a derived specification[36]. Closed means that no more values can be added to the value set in a derived specification.

Stability: indicates whether the content of a value set can change outside of the definition of the value set or the specification in which it is bound to one or more data elements. Often the value set is dependent on an external code system that may be updated or replaced after publication of the interoperability specification. Stability can have one of two values: *static* or *dynamic*. Static indicates that for this version of the specification the value set is completely fixed, both its definition and the expanded list of codes. If the value set needs to be modified, then a new value set and an updated version of the specification must be created. Dynamic means that a value set can change outside of the definition of the specification (and thereby becoming, in essence, a new value set that may be managed and referred to in various ways). In some instances, dynamic value sets are entire code systems that periodically are updated.

Implementation Note: The CVX code system (used for immunization messaging in the US) and the German PZN (pharmaceutical central numbers) are good examples of when value sets with dynamic stability would be included in a specification. A new vaccine may be developed or an existing one deprecated and replaced with another. Another good example is the set of codes used for diagnoses. In Germany, a new code system for this purpose is released every year (e.g., "ICD-10 German Modification Release 2016"). In order to avoid the need for an updated release of the specification each year, the binding to a value set with dynamic stability is established. Under these circumstances the latest version[37] of the value set should be used. How

[35] A proposal for future versions of HL7 v2 is to replace coded data types with a "simple code data type" and a "complex coded data type" and divorce the vocabulary implications from their definitions. These aspects will be separately defined in other binding mechanisms.

[36] Only for concepts that don't exist; i.e., a new code can't be added for an existing concept.

[37] In essence a new value set has been created.

trading partners convey the use of the updated value set varies in practice. In some cases, it may be critical for them to be in harmony with the latest version immediately, while in others it may not be. When a prior agreement about this harmonization is not negotiated, the use of the conformance claim is helpful. The Message Profile Identifier element (MSH-21) can declare the value set being used (the value set can be managed as a profile component).

An overview of the vocabulary profiling mechanics is given in Figure 6.1. Point (1) indicates an element with a data type that supports coded data. A **Value Set Binding**, in simple terms, refers to assigning a set of coded concepts to a particular data element in order to implement the concept domain associated with the data element. A binding is thus to a value set and is what is indicated as point (2). In Figure 6.1, the Administrative Sex data element (1) in the Patient Identification Definition is bound to the value set identified by HL70001_EX in the value set column (2). The Value Set Identifier is an attribute of the value set definition (6). The **Binding Strength** (3) indicates the conformance of the binding; that is, whether the vocabulary must be used or not.

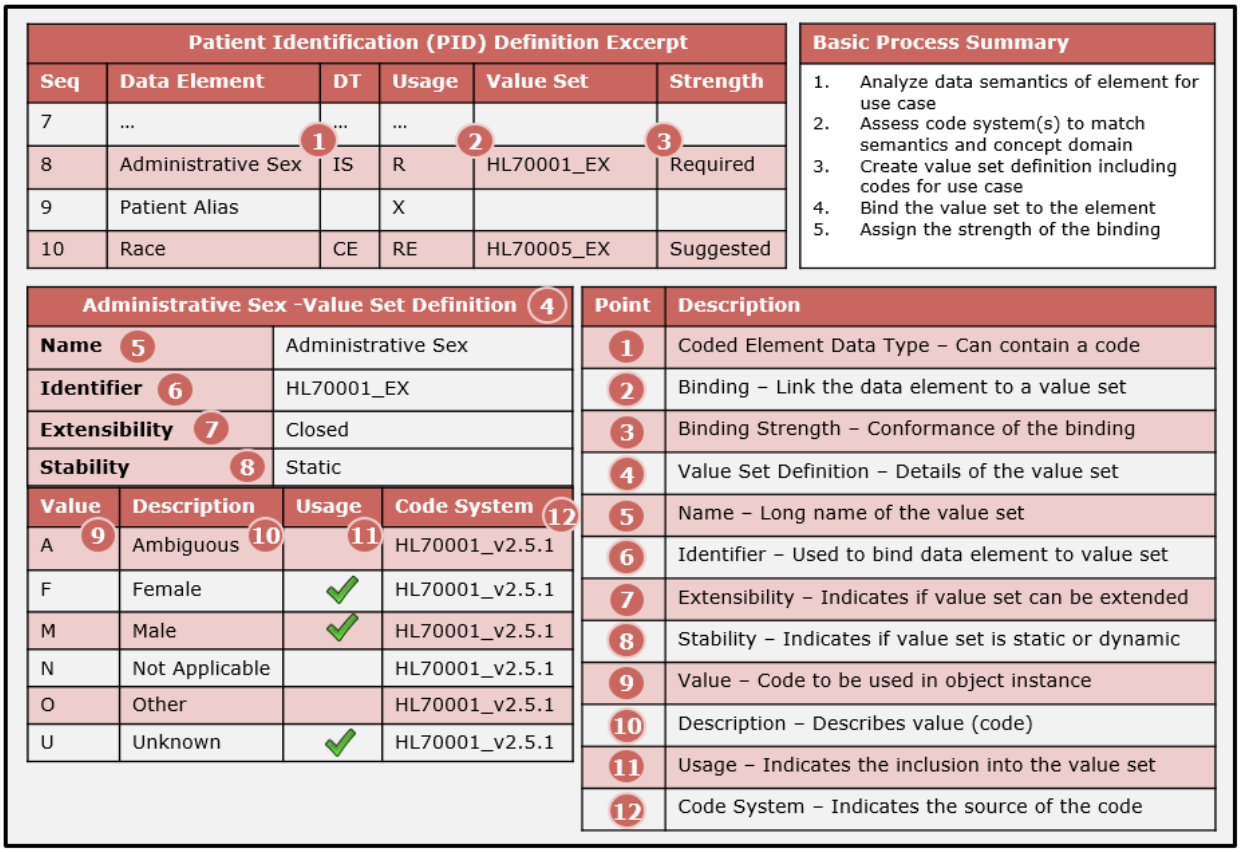

Figure 6.1: Summary of vocabulary mechanics

The **Value Set Definition** (4) provides meta data and the codes[38]. Meta data shown in Figure 6.1 include the name of the value set (5), the identifier (6), and value set properties extensibility (7) and stability (8). The codes (9) contained in the value set are F, M, and U, as indicated by the check marks[39] in the usage column (11). For each code (value), a description is given (10). The value set is composed of codes from the HL70001 v2.5.1 code system[40] (12). Note: a value set may include codes from multiple code systems. The value set definition shown in Figure 6.1 shows all of the codes in the code system (A, F, M, N, O, U) and the codes selected for inclusion into the value set (F, M, U). A value set definition may or may not include the complete set of codes in the source code system. The complete set is presented in this example to show the value set in the context of the source code system. Other forms of representation are equally acceptable as long as sufficient information is provided to indicate the status of the code. In this example value set, the usage[41] is limited to inclusion or exclusion. Additional specificity can be associated with the code usage, the methodology is presented in Section 6.5. Here, vocabulary usage is

[38] To simplify explanation this definition includes meta data and the codes. See recent work in HL7 WG and discussion to follow in which the Value Set Definition consists of the meta data and *information to determine* the codes. The generated list is known as the Value Set Expansion.

[39] Check marks are used for simplicity and replaced by more detailed usage concepts Section 6.1.5.

[40] As mentioned, HL7 v2.x tables are not yet formally code systems, and HL70001 is used here as a simple example. For convenience, HL7 v2 tables are referred to as code systems.

[41] The term "Usage" here is not related to the usage of elements (e.g., R, RE, etc.) described earlier in this section; See Section 6.1.5 for details of vocabulary usage.

defined in terms of the exclusion, definite inclusion, or possible inclusion of a code in a value set.

The **Content Definition** of the value set indicates the method used to expand a value set definition into a set of codes. The example presented in Table 6.1 is an explicit listing (i.e., enumeration). An enumerated value set definition is called an **Extensional** definition. Other means used to expand a value set definition into a set of codes include using regular expressions, characterizing specific attributes for a specific code system (e.g., all laboratory codes in LOINC), or using hierarchical associations that include all specializations of the codes in question. By applying these kinds of mechanisms, a value set definition can be expanded to the **Value Set Expansion** providing all codes required for the identified business use in the implementation. This form is called an **Intensional** definition. An Intensional definition should include how to obtain the value set expansion.

6.2 Typical Vocabulary Bindings

Figure 6.2 indicates the types of vocabulary bindings that are typical for each profiling level. The base standard level supports broad use cases, so it follows that the vocabulary bindings are typically at the concept domain and code system level. As use cases are developed and refinement of a specification occurs at the constrainable profile level, more information becomes known, and, therefore, relevant value sets begin to emerge. At the implementation level, all coded elements must be linked to a specific value set. In some implementation-level specifications, an element is bound to a code system, which is then deemed implicitly to be the value set (although explicit designation is recommended).

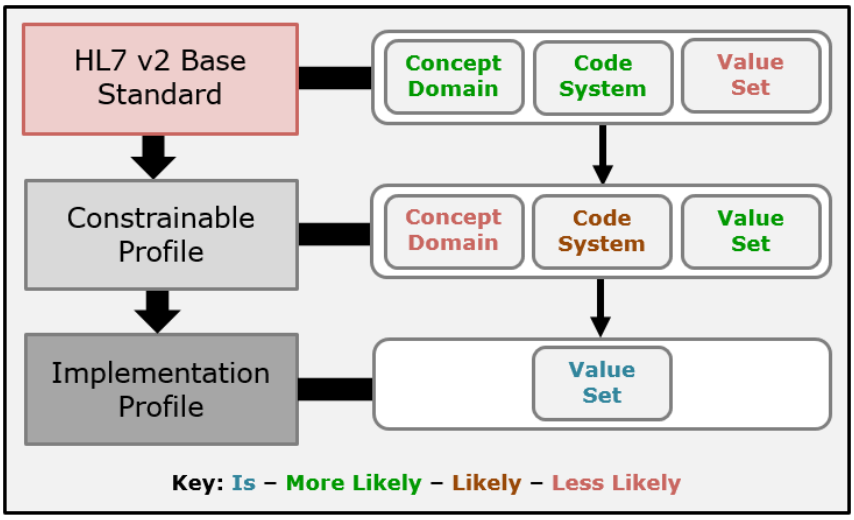

Figure 6.2: Typical Vocabulary Bindings

6.3 Single Code Element Binding

A **single code element binding** is a vocabulary binding in which the binding to an element is to a single value selected from a vocabulary definition. Through analysis, it is determined that only a single code is suitable for the use case. In such circumstances, the binding is to a constant value and a special binding mechanism can be applied. The single code element binding constraint

pattern allows for the vocabulary binding without having to employ the complete vocabulary mechanics process. The constraint pattern is shown below:

> **Constraint Pattern:** Element Location (Element Name) SHALL contain the constant value 'Code' drawn from the 'Code System' code system.

A typical example is for MSH-9 values and MSH-12 (Version ID). The example below is for the message event defined in MSH-9.2.

> **Example:** MSH-9.2 (Event Type) SHALL contain the constant value 'A04' drawn from the 'HL70003' code system.

The single code element binding is a convenience mechanism for specifying a complete vocabulary binding in a single statement.

6.4 Use of Extensibility and Stability

Figure 6.3 shows value set attributes of Extensibility and Stability, along with the allowable settings these attributes may take on related to the specification space and their impact in post-specification and runtime environments. The requirements for value sets are determined when specifications are created initially in the base standard, and then these requirements are subsequently refined through the mechanism of profiling. The specification (and the impact of that specification) of the extensibility and stability for a value set can be considered in three distinct spaces: the specification space, post-specification space, and operational space. In the specification space, the extensibility of a value set can be designated as "open" or "closed". Constrainable profiles (which include the base standard) can specify value set extensibility as either "open" or "closed". For implementation profiles, the value set extensibility is closed. At this point in the creation of the specification, the use case is definitive; therefore, the universe of allowable values for the element is known and is documented as such. Any desired change would have to occur in another version of the specification with a new value set. Extensibility is a construct in the context of the specification space; modification of a value set outside the specification space is controlled by Stability and is discussed next.

There are circumstances when, although the complete state is known, it is anticipated that additional updates to a value set will be necessary. In some cases, the updates can be managed, published, and communicated in an orderly manner. This situation occurs in the post-specification space, and it is indicated by the value set having a stability of "dynamic".

For most internal value set bindings (i.e., HL7 tables), the settings for extensibility and stability predominantly are "closed" and "static" (and for implementation profiles extensibility must be "closed"). A stability of "static" is appropriate for value sets where all concepts needed for the use case are known at the time of specification. These value sets may also have dependent concepts or those in which the value set is tightly-coupled to the underlying technical infrastructure; that is, the concepts have certain dependencies within the use case, and any change that is made has a significant and consequential impact on implementations. For example, if a value set for laboratory results defined a *state machine* using various status codes (preliminary, final, corrected, etc.), it would be unacceptable to allow another *state* to be defined outside the bounds of the specification. This example is one in which the value set codes are tightly-coupled to the technical infrastructure. A more subtle example is the definition of the abnormal flags (HL7 v2 Table 0078) that indicate the result interpretation. The code "H" indicates "above high normal" and "HH" indicates "above upper panic limits". If an intermediate setting is needed, for example, "HU" for "very high", then this concept should be defined as part

of a new version of the interoperability specification and value set (and not as part of a local value set or a dynamic value set). Adding an intermediate value changes the interpretation of the existing result interpretations in the value set, which, in turn, is a patient safety issue (in some circumstances).

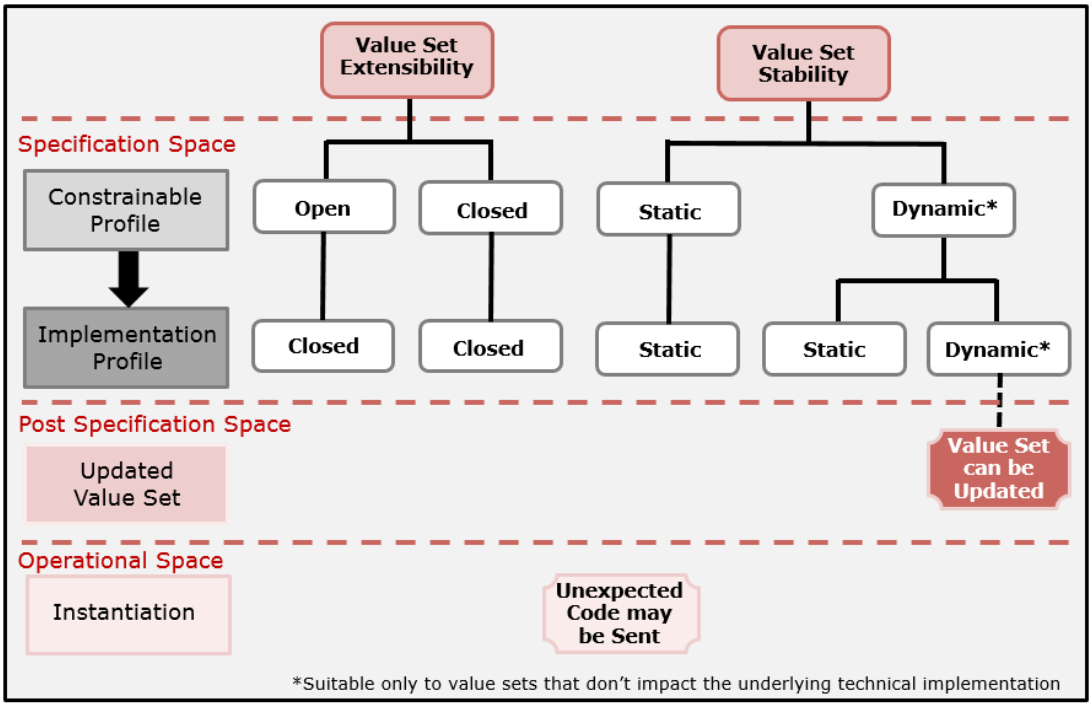

Figure 6.3: Extensibility and Stability Use in Specifications

Other value sets that contain independent concepts can have a stability of "dynamic". For example, ICD-10 codes could be updated and published periodically (as a new version or a new code system). This update is outside the scope of the published version of the specification that references the ICD-10 code system as a value set. Tagging the value set as dynamic, however, is a notification to implementers that the value set can change and that they should check for and make their updates accordingly.

Certain real-world circumstances necessitate communication of an unknown code. An example might be when a new strain of a virus is detected that must be reported to public health agencies. A code for this virus could be established (without formal recognition of a code system steward[42]) and reportable lab result messages with this code could be sent immediately, even though the receiver may not know how to handle the new code. If the code can be handled by "generic" processing, however, a complete understanding of it by the receiver may not be necessary. Also, when the code and the associated concept are published officially, receivers that update to this new value set can then process this information in a meaningful way (if so desired). This circumstance is indicated in the "Operational Space" as an "Unexpected Code may be Sent".

[42] Although the code would be expected to be forthcoming. The necessity of having a code for the concept immediately prevents an orderly deliberation and document process (which will happen later).

In this circumstance, there is no pre-coordinated agreement about this code, but there can be a pre-coordinated agreement that an unknown code is possible for a given element. It is important that standards provide an explicit and distinct mechanism to support this notion such that it is not conflated with other mechanisms, which dilutes the specification and lessens the capabilities for validation. HL7 v2 introduced the concept of **Implementation Tolerance** that allows for unknown (undocumented) codes to be exchanged at runtime with impunity, because the complete code set is not known to the implementers. For a binding that supports implementation tolerance, an implementation must process unknown codes without raising an exception.

Implementation Tolerance: There are certain use cases that necessitate sending unknown codes. A profile can declare implementation tolerance for a given vocabulary binding.

It should be pointed out that, regardless of the use of implementation tolerance, implementations always have the latitude to be tolerant. A conformance violation may be detected, but the implementation can process the data in whatever manner they see fit.

It is important that vocabulary specification provide a way to indicate whether a data element definition allows or disallows unexpected (unpublished) codes. The mechanism should not be conflated with existing mechanisms (i.e., Extensibility and Stability). Implementers must be able to distinguish concepts of extensibility and stability from the case of unexpected codes in operational environments. Such information enables precise specifications and robust implementations. Systems can handle the "expected" unexpected code for data elements specified with this attribute and report a violation for data elements in which unexpected codes are not allowed. For most data elements, unanticipated and unexpected codes would not be expected (in well-defined specifications).

6.5 Profiling at the Code Level

Similar to, but not equivalent to, usage for data elements is the concept of usage for the codes in a value set. Vocabulary usage is a mechanism for defining and setting the scope of the concepts to be included, considered for inclusion (or exclusion), or excluded in a value set definition.

Table 6.3 defines usage indicators for profiling vocabulary. Required (R) usage indicates that the code must be supported (and thus can be used); Permitted (P) indicates that the code can be *profiled* to R, P, or E in a derived profile; and Excluded (E) indicates that the code must not be supported (and thus cannot be used) for the given element. The Permitted usage indicator is only applicable to constrainable profiles; that is, it cannot appear in an implementation profile (it must become an R or an E). Permitted usage can be included in a closed value set (at the constrainable profile level). In this situation, the set of all known concepts are defined, but some concepts are left to be profiled at the implementation level based on local requirements/needs.

Table 6.3: Vocabulary Profiling Usage

Usage	Name	Conformance	Allowable Usage[43]
R	Required	The code SHALL be supported.	R
P	Permitted (applicable to constrainable profiles only)	None; to be specified.	R, P, E
E	Excluded	The code SHALL NOT be supported.	E

Figure 6.4 indicates the relationship of vocabulary usage to the profiling hierarchy. The value set is deemed to be "scoped" (at the constrainable profile level). At the base standard level, all codes have an implied usage of Permitted[44]. The base standard defines a set of concepts and the representative codes. The binding strength determines how the value set can be used in derived specifications. If the binding strength is "Required", and if the concept that needs to be conveyed is available in the code set, then the code must be used. The use case analysis determines which codes are applicable for the element to which the value set is bound. An important point to make here is that the base standard would have defined a set of codes with associated concepts. A binding strength of Required means that the value set must be used; however, only the codes that are appropriate to use (based on use case analysis) for the element to which the value set is bound should be specified in derived profiles. This analysis should be completed for every element to which the base vocabulary is bound. In practice, specification at this granular level rarely occurs (but should) in implementation guides.

[43] Allowable usage in a derived profile.

[44] Although this is not explicitly stated but can be inferred.

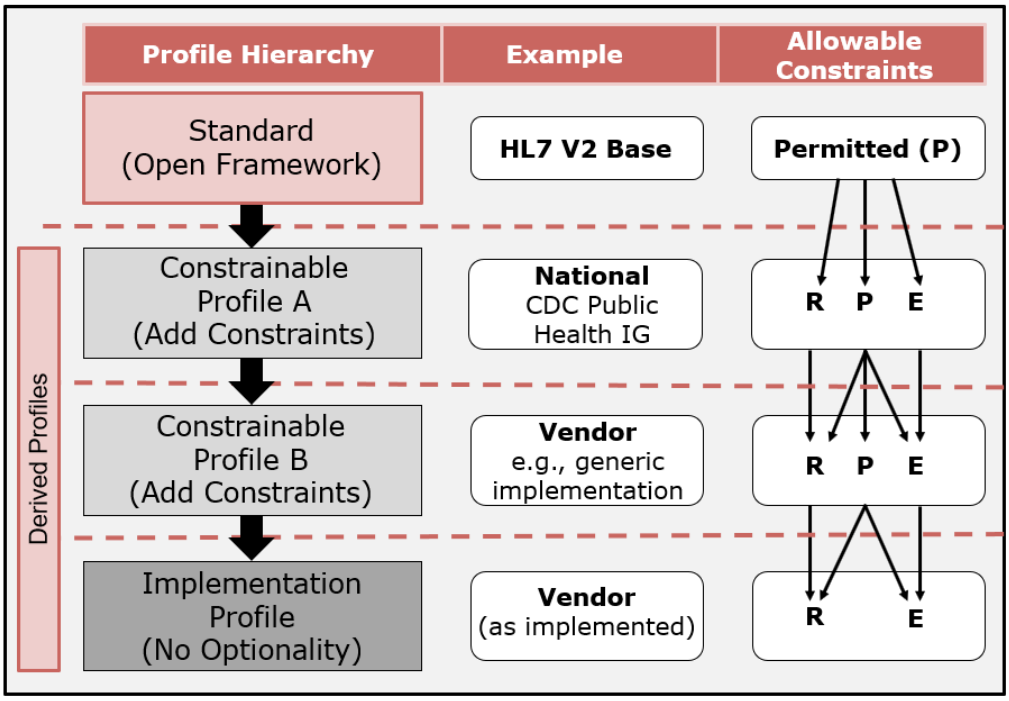

Figure 6.4: Vocabulary Profiling Usage and Allowable Transitions

Figure 6.5 presents a typical vocabulary specification for administrative gender at the base standard level. The table indicates the value (or code), a description, usage, and the code system to which each value belongs. Typically, usage is not designated at the base standard level; Permitted is the implied usage for each code.

Value	Description	Usage	Code System
A	Ambiguous	P	HL70001_v2.5.1
F	Female	P	HL70001_v2.5.1
M	Male	P	HL70001_v2.5.1
N	Not Applicable	P	HL70001_v2.5.1
O	Other	P	HL70001_v2.5.1
U	Unknown	P	HL70001_v2.5.1

Figure 6.5: Base Standard Sample Vocabulary Definition

Also, unless explicitly stated otherwise, a vocabulary specification at the base standard level has open extensibility. Extensibility is rarely restricted in a base standard (or, more precisely, there is no indication of the extensibility).

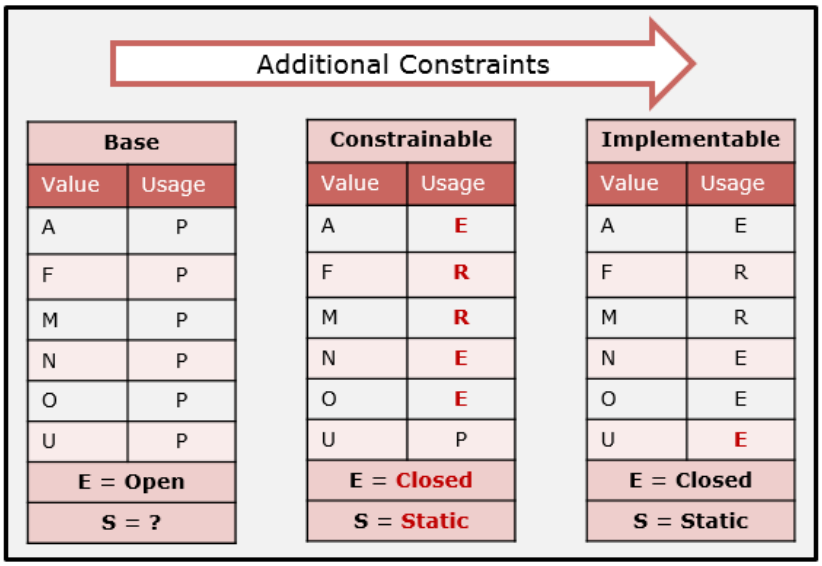

Figure 6.6: Example Vocabulary Profiling 1

Figure 6.6 shows a vocabulary profiling example using an abbreviated value set definition. Starting with the base standard definition, a set of concepts and associated codes are defined. The Usage for all of the codes is Permitted, and the value set definition has "open" extensibility. This value set definition is equivalent to the sample vocabulary definition in Figure 6.5 (but the description and code system columns are omitted for brevity in Figure 6.6). The usage of each code in the value set is determined based on the use case analysis conducted during the development of a derived specification. In the constrainable profile in Figure 6.6, the usage for codes "F" and "M" is profiled to Required; for "A", "N", and "O" the usage is profiled to Excluded; and usage for "U" is left to be decided in another derived specification (e.g., a local implementation). Furthermore, the specification authors decided that no additional codes can be added in derived profiles (Extensibility = "closed"), and that this value set is not suitable for revisions post-publication of the specification (Stability = "static"). Next, Figure 6.6 shows that, at the implementable profile level, the local use case dictated that the concept represented by the code "U" was not needed, therefore, it is profiled with a usage of "E". A different local use case could specify that the concept is needed, in which case the usage for the code "U" would be set to "R".

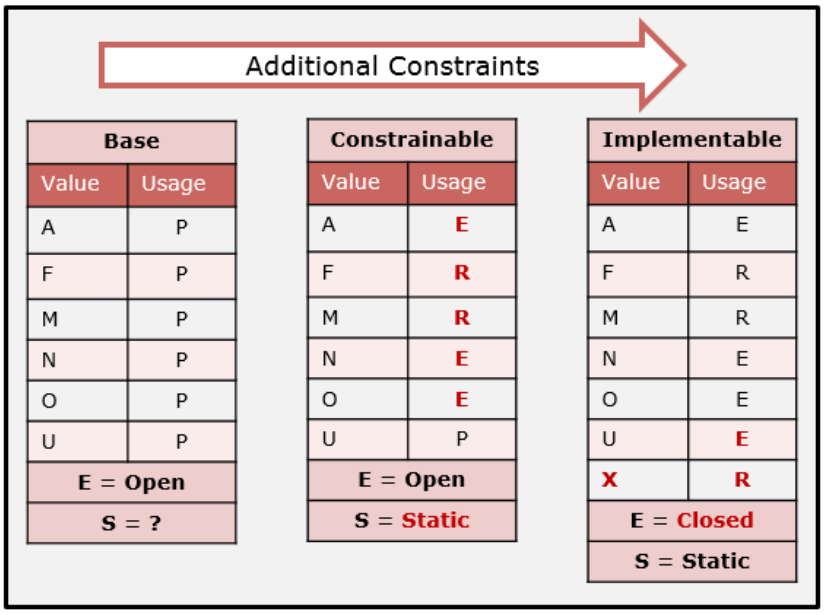

Figure 6.7: Example Vocabulary Profiling 2

In the next example (Figure 6.7), the use case for the implementable profile warranted an additional concept represented by the code "X". This extension is valid since the value set has Extensibility set to "open" in the constrainable profile. If the Extensibility had been "closed" in the constrainable profile, the code "X" could not have been added to the implementable profile (to be considered compliant with its *parent* profile).

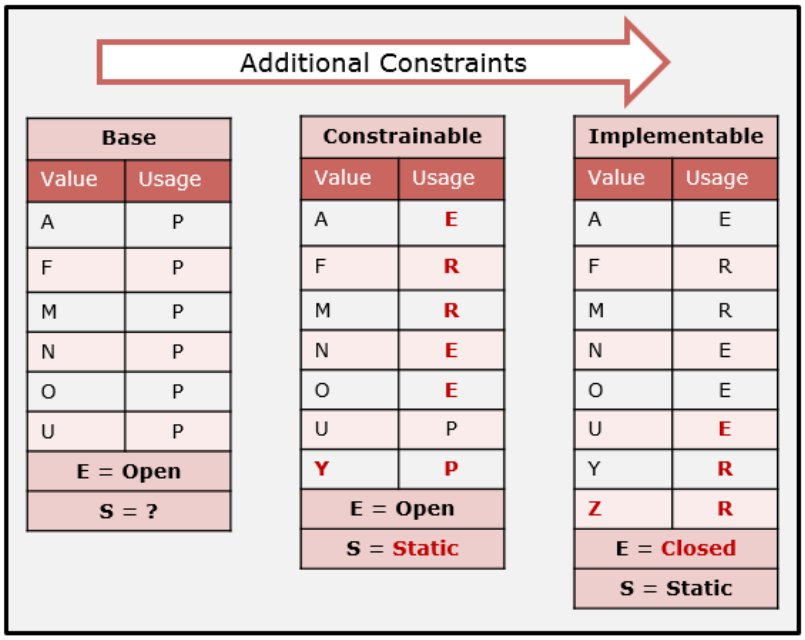

Figure 6.8: Example Vocabulary Profiling 3

The example in Figure 6.8 shows that a new concept (represented by "Y") was added in the constrainable profile, and another concept (represented by "Z") was added in the implementable profile. The base standard Extensibility is "open", which allows for the addition of code "Y" in the constrainable profile; likewise, the constrainable profile Extensibility is "open", which allows for the addition of the code "Z" in the implementable profile. It is important to note that the *extensibility* is a construct that is in the specification space; as long as the specification authors indicate that it is "open", additional codes can be added as the use case is refined. Additional codes can be added even when the Stability is "static", because Stability describes another dimension of specification, which is whether the value set can be modified post-specification by an external steward. In this example, the authors deemed this value set not to be modifiable post-specification.

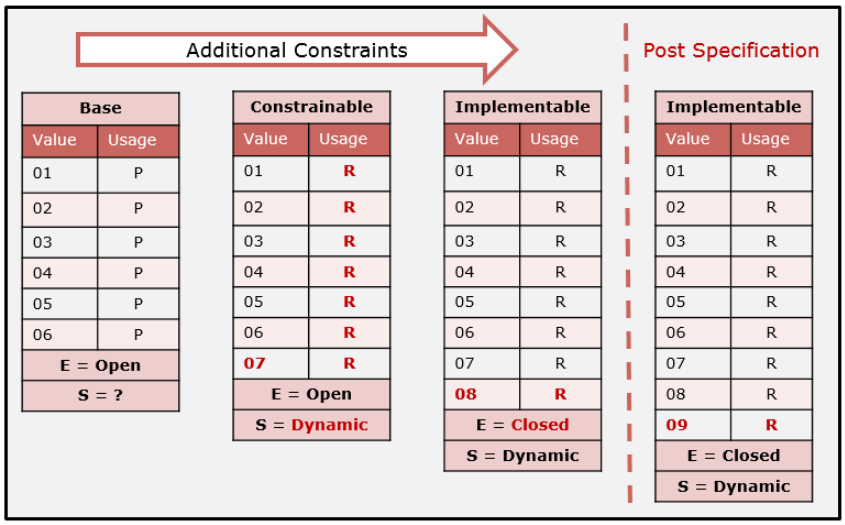

Figure 6.9: Example Vocabulary Profiling 4

Figure 6.9 demonstrates how the use of a dynamic value set might be defined in practice. Initially at the base standard level, not only is the concept domain known, but a particular code system is specified. An example might be the set of CVX codes for reporting immunizations. When published, the base standard specified codes ”01” through ”06”. It might be assumed that the binding strength and all code usages are defined as Required; however, generally no indication of this explicit requirement is given in the standard. It may also be assumed for this vocabulary definition that Extensibility would be “open”, and, for Stability either “dynamic” or “static” might be assumed (but again, typically no indication is given in the standards). Therefore, in Figure 6.9 all usages are initially shown as Permitted, Extensibility is set to ‘open’, and Stability is set to ‘?’ (unknown, but to be defined). In a constrainable profile (e.g., at the national level), it is likely that a newer version of the CVX code system exists, as years may have passed since the publication of the base standard and the development of an implementation guide, so the constrainable profile would reference the latest CVX code system. Figure 6.9 illustrates this possible change; the immunization represented by the code ”07” has been added to the code system. Stability for the constrainable profile is defined explicitly to be “dynamic”, which is an indication that post-specification the implementer can expect this code system (implicitly a value set) to change. At the implementable profile level, another update has been made to the CVX code system: code “08” has been added. This change may have occurred by the time a particular implementation was installed, as the CVX code had been updated since the publication of the national level specification. Implementers should use the latest version of the code system available. Additionally, the implementers should be aware that subsequent revisions will be made and should plan accordingly. This situation is depicted in Figure 6.9 as a post-specification revision; code “09” has been added. When the immunization message is transmitted, the *version* of the CVX code system used should be indicated (but it rarely is). HL7 v2.x supports this capability with the message profile identifier. The dynamic value set can be considered a profile component, and the sender can indicate support of a specific version of the value set.

Another situation to be aware of is the deprecation or status (e.g., made inactive) of codes in an external code system. In such a case, the usage of the code needs to be clearly indicated in the

specification. In the previous example, what if code ‘03’ is made inactive by the external steward of the code system in the post-specification space? Is it valid to continue to send the code? The answer depends on the value set specification tied to the anticipated use. If the value set is deemed to support newly administered vaccines, then the specification would indicate that inactive codes are now “excluded”, and a new value set is created. However, if the use is for the messaging of historical vaccines, then the inactive codes are still valid (and therefore “required”). Separate value sets and bindings should be included in the specification to articulate the requirements precisely.

These examples illustrate the various ways in which a value set can be specified. The vocabulary profiling mechanism provides a flexible utility such that a broad set of value sets can be specified.

An analysis should be performed for each coded element in the specification to determine which codes from a source code system are applicable for that particular element. Unfortunately, this level of definition usually does not occur in practice. Developers, therefore, must decide for themselves which codes apply to a particular element. For example, in many HL7 v2.x implementation guides, HL7 table 0203 (identifier type) is universally applied to every element that includes an identifier type with no profiling (i.e., HL70203 is simply referred to or is copied as-is from the base standard). There are over one hundred identifier types, and in most cases only a few of the codes are pertinent to any given data element. Figure 6.10 presents one possible solution for specifying the appropriate level of detail.

Concept Domain = Identifier Type	Binding ID	1	2	3	4
	Strength	R	R	R	R
	Extensibility	Closed	Open	Open	Open
Value Set Root Name = HL70203_USL	Stability	Static	Static	Static	Static
	Profile ID	LRI	LRI	LRI	LRI
	Binding	PID-3.5	PID-18.5	ORC-12.13	OBX-23.7

Value	Description	Usage	Usage	Usage	Usage
AN	Account Number	E	R	E	E
BA	Bank Account Number	E	E	E	E
DL	Drivers License Number	P	E	E	E
FI	Facility Identifier	E	E	E	R
MR	Medical Record Number	R	E	E	E
NPI	National Provider Identifier	P	E	R	P
PT	Patient External Identifier	R	E	E	E
SS	Social Security Number	P	E	E	E
XX	Organization Identifier	E	E	E	R

Location	Element Name (Identifier Type)	Value Set	Strength
PID-3.5	Patient Identifier List	HL70203_USL.1	R
PID-18-5	Patient Account Number	HL70203_USL.2	R
ORC-12.13	Ordering Provider	HL70203_USL.3	R
OBX-23.7	Performing Organization Name	HL70203_USL.4	R

Figure 6.10: Creating a Collection of Value Sets and Binding to a Data Elements

The Identifier Type in HL7 v2.x is bound to a number of data types and is used in higher-level elements. In a given specification, many data elements will use the Identifier Type for many different purposes. The base standard provides a set of codes for common identifier types that apply to a broad spectrum of identifiers. This set can range from a medical record number to a bank account number. For a particular element, however, only a few of the codes might be applicable.

At the base standard level all codes in a code system can be implicitly considered Permitted, thus giving the most flexibility for specification in derived profiles. An analysis must be performed for each element that uses the code system in order to ascertain which codes actually do apply. Figure 6.10 shows a sample list of codes for HL7 table 0203. It also indicates four elements to which this code system is bound in the base standard. For each binding, a value set is created that is shown in the form of a separate column. For the Patient Identifier List-Identifier Type (PID-3.5) element, analysis of the particular use case is necessary (e.g., analysis related to the Lab Results Interface (LRI) determined that support for the Medical Record Number (MR) and Patient External Identifier (PT) codes are required). The Driver's License Number (DL),

National Provider Identifier (NPI), and Social Security Number (SS) codes are Permitted, while the rest of the codes are Excluded[45]. For the Patient Account Number-Identifier Type (PID-18.5) only the Account Number code is applicable. Value sets for Ordering Provider-Identifier Type (ORC-12.13) and Performing Organization Name-Identifier Type (OBX-23.7) are defined as well. In addition to specifying the usage for each code, other attributes for each element-specific value set are determined, such as the Binding Strength, Extensibility, and Stability. Essentially, the collection of value sets in Figure 6.10 are domains for the concept of identifier type, and elements can be constrained by indicating the domain.

The table in the lower part of Figure 6.10 shows examples of value set bindings that might appear in a specification (each row in the table would be in the associated element location in the specification). The upper part of the diagram shows the value set collection. Each column represents a value set and lists the associated attributes and vocabulary-constraint usage for each code in the value set. Some of the attributes appear both in the specification and the value set collection, which allows for easy cross-referencing (since they can be specified in separate documents).

The Binding ID is a qualifier to the Value Set Root Name (HL70203_USL) and, when combined, they provide the binding in the specification (e.g., HL70203_USL.3 is used to bind to the Identifier Type of the Ordering Provider element). The binding qualifier may be included as part of the binding definition in the specification (as shown in Figure 6.10) or it may not be included. If the qualifier is not included in the specification (profile), then the profile and the value set collection are separated, which means the value sets could be changed without requiring changes to the specification. This approach promotes effective management of updates; however, versioning must be accounted for and stakeholders need to be notified.

Although these value sets are shown to be bound to particular elements for a particular profile, value sets are defined independently in order to promote reuse of the value set. As an example, perhaps several value sets could be created from the administrative gender code system, and then the different value sets could be referenced in many specifications (not only HL7 v2.x but any other specification based on other standards, such as CDA and FHIR). This approach to value-set reuse requires broader vocabulary management.

6.5.1 VOCABULARY COMPATIBILITY

Value sets can be created for senders and receivers independently. Since this is the case, under what circumstances are the value sets compatible? Figure 6.11 shows how a sender and receiver might profile a value set differently in a constrained profile. The sender excludes both D and H, while the receiver excludes D but includes H. In this case, the derived profiles (and therefore implementations) are compatible.

[45] For the complete HL70203 table many more codes would be listed, most of which would be “Excluded” for these four data elements.

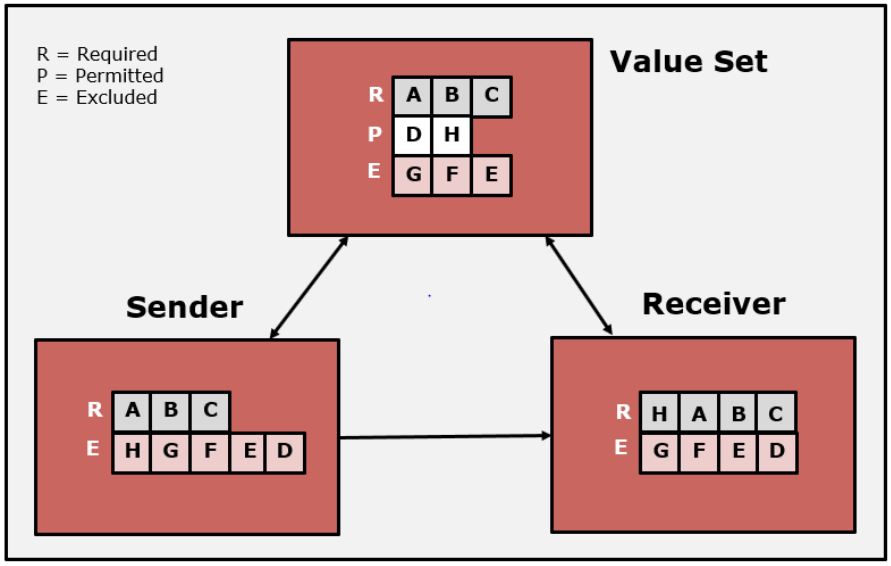

Figure 6.11: Terminology Assessment for Sender/Receiver Implementations

In general, if the sender is only sending a subset of the codes that are supported by the receiving application (Figure 6.11), then the profiles are compatible. If the sender uses a code the receiver does not support, however (see element "D" in Figure 6.12), a compatibility issue occurs.

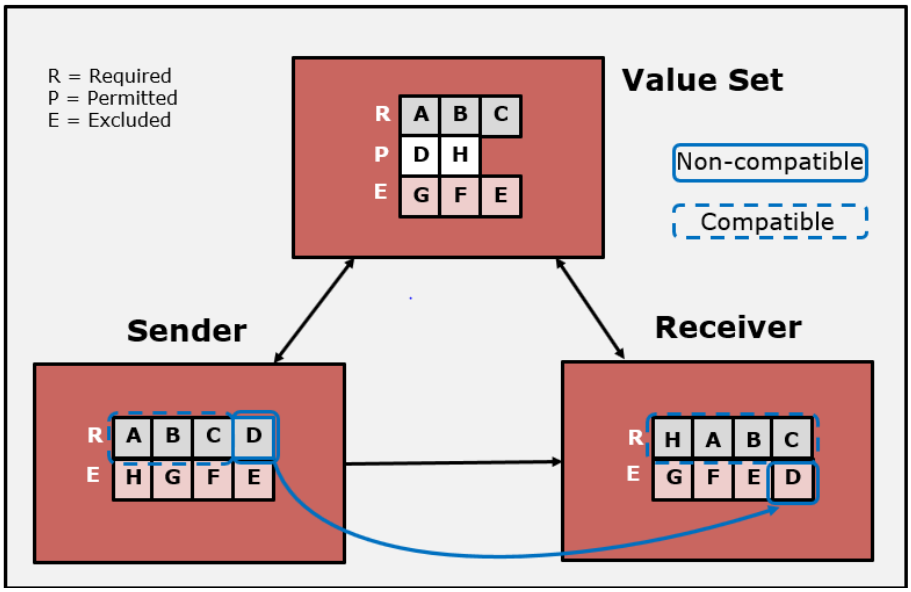

Figure 6.12: Compatibility Issues with Supportive Set of Codes

According to the implementation guide, both derived profiles are compliant, because they specify valid constraints. The sender and receiver are not compatible, however, because the sender specifies code "D" as R-required while the receiver has specified code "D" as E-excluded. Thus, even if both sender and receiver implement their specified profiles correctly (that is, they are conformant), they are not interoperable (because the specifications are not

compatible). This example stresses the importance of trading-partner agreements to address specific use case needs. In this case, the sender has an expectation about a code that is not supported by the receiver. Table 6.4 depicts the situation in a row and column format.

Table 6.4: Compatibility Analysis for Vocabulary

Sender	Relationship	Receiver	Compatible	Comment
S := { c | c is supported by sender}	⊆	R := { c | c is supported by receiver}	Yes	The receiver has more supportive capabilities than the sender.
	⊇		No	The sender can send a code that the receiver does not understand.

6.6 Profiling HL7 Tables

The base HL7 standard defines four types of tables:

- User-defined
- HL7-defined[46]
- External
  - Referenced (maintained by another standards organization)
  - Imported (Imported in the base HL7 v2 standard)
- Local

Certain rules that are defined for these tables in the base standard must be adhered to in the constraint of the table. These rules are the starting point for placing constraints as described in the previous sections on vocabulary profiling.

User-Defined Table: A user-defined table provides an initial table identifier and, in some cases, suggested code values. These are recommendations and, therefore, place no initial requirements on the vocabulary definition. In a message profile, a specifier may use, extend, modify, or replace the suggested codes and descriptions. Once the vocabulary definition is defined, a specifier may bind the vocabulary definition to a data element and designate the binding strength as "required", as prescribed in Section 6.1. That is, the specifier considers the data semantics associated with the data element, determines a suitable vocabulary definition, and declares that definition required[47]. The status of the binding is required at this point, even though the underlying data type is 'IS'[48]. A specifier can promote the data type to 'ID', however, the implications of the initial data type are of little consequences once the vocabulary requirements are specified in a message profile.

HL7-Defined Table: An HL7-defined table provides an initial table identifier, code values, and certain base standard requirements. Code values defined in an HL7 table must not be redefined in

[46] The term HL7 table is overloaded. HL7 tables are those types defined in the standard and include User, HL7, External, and Local. An HL7 defined table is a certain type of a table defined in the base standard.

[47] The specifier could also bind the vocabulary definition to an element with a binding strength of suggested. However, this is not typical since once a use case is known, specific values are also generally known.

[48] Also applicable to CWE.

a message profile; and if a concept needed for a use case is not defined, the table definition can be extended. Below is a list of rules for constraining HL7-defined tables:

- for a concept that is needed in a message profile and is contained in the HL7 table, the message profile must use that code for that concept (i.e., the code can't be replaced or redefined)
- for a concept that is needed in the profile and not contained in the HL7 table, the message profile can add a code to the vocabulary definition for that concept
- for a concept that is not needed in a message profile but is contained in the HL7 table, the message profile should exclude the code from the vocabulary definition
- for a concept that may be needed in a derived profile and is contained in the HL7 table, the message profile should designate the code as permitted in the vocabulary definition (a determination of inclusion or exclusion is made in a derived profile)

Beyond the initial set of rules imposed by HL7-defined tables in the base standard, constraints are applied as described earlier in this section.

External Tables: An external table is a vocabulary definition created, maintained, and published by another standards organization. A specifier may reference an external table (code system) and bind that table to a data element in a message profile. The base standard may bind a code system or a set of code systems[49] to a data element. In cases of a CNE data type, this binding is the starting point for profiling.

Local Tables: The HL7 base standard provides no requirements or guidance for local tables. A local table is a table with a non-HL7 assigned table identifier that contains a set of locally- or site-defined values. Local tables are typically used for elements with a CWE data type. During the process of profiling, a specifier will create and bind local tables. If the specifier designates the binding strength as "required", the status of the binding is required at this point, even though the underlying data type is 'CWE'. The mechanisms described for specifying vocabulary requirements in this section must be used for defining local tables.

Table 6.5: Implied Vocabulary Binding Parameters for Base Data Type

Replaced	Bind. Str.	Ext.	Stability	Comments
IS	Suggested	Open	Static	Completely open for specifier to create and constrain.
ID	Required	Open	Static	Must use HL7 codes and can be extended.
CE	Suggested	Open	Dynamic	Completely open for specifier to create and constrain.
CF	Suggested	Open	Dynamic	Completely open for specifier to create and constrain.
CWE	Suggested	Open	Dynamic	Completely open for specifier to create and constrain.
CNE	Required	Closed	TBD	Must use bound vocabulary as-is; may be dynamic.

[49] Depending on context (such as realm) a particular code system is designated.

Table 6.5 provides a summary of the initial requirements the HL7 v2 base standard places on vocabulary bindings. These requirements are the starting point for message profiles. In most cases, the responsibility of defining and specifying vocabulary requirements falls on the message-profile specifier. In other cases, an initial set of requirements are given that can be further defined and constrained.

6.7 Null Flavors

A null flavor (or null value) is a concept for representing "unknown" content. HL7 v2 does not explicitly provide a mechanism for null flavors. However, an equivalent outcome can be achieved for coded data elements by including one or more null flavor coded values in a value set along with the values necessary to satisfy the use case. Any element that is bound to that value set containing a null flavor can use that value. The determination of employing the null flavor mechanism is use case-dependent. Some HL7 tables include null flavors as part of the code set in the base standard. Additionally, HL7 Table 0353 (CWE Statuses) contains a set of null flavors that can be used in the creation of value sets with a need for null flavor coded values.

6.8 Relationship of Coded Element Based Data Types and Flavors

The base standard provides a set of coded element data types for conveying coded data in message elements. This initial set covers only a small portion of specification possibilities. Using the concept of data type flavors and the vocabulary profiling mechanisms together provides a complete set of specifications. For example, the less restrictive "CWE" in the base standard can be constrained in many ways to precisely state the messaging and vocabulary requirements at a certain profile level. The data type definition determines the requirements for the presence of a code. Vocabulary profiling (binding and vocabulary definition) indicates the requirements for the code set and also for application of further constraints. The base standard co-mingles these dimensions; the Conformance Methodology seeks to delineate the constructs in such a way that specifications and requirements can be clearly understood by implementers.

7 PROFILE CONSTRUCTION

7.1 Profile Design and Management

A message profile is normally thought of as a complete message-structure definition with additional constraints applied to it as a "whole". However, in some circumstances it is convenient and efficient to employ a modular approach to profile construction. A profile component defines a part or a certain aspect of a message definition and is used to aggregate correlated requirements and/or to differentiate requirements from another profile. It provides a mechanism to support a set of reusable requirements. A profile component can be applied to any construct or section of a message definition. A core profile is used to document the common set of requirements across the set of related profiles. A composite profile is the composition of a profile or core profile and one or more profile components. In the end, a composite profile is a profile with the distinction that the profile was created by combining a profile or core profile and one or more profile components. Profiles and profile components can be combined to develop and manage other profiles. A profile component in a family of profiles can be used to identify different levels of requirements for the same use case or to identify the differences in requirements for different, but closely related, use cases. Table 7.1 summarizes the profile building blocks.

Table 7.1: Profile Building Blocks

Concept	Definition
Profile	A message definition and a set of constraints applied to the message definition that addresses all interface requirements for the use case.
Core Profile	A message definition and a set of constraints applied to the message definition that addresses partial interface requirements for the use case(s). A core profile documents a set of common (base) requirements and is intended (typically) to be used in more than one composite profile.
Profile Component	A set of individual independent constraints linked to elements in a message definition. A profile component is used to document differential requirements and is a subset of a message definition.
Composite Profile	The composition of the profile or a core profile and one or more profile components. A composite profile is a profile but differs in the manner in which it was constructed.

7.1.1 PROFILE COMPONENTS

A profile component defines a part of or a certain aspect of a message definition and is used to differentiate requirements from another profile. A profile component can be applied to any construct or section of a profile. A profile component in a family of profiles can be used to identify different levels of requirements for the same use case or to identify the differences in requirements for different, but closely related, use cases.

Entries in a profile component will identify the element location and constraint value. Entries can apply to any element in the message definition and any constraint can be applied. Examples include specifying the usage for an element or defining a value set and binding that value set to an element.

In one case, a specification may need to express different levels of conformance. For example, a profile in the specification may be written to require the use of Object Identifiers (OIDs) for all Universal ID data elements. Another profile may be written in which use of OIDs for these data elements is not a requirement (i.e., other identifier types are allowed to be used). An intermediate profile may be written that requires certain, but not all, of these data elements to support the use of OIDs. This specification is, in principle, describing three levels (as mutually exclusive sets) of conformance requirements. These three profile levels can be described using a core profile definition and three profile components. The profile components describe the differences in the requirements (this approach can be thought of as a substitution mechanism).

In another case, a profile component may need to be employed to express requirements for a different, but closely related, use case. Here the creators of the new profile component leverage the requirements in an existing profile, since that existing profile contains many common requirements.

In the first case above, the use case is the same, but the requirements in which it can be achieved are different. The profile component is expressing a different level of conformance. In the second case above, the use case is similar, but there are several important differences, and, therefore, the requirements are different. The profile component concept is used to leverage the in-common requirements defined by the profile while allowing any different (additional) requirements to be defined in a profile component.

Profile components can be used as "*building blocks*" to specify a complete profile (set of requirements) as identifiable sub-units. As such, they can express common requirements, additional requirements, or substitute requirements. Profile components are an efficient utility to manage and define a family (a related set) of profiles.

The descriptions of the different conformance levels, profiles, and profile components should be contained in the conformance clause section of a specification. Based on the information provided in these descriptions, an implementer is able to make a conformance claim as to which profiles they support, and if applicable, the level of conformance they support.

7.1.2 COMPOSITE PROFILES

A composite profile is a complete profile consisting of a profile or a core profile and one or more profile components. A composite profile is synonymous with a message profile but differs in that it comprises a set of sub-parts that represents the whole.

7.1.3 PROFILE CONSTRUCTION EXAMPLES

This section presents some examples of how profiles can be constructed using the profile building blocks. The examples provide a sampling of uses. When writing a set of related profiles (or a family of profiles), it is important to reuse the profile, core profile, and profile components in order to harmonize the requirements and to gain efficiency. The concepts of the profile levels and profile components provide an effective approach for managing and documenting extensions.

Figure 7.1 through Figure 7.5 illustrate possible configurations for composing a family of related profiles. One design principle is to develop a core profile that applies across a family of profiles with the intention of using the profile component concept to specify *complete* composite profiles.

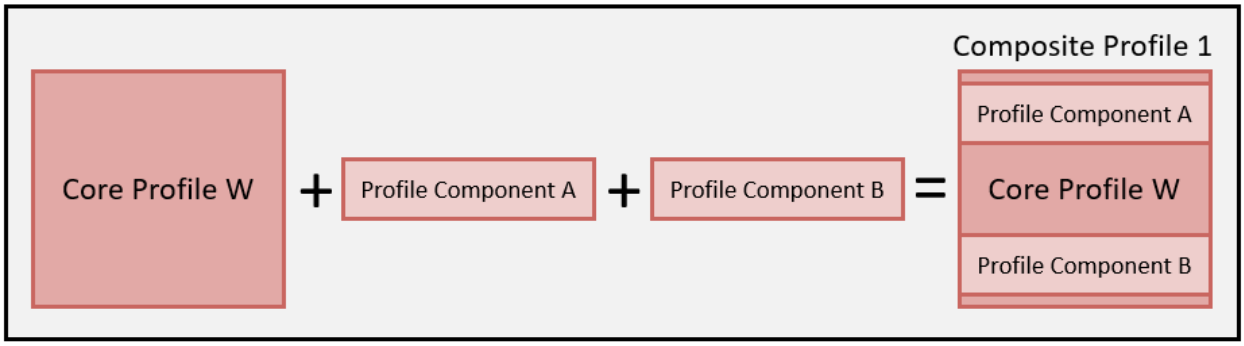

Figure 7.1: Profile Design Principles – Example 1

In the first depiction (Figure 7.1), a core profile is developed that expresses all of the common requirements[50] for a related set of profiles. Profile component A and profile component B are also created for aspects that are not defined (constrained) in the core profile. Combined, the three definitions are used to describe a complete specification, Profile 1 (composite profile).

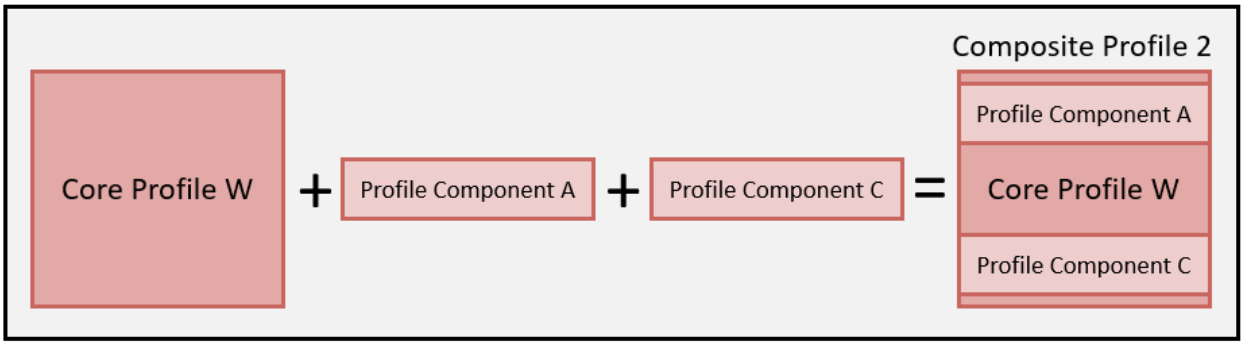

Figure 7.2: Profile Design Principles – Example 2

For the second depiction (Figure 7.2), the core profile and profile component A are reused and combined with profile component C to specify Composite Profile 2.

In some use cases, a specifier may want to create profiles that differ slightly for the sender and the receiver profile. Profile components can be used to specify these sender and receiver profiles. Figure 7.3 shows an example in which a core profile is created along with a separate sender profile component and receiver profile component. A composite profile for each, the sender and receiver, is the outcome.

In the fourth depiction (Figure 7.4), Profile Y is combined with profile component D to create Composite Profile 3. Example 4 demonstrates how use case expansions can be managed. This may be the case where a complete profile for a given use case is defined, and another use case exists that extends the requirements. An example is laboratory reporting where a profile is

[50] That can be, and often is, an incomplete set of requirements for a particular use case.

created to submit laboratory results to an EHR-S and then extended to a second profile to submit reportable laboratory results to public health. The requirements for submitting to public health are specified in a profile component. Relating this to example 4, Profile Y is the laboratory reporting profile, profile component D documents the additional requirements for public health reporting, and Composite Profile 3 is the complete public health specification for reportable laboratory results. Another example is the case of national and local requirements. A typical domain for which localization is needed is public health, where a national-level profile is created and additional requirements (constraints) are specified at the state- or jurisdictional-level. A constrainable profile for the national level is created for a particular interaction (e.g., send immunization record). State-level requirements could be expressed in a profile component. When combined with the national-level profile, the result expresses the complete requirements for the state.

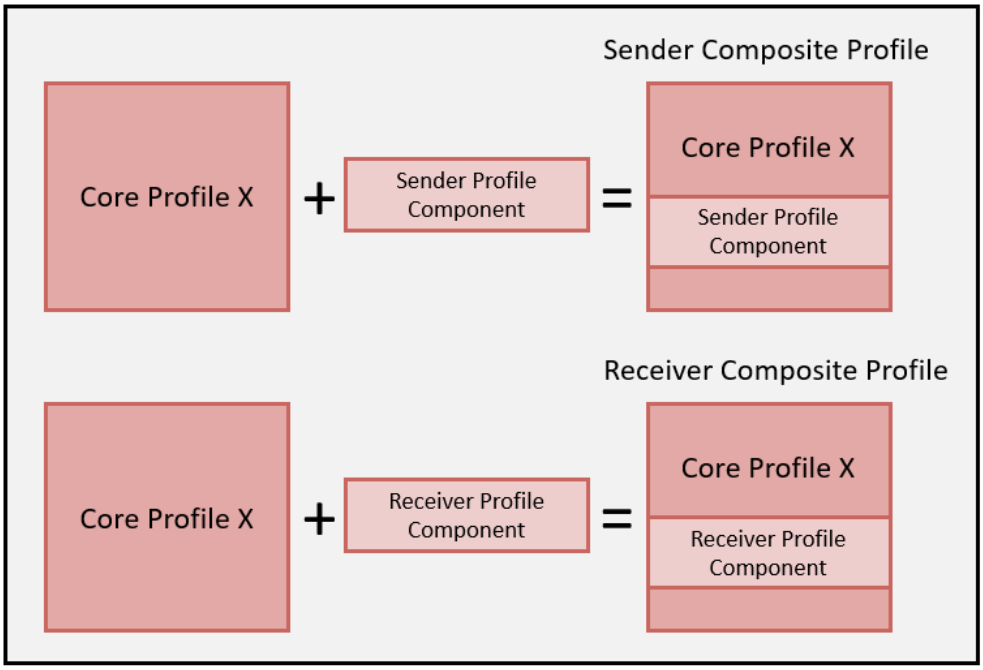

Figure 7.3: Profile Component – Example 3

In this situation, using profile components provides an efficient mechanism to reuse and repurpose in order to accommodate a different, but closely related, use case. Profile components also help reduce implementation efforts by clearly indicating the essential differences.

Profile components also can express new requirements that replace requirements established in a profile or core profile. This approach often is used when different levels of profiles are developed, or when the profile provides utility outside the original intent of the profile. The fifth depiction (Figure 7.5) illustrates a case where a subset of requirements for an existing profile are overridden. Here, Profile Z is used; however, certain aspects are redefined according to the constraint rules and are documented in profile components E and F, which results in Composite Profile 4. It is important to note that if the new profile is intended to be a refinement in the existing profile hierarchy, then the requirement replacement is limited to further constraints (in essence, this is a level). However, if the intent is to establish a new specification to address a

similar (but different) use case, then there is no restriction on the requirement replacement, since it is a new profile (i.e., it is not intended to be a specialization of the original). Here, the use of profile components is a mechanism to leverage an existing specification.

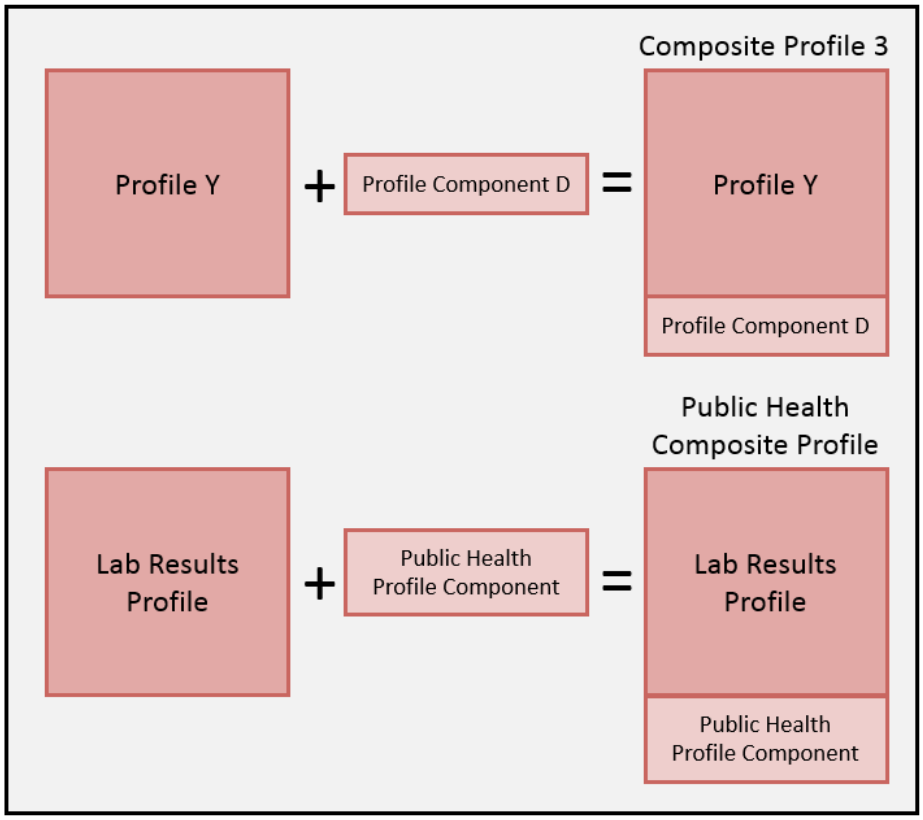

Figure 7.4: Profile Component – Example 4

For each of the complete specifications illustrated in Figure 7.1 through Figure 7.5, the resulting profile can be a constrainable or an implementable profile.

The key design principles are that when related specifications are being developed, the authors should leverage as much information as possible from existing profiles, or they should design/create core profiles that are a harmonization of requirements for a related set of use cases. The profile components can be developed at any level of granularity; however, caution should be exercised when creating profile components at a fine-grained level or when specifying numerous details. Often, having to manage many building block artifacts can outweigh the benefits these artifacts are supposed to provide. If management tooling is available, then fine granularity of profile components is attainable. A good practice is to introduce an orthogonal structure of the individual requirements (e.g., data type constraints) in one case, message fragments (e.g., for insurance or diagnosis data) in a second case, and value set definitions in another instance, which would allow for easy integration and combinations (i.e., a data type specialization should not include a specific value set binding, as it significantly reduces the ability to reuse).

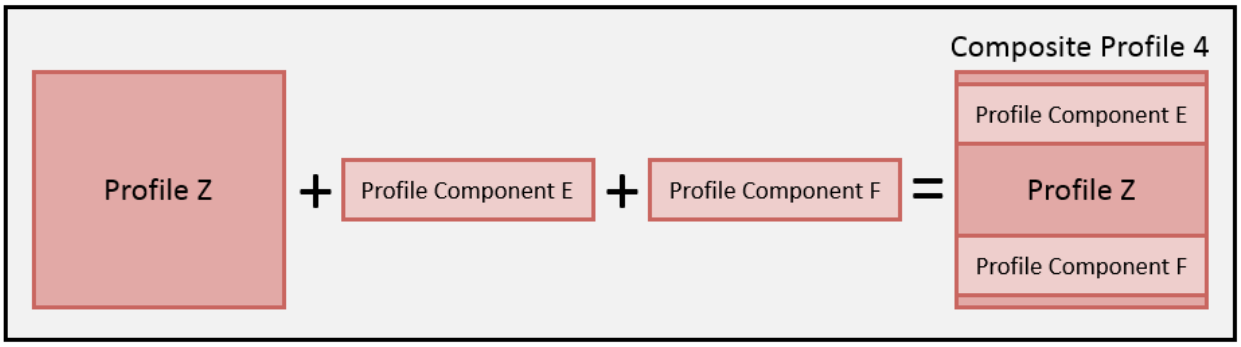

Figure 7.5: Profile Design Principles – Example 5

Frequently, standards developers fully specify each of a related set of profiles, entailing duplication of sizeable sections of the standards document. These profiles typically are not harmonized, which unnecessarily leads to inconsistences and maintenance issues. Furthermore, though it often occurs in practice, it is not a good idea to combine requirements targeted for different use cases (interactions) into a single profile definition. For each interaction, a separate profile needs to be defined, and the use of profile components, as described within, facilitates this approach.

The methodologies described are ideal for managing and creating customized interface products. A purchaser (e.g., a hospital) may want to know the capabilities of an interface in order to assess its suitability for a particular need. In most cases, the vendor provides an interface that supports many features, most of which typically are driven by market demand. The system is designed to be configurable so as to support a variety of specific interface needs. The use and documentation of profiles is a powerful mechanism to manage system configurations. In essence, each installation of an interface is an implementable profile whether it is documented explicitly or not. All of these aspects can be described exactly in the form of profiles as well. The vendor might publish what could be called a “configurable implementable profile”, which declares the implementation capabilities and allows a prospective purchaser to compare the profile to their needs. Once an interface has been installed, the capabilities are clearly defined and configured as the implementable profile, and, ideally, this profile is documented.

7.2 Selective Adoption

Many versions of the HL7 v2 base standard exist. Each version contains message definitions that are constrained by implementers for a given use case. The process of placing constraints on message definitions is called profiling. Typically, constraints are placed on a message definition defined entirely in a given version of the standard. In some cases, however, specifiers and implementers may want to use as a foundation a given version of the standard (e.g., 2.5.1) while selecting certain features offered by another version of the standard (e.g., 2.8.2). In essence, all HL7 v2 base standard versions can be viewed as a single collection of objects from which specifiers and implementers can pick and choose to construct message profiles. Implementation guides can contain message definitions from different versions of the standard and also could contain message definitions that contain objects from more than one version of the standard. The selective adoption only applies to the base version from which the message profile originates. Selective adoption does not apply to implementation guides. Aspects that need to be included from implementation guides can be included by using the profile component mechanism.

This section describes:

- how to construct a message profile using objects from different versions of the standard
- the different methods for selective adoption
- the type and granularity of the objects that can be adopted

The HL7 v2 message profile is the governing artifact for specifying requirements for a message interface. Regardless of the base standard version designation, the profile modifies the standard-level definition to satisfy use case requirements. Such modifications can selectively include objects from different versions of the standard. Therefore, implementers must interpret the requirements based on the message profile (indicated in MSH-21) and not merely rely on the message version (indicated in MSH-12). This premise is true regardless of whether the message profile contains objects from a version other than the foundational version. The message profile declares a foundational version and, for every adopted object, the HL7 v2 version of that object.

In the context of selective adoption, the following terms are defined to help explain the process:

Object – an object from the HL7 v2 standard; can be any element (group, segment, field), data type, or table.

Adopted Object – an object from another version of the standard that is imported into a message profile.

Indigenous Object – an object from the foundational version of the message profile. Indigenous objects do not have an explicit version attribute. The version of the object is the version of the Foundational Version.

Foundational Version – this is the version of the base standard that serves as the core (starting) version of the message definition. All objects in the message profile will comply to this version unless explicitly overridden by an adopted object.

Adopted Version – this is the HL7 v2 version from which an object was adopted.

7.2.1 RATIONALE FOR SELECTIVE ADOPTION

HL7 v2 has a very large installed base of implementations that are built on earlier versions of the standard. As use cases and needs evolved, so did the standard. For a given use case, some, but not all, features of the newer standard are needed or wanted. In such circumstances, specifiers and implementers may want to add only the features needed (and not be affected by other features that are not needed). Therefore, instead of requiring implementers to upgrade to a newer version of the standard, selective adoption allows them to upgrade only the parts needed for their use. For example, if a domain or a large installed base of interfaces already exists in version 2.5.1 of the standard, and if in an update to an implementation guide a few fields from version 2.8.2 are desired, then only those fields from version 2.8.2 need be included in the message profile definition. This approach likely will have the least impact on implementers. In any project, however, implementers must consider the pros and cons of each approach.

7.2.2 METHODS FOR SELECTIVE ADOPTION

Three methods by which objects can be adopted into a message profile are available:

1. Selective Pre-adoption
2. Selective Post-adoption
3. Selective Post-adoption from the current HL7 v2+ edition

The details of each method are given in the sections to follow.

7.2.2.1 SELECTIVE PRE-ADOPTION

The selective pre-adoption method allows a profile definition to be created using a version of the standard while allowing adoption of objects from a later (newer) version (or versions) of the standard. This method is suitable when a specifier or implementer wishes to keep an implementation guide or implementation at the current version *level* while taking advantage of features from later (newer) versions of the standard. This approach has minimal impact on installed implementations.

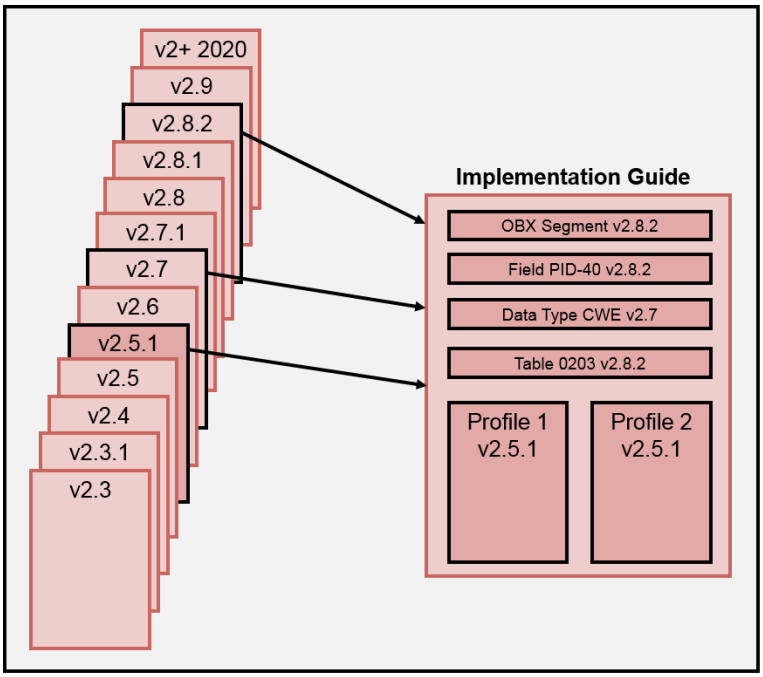

Figure 7.6: Selective Pre-adoption Example

Figure 7.6 shows an example of Selective Pre-adoption. An implementation guide is produced in which the foundation profiles are created with version 2.5.1 message definitions. The figure also shows that a segment, field, and table are pre-adopted from version 2.8.2 and a data type is pre-adopted from version 2.7. The pre-adopted objects are used to replace or extend existing objects in the foundational profile (message) definition. Note, the figure does not show the relationship of the pre-adopted objects in the message profiles, because, depending on how the implementation guide is constructed, this relationship can vary.

Table 7.2: Selective Pre-adoption Examples

Object	Example
Table	One example of a table pre-adoption comes from the laboratory results interface (LRI) implementation guide in which the foundational version of the message profile is 2.5.1, but an updated table definition from version 2.8.2 is sought. Table HL70078 (abnormal flags) in version 2.5.1 contains codes L (below low normal), LL (below lower panic limits), H (above high normal), and HH (above upper panic limits). For table HL70078 (name

	changed to “interpretation codes”) in version 2.8.2, another level of granularity was added to the code set: LU (very low) and HU (very high), which are levels between L and LL, and, H and HH, respectively. Additionally, the definitions of L, LL, H, and HH were refined. The LRI implementation guide pre-adopted the 2.8.2 version of the table, and this table replaced the 2.5.1 version of the table.
Field	An example of field pre-adoption comes from the laboratory orders (LOI) implementation guide in which OBX-26 through OBX-29 are pre-adopted from 2.8.1. These fields are not present in the foundational version is 2.5.1. LOI needed only OBX-29 but needed to include OBX-26 to OBX-28 in order to maintain field order. Those fields were kept as optional in the implementation guide.
Segment	An alternative for the prior example of field adoption is a segment adoption in which the entire 2.8.1 segment is pre-adopted (which would include all updates to those fields if any). This is in contrast to the field adoption in which the first 25 fields are defined as version 2.5.1 and the last 4 are defined as version 2.8.1.

7.2.2.2 SELECTIVE POST-ADOPTION

The selective post-adoption method allows a profile definition to be created using a version of the standard while allowing adoption of objects from an earlier (older) version (or versions) of the standard. This method is suitable when a specifier or implementer wishes to take advantage of the capabilities of a newer version of the standard but doesn’t want to use all *upgrades*. There may be aspects of the newer standard that the community (or established base) is not ready for, or the cost of such changes is prohibitive, or the earlier functionality suits the use case better. For example, it may be the case where a specifier wishes to use a later version of the standard for certain capabilities but wishes to keep current capabilities for a large installed base. However, if these capabilities have been deprecated (i.e., element changed to B or W) in the newer version of the standard, the specifier can "resurrect" the capabilities using post-adoption.

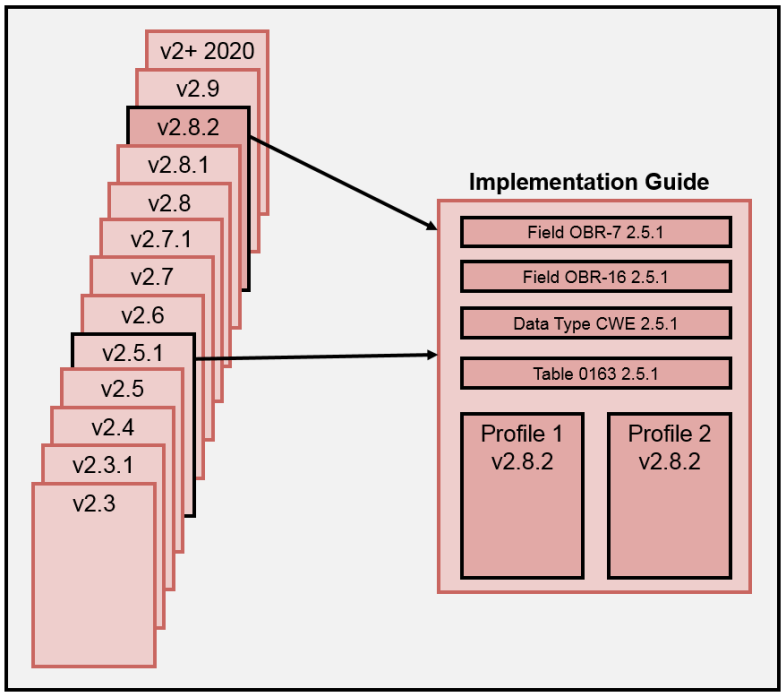

Figure 7.7: Selective Post-adoption Example

Figure 7.7 shows an example of Selective Post-adoption. An implementation guide is produced in which the foundation profiles are created with version 2.8.2 message definitions. The figure also shows that two fields, a data type, and table are all post-adopted from version 2.5.1. The post-adopted objects are used to *roll back* capabilities to previous definitions of concepts. Note, the figure does not show the relationship of the post-adopted objects in the message profiles, because, depending on how the implementation guide is constructed, this relationship can vary.

7.2.2.3 SELECTIVE POST-ADOPTION FROM CURRENT HL7 V2+ EDITION

The Selective Post-adoption from the current HL7 v2+ edition is fundamentally the same as the previously explained selective post-adoption. The key difference is that specifiers are limited to using the current v2+ edition as the foundational version.

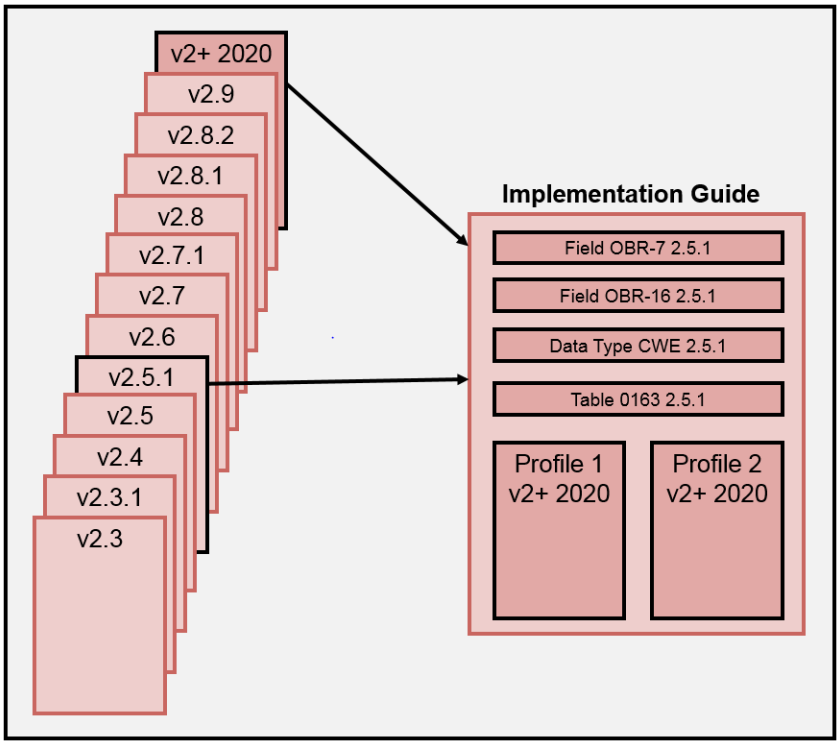

Figure 7.8: Selective Post-adoption from the current HL7 v2+ edition

7.2.3 OBJECT ADOPTION LIMITATIONS AND RULES

The type and granularity of the objects (and their attributes) that can be adopted are limited. Below is a list of candidate objects:

Table 7.3: Selective Adoption Candidate Objects

Object	Comments
Abstract Message Definition	Abstract message definitions (AMD) should not be modified to include additional segment definitions. If a desired new segment is needed, then the latest version of the standard should be used that includes that segment. Post-adoption can be used for aspects desired from a previous version of the standard. Changing the AMD invalidates MSH-9. If a new AMD is needed, then that AMD should be created in a new version of the standard.
Segment	The segment would carry along with it the fields and all associated attributes (e.g., vocabulary bindings, tables, usage) from the adopted version. The segment would carry along with it the data types and all associated attributes.

Field	Fields that are added must maintain the order integrity of the fields in the adopted segments even though all fields may not be added. For example, if the foundational version of the segment has 11 fields, and the specifier wants to add fields 14 and 15 from another version, then fields 12 and 13 must "come along" in the adoption process. If there are more fields, say 16 and 17, then these fields need not come along. The fields would carry along with them the data types and all associated attributes (e.g., vocabulary bindings, tables, usage) from the adopted version. Therefore, the segment would have a mix of objects from various versions. Specifiers that are pre-adopting fields should be aware that some fields may have dependencies on other fields; this factor needs to be considered and addressed in an appropriate manner, which may be to include the dependent fields.
Data Type	The data type can only be wholly adopted. Complex data types must contain data types of the same version (e.g., an XAD data type with a version of 2.5.1 must have components, both primitive and complex of version 2.5.1). Components can't be added to data types (all or partial). Reason: Data types are fundamental building blocks and should not be implemented piecemeal. Data types are analogous to data types in programming languages. A data type is a "cohesive" concept. A Field references a pre-adopted data type (wholly) of the same root or a with compatible data type. Rules for data type substitution apply equally to data type adoption. For example, CWE version 2.7.1 for ID version 2.5.1 is permissible.
Tables	Tables (code sets) must be adopted wholly. Modifications can be made through profiling to create value sets. No single codes can be adopted; the addition or removal of a single code can be accomplished via profiling and creating a value set to which codes are added or are removed via exclusion. That is, in essence, individual codes can be "adopted" via value set creation mechanism.

8 PAIRING SENDER AND RECEIVER PROFILE FOR USE

Profiles document a set of requirements (or capabilities) for systems. A profile is applicable to a sender, a receiver, or to both if a common expectation is sought. An interaction profile pair associates a sending profile and a receiving profile, for example, for an ADT message. A profile pair at the transaction level is for the initiator and the responder, for example, an ADT message and an ACK message. The focus of this section is on the interaction profile pair.

Sender and receiver profiles can be paired in various ways to satisfy a targeted use. The profile pair binding can have various patterns including:

- One-to-one
- One-to-many
- Many-to-one

Regardless of the profile pairing pattern, a set of expectations is specified in a *higher-level* constrainable profile for each sender and receiver in the use case. In practice, the expectations can vary substantially. For example, in one case the sender and receiver may have mutual expectations about how the data are processed, and in another case the sender may be agnostic about how the data are processed. Any combination of the profile pairing patterns and processing expectations is valid.

It is important to note again that the use case defines these expectations, because it describes how the sender and receiver are interpreting requirements for the same message. In the subsections that follow, a representative set of profile pairings are presented along with general expectations of the sender and receiver in the context of a given use case; however, the details about a specific use case are not considered.

8.1 One-to-one Profile Pairing

A common profile pairing is an exchange between a sender and receiver in which there are mutual expectations. In this case, the sender and the receiver share the same (or nearly the same) profile and, therefore, implement a common set of requirements. An example is the US Realm Laboratory Results Interface (LRI), where the sender has an expectation that the receiver will process and use the data in a prescribed way. From a regulatory perspective in the U.S., the Clinical Laboratory Improvement Amendments (CLIA) places the responsibility on the sender (i.e., the Laboratory) for ensuring that the laboratory results are correctly consumed, processed, and displayed by the receiving system (e.g., an EHR). As such, nearly identical constrainable profiles are specified to meet the requirements of the use case.

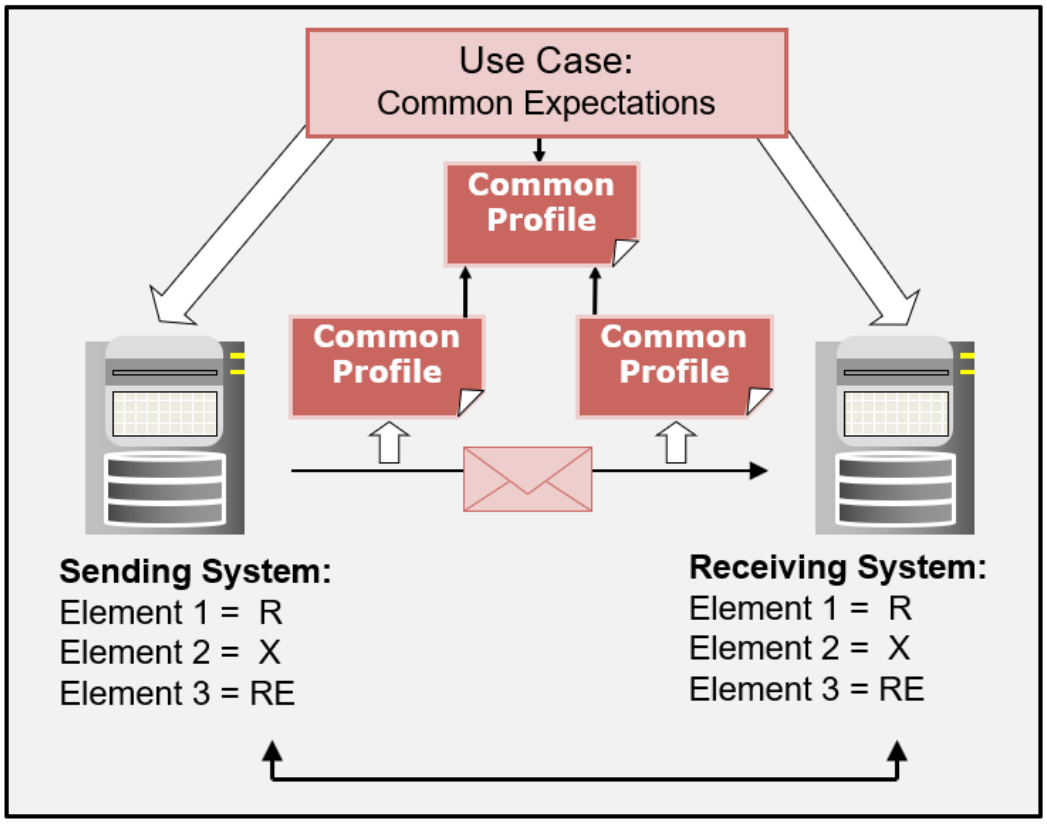

Figure 8.1: One-to-one Profile Pairing Pattern (Mutual Expectations)

Figure 8.1 illustrates a common profile being used, which signifies mutual expectations for the data exchange. This use case documents how the profile pair is to be utilized.

In another situation, the profile pairing may exhibit the one-to-one pattern, but it may not have strongly-correlated expectations about how the exchanged data are handled. Again, these expectations are described in the *higher-level* use case.

Note that in this example, and the examples to follow, an examination of the requirements and their compatibility is limited to the usage conformance construct in order to simplify the explanation of the concept. Comparable analysis applies to the other constraints. In Figure 8.1, the common expectation for the sender and receiver is indicated by the same usage settings.

8.2 One-to-many Profile Pairing

The one-to-many profile-pairing pattern typically is used for broadcast applications in which there is loose correlation of sender and receiver expectations. The sender has no or limited expectations about how the receiver processes and uses the data. The sender is providing a *service* for the receivers. It is the responsibility of a receiver to ensure that the sender is providing the information necessary to complete the particular use case. An example is the ADT (Admissions, Discharge, and Transfers) use case. Typically, ADT systems will broadcast a patient's information to a number of other systems (which may be internal or external to the sending entity). In such a case, the sender will provide as much information as possible about the patient. This information can be documented in a profile and implemented by the sender. The sender is providing an indication of the data it is able to give. The requirements for the sender

often are derived from the collective set of receiver requirements, which is typically fluid, as receiver requirements can change and/or other receivers can be added to the network. Each receiver can provide a profile to indicate what it needs. In essence, the sender profile is a *superset* of the receiver requirements.

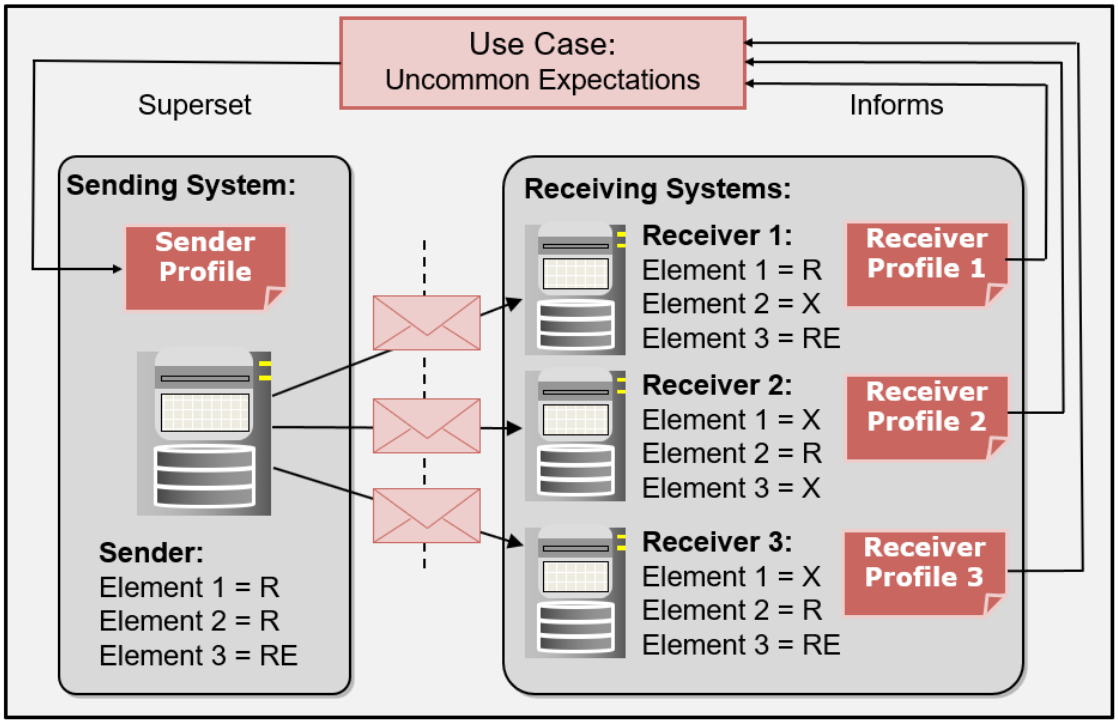

Figure 8.2: One-to-many Profile Pairing Pattern (Receiver-side Expectations)

Figure 8.2 illustrates a single sending system along with three receiving systems and excerpts from their profiles. As shown, the sending system profile provides support for all the elements needed by all of the receiving systems. For example, for Receiver 1, the sending system sends Elements 1 and 2, and if known, Element 3. This set of elements satisfies the needs of Receiver 1, because this receiver requires Element 1 (which is supported by the Sender), does not need Element 2 (and will discard it), and does support and will process Element 3 if it is provided by the sender. The *higher-level* use case indicates the expectations of the sender and receiver. Each receiver has its own use case that informs what data are to be received from the sender (and, hence, defines the sender profile). The sender has no expectation about how the data are processed by the receiver—this processing requirement is specified in the *higher-level* use case. The profiles in this pair are compatible with each other, because the receivers are provided with the data they are requesting.

Figure 8.2 shows that the pairing of a sender usage of R (or RE) and a receiver usage of X is allowed (and is compatible). Compatibility is a concept that is considered from the receiver's perspective. A simple question used to assess compatibility is: will the receiver be provided the information it needs? In this example, and unlike the example of mutual expectations, the sender has limited expectations or responsibility for the treatment of the data that are sent. The need for the data is driven by the business requirements of the receiver. The receiver takes the information it needs to fulfill its use case and ignores (processes and discards) unwanted data. If the sender

had expectations about the processing of one or more data elements, then these expectations are documented in the *higher-level* constrainable profile, and both sending and receiving profiles would specify a usage of *required* for each element for which there is a common expectation.

8.3 Many-to-one Profile Pairing

The many-to-one profile-pairing pattern is typically for use cases in which there is a single collection point for data. Figure 8.3 illustrates a case where multiple senders exchange information with a single receiver. This situation is common in the public health arena where, for example, multiple providers send data to an Immunization Information System (IIS).

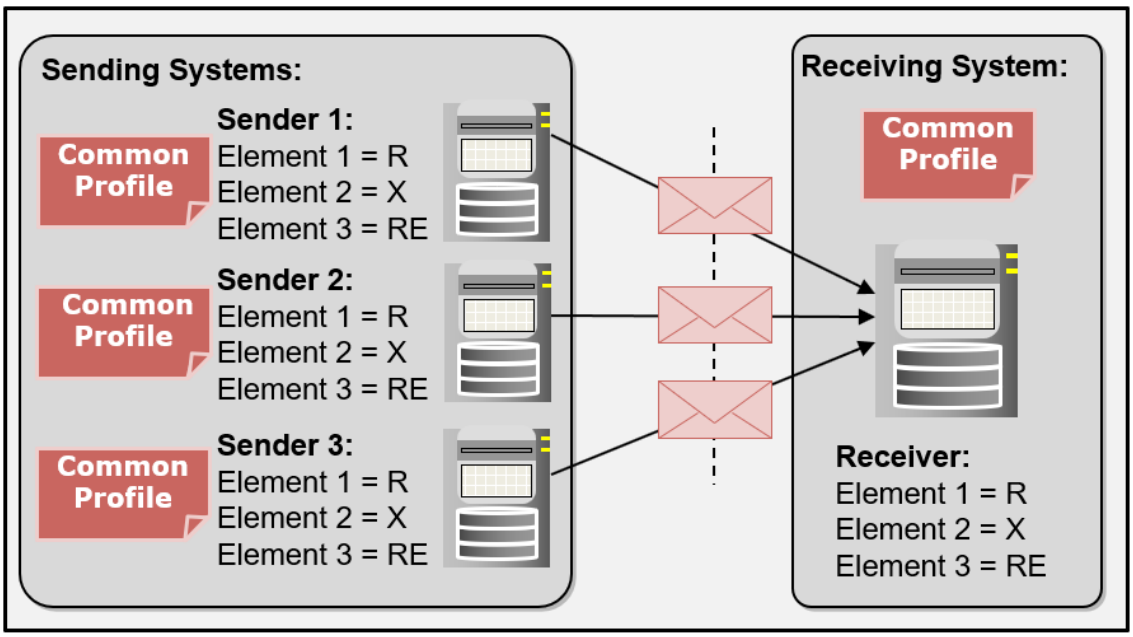

Figure 8.3: Many-to-one Profile Pairing Pattern (Example 1)

For this example, the use case defines a common set of messaging requirements (as indicated by the common usage settings in Figure 8.3). As with the previous examples, these requirements, along with the processing expectation, are defined in the common *higher-level* constrainable profile (not shown in this diagram). The processing expectations can be mutual (more or less), sender-side oriented, or receiver-side oriented. In the case of the U.S. national guidance for immunization, the processing expectations are mutual (more or less) for the sender and the receiver. This use case is explored below and is related to the many-to-one profile-pairing pattern.

In immunization systems, two basic types of information are collected: patient-identifying information and vaccination events. The IIS is not the source of truth for the patient-identifying information, so the patient-identifying information is generally loosely correlated in the exchange. The submitter provides patient-identifying information to allow patient matching and consolidation of immunization histories, but the submitter is agnostic about how the IIS processes or records patient-identifying information. However, the IIS is tasked with storing and creating a complete vaccination history, so the sender expects the IIS to accept and store all

submitted vaccinations[51], which must be available for retrieval later. In this regard, the vaccination requirements between the sender and receiver are mutual (strongly correlated).

As shown in Figure 8.3, the data collected by the receiver are provided by many sources, and, therefore, the information about a particular patient may not match the information of any individual sender (provider). For example, an IIS may have information about a patient from both a doctor's office and a pharmacy. Some data (such as patient demographics) may be submitted by both sources while other data (such as childhood immunization information from the doctor's office and a recent influenza immunization from a pharmacy) may be unique to one system or the other. The *high-level* use case would address this situation. An individual sender would expect the IIS to handle the information they submitted, but also would expect it to handle additional and modified data.

It is important to note that expectations set for particular data elements will vary according to each use case. In some situations, the expectation may be that the received data are processed and stored, and in other situations the expectation may be that the received data are made available for performing a function after which they will be discarded. For example, patient demographic data are sent as part of the immunization scenario, but there is limited or no expectation that these data will replace the data that exist in the IIS. An individual provider usually is not the source of these data. In this case, the data are needed by the receiver to obtain a match to an existing patient. Once the matching function is performed, the data may be discarded.

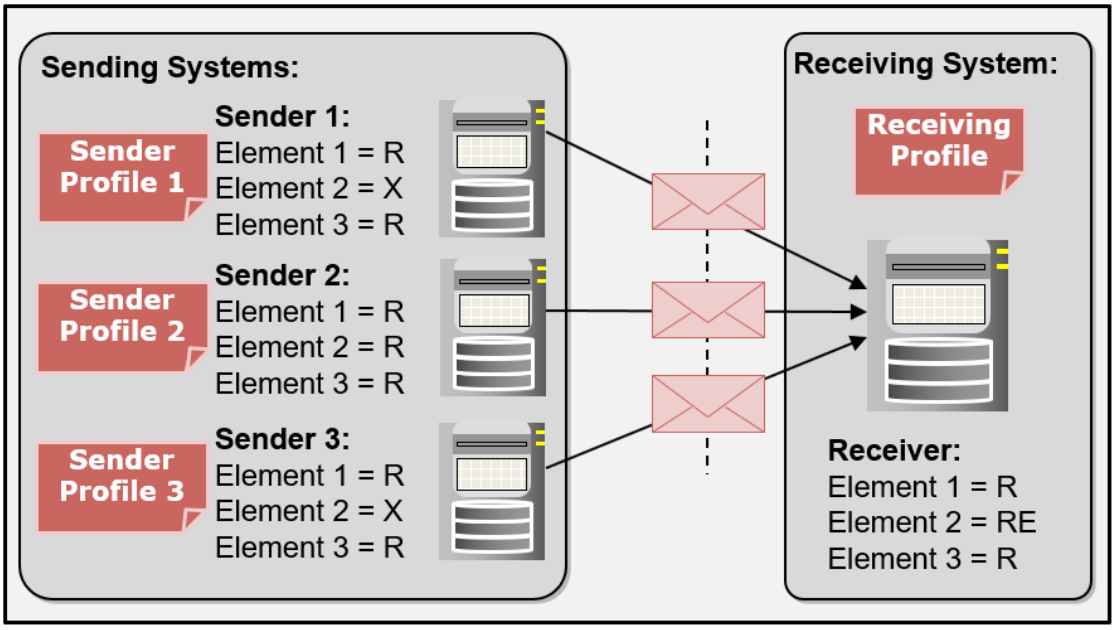

Figure 8.4: Many-to-one Profile Pairing Pattern (Example 2)

[51] In this case, the sender's expectation is in a "collective" sense. For a particular instance, data may be processed and discarded because the receiver recognizes that the data are duplicates, and the existing data are of better *quality*. A typical case is where the receiver has data from the provider who administered the vaccine and the duplicate data newly received are a historical recollection provided by the patient.

A slightly different circumstance is where not all senders have the capabilities desired by the receiver. As shown in Figure 8.4, Sender 1 and Sender 3 are not capable of supporting a certain requirement. For example, a particular EHR-S may not be capable of reporting refused vaccines; however, other systems in the network support this capability, and the information is useful to the IIS. In this case, the receiver publishes a profile that includes the *superset* of sender capabilities (or, more appropriately, the receiver's *wish list*), but does not require support of all capabilities desired. In the example presented in Figure 8.4, this situation is indicated by the designation of "RE" usage for Element 2. If the data are provided, the receiver can process the information, but the receiver is not dependent on the data to operate. To reiterate, it is the use case that sets the expectations and describes the relationship between the sending and receiving profiles.

8.4 Design Considerations: Profiling Pairing

A design question to consider is: should all profile pairings adhere to the one-to-one pattern? In this paradigm, the sender and receiver share the same (or nearly the same) profile and, thus, expectation about the handling of the data from the sender and receiver perspective is documented in the profile. Take, for example, the use case where the sender is broadcasting to a set of receivers. The sending system documents, in the form of a profile, the superset of requirements that accommodates all receivers. The sending system extracts information from its data model, maps the data to the superset profile, and sends a superset message. When additional requirements are needed in the receiver set, additional provisions are made. The sender unilaterally updates its profile and then broadcasts updated messages[52]. The receivers, based on an original agreement, are prepared to accept unexpected data and deal with them appropriately. Such fluid expectations have advantages in terms of efficiency, operational tolerance, and expansion of capabilities with little or no negotiations in operating interfaces. Alternatively, the sender could tailor messages based on each receiver's profile. Upon data extraction from the data model, the sender maps the data and creates a message specific to a particular receiver based on the negotiated profile. This tight association binds the sending and receiving set of requirements together. There is no ambiguity between what the sender is providing and what the receiver is expecting. The receiver always gets what it expects and nothing more. The robustness of the interface sets a clear expectation and reduces the chance of misconceptions and, therefore, error in interpreting and using the data. The downside of this approach is the increased effort involved in each sender/receiver negotiation and interface implementation. The practicality of this approach also must be considered: is it feasible or optimal to implement it into today's environments? In some circumstances there are clear benefits, in others, maybe not; but achieving the goal of tight associations may reduce gaps in the interoperability bridge. Implementers need to examine the use case and weigh the trade-offs relative to cost and effectiveness.

[52] This is an underlying basic principle of many data exchange standards such as HL7 v2.

9 PROFILE DOCUMENTATION

One of the most important steps for successful interoperability is thorough documentation of the implemented interfaces. Unfortunately, most vendors do not follow this practice. When vendors do document the implemented interface, they often include nothing more than an extract of the original standard. Profiles can be, and should be, used to facilitate the comprehensive documentation of interfaces.

If an interface fails, an accurate understanding of the expected system behavior is essential for resolving the problem, which is why the interface documentation a vendor provides must include more information than just a reiteration of the base standard. Purchasers should seek complete and correct interface documentation. One obstacle to gaining vendors' cooperation in providing these specifics, however, is the fact that when a vendor "customizes" an interface, the hidden configuration details that they may consider proprietary must be made transparent and available to possible competitors as well as the customer.

9.1 Profile and Implementation Relationships

Figure 9.1 shows the relationships between profiles and implementations and the associations that can be drawn among them. Profiles at the various levels provide a source of the documentation about what is to be implemented or what has been implemented. Purchasers can use profiles to express their requirements. Likewise, vendors can use profiles to convey system capabilities. Such documentation can be used to assess needs and capabilities. If the documentation is provided in a standardized computable format, then efficiencies can be gained in assessing compatibilities. Comparisons also can be made between vendor interface implementations for a given use. In Figure 9.1, implementable profile (E) and implementable profile (F) can be assessed for profile compatibility (shown as point 6). Profile E and F are documentation of requirements for what is to be implemented or what has been implemented.

These profiles are derived from the constrainable profile (B), shown as points 2 and 4. These implementable profiles constrain the national (or realm/hospital) profile that is typically specified in an implementation guide. Profiles (E) and (F) are said to be compliant with profile (B) if the rules for adding constraints are followed faithfully. Likewise, profile (B) is said to be compliant with profile (A) if the rules for adding constraints are followed faithfully. In this case profile (B) is constraining the base standard (A). The base standard can be considered a constrainable profile.

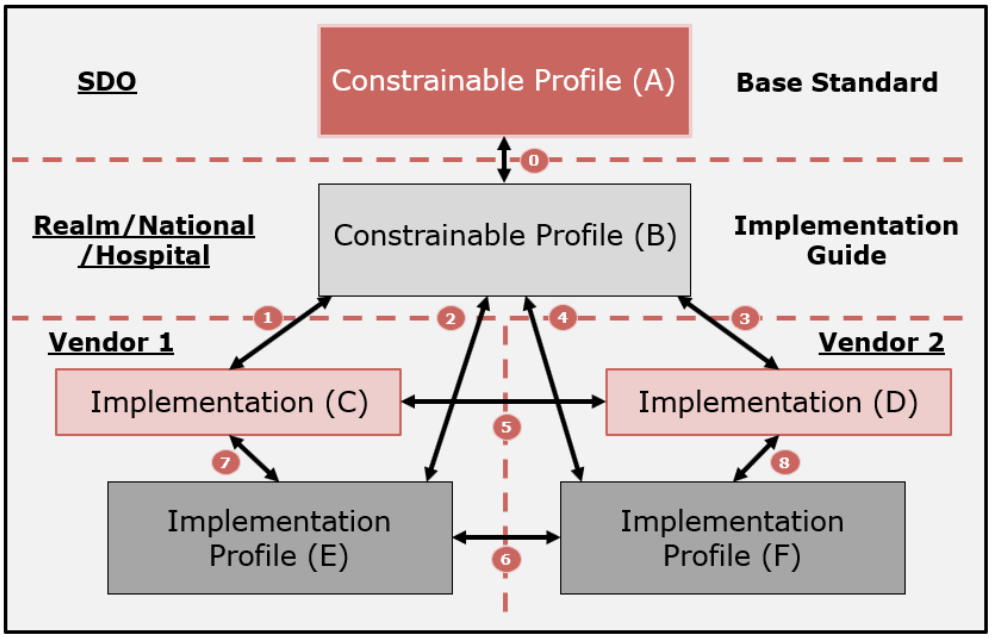

Figure 9.1: Profile and Implementation Relationships

Implementations (C) and (D) are conformant to profiles (E), (F), and (B) if the software implements the requirements as stated in the specification (shown as points 1, 3, 7, and 8 in Figure 9.1)[53].

Finally, implementations (C) and (D) are said to be interoperable if they can exchange information and use the exchanged information as intended (shown as point 5).

Compliance and compatibility are terms that are used to indicate relationships between profiles (that is, its documentation). Conformance is a term that is used to indicate the relationship between a profile and an implementation. Interoperability is a term that is used when discussing the relationship between implementations. Table 9.1 summarizes these relationships and the various assessments that can be made.

Table 9.1: Assessment of Profile and Implementation Relationships

Test Type	Dimension	Artifact	Description
Profile Compliance (Points 0,2,4)	Hierarchical	Profile	Profiles are tested against each other to determine whether one is a constraint of (i.e., consistent with) the other. Profile compliance testing is appropriate when additional constraints are specified to successive profiles in the hierarchy (e.g., standard to a constrainable profile to an implementable profile).

[53] A constrainable profile is not typically an artifact that is considered to be implementable (points 1 and 3). It is included in this diagram to show that, in some cases, implementations are developed only to the requirements specified in a constrainable profile and not to optional (or undefined) aspects. In essence, the constrainable profile is an implementable profile, although not explicitly documented as such; and therefore, in this regard, the implementation can be said to be conformant to the constrainable profile.

Test Type	Dimension	Artifact	Description
Implementation Conformance (Points 1,3,7,8)	Hierarchical	Implementation	Provides an assessment of how well the application fulfills the requirements specified in a profile. This is conformance testing. ONC Health IT Certification is an example of conformance testing.
Profile Compatibility (Point 6)	Peer	Profile	Profiles are tested against each other to determine whether the pair can be used by applications to successfully exchange information (interoperate). If a profile pair that constrains the same underlying profile conflict with each other, chances of interoperability for applications that implement these profiles are diminished.
Implementation Interoperability (Point 5)	Peer	Implementation	Applications are tested with each other to determine whether they can successfully exchange information (interoperate). Applications that implement the same profile or compatible profiles and have successfully passed conformance tests have increased likelihood of interoperating. IHE connect-a-thons are an example of interoperability testing.

The ultimate goal in development of an interface is to ensure that two implementations of that interface are interoperable with each other. The final step before deployment of an interface is to test the interface. In the simplest case, this test is performed "live"; that is, two systems are directly tested with each other, as in the IHE Connect-a-thon for example. For a hospital, performing a live test is often difficult to accomplish because of their implementation-specific requirements, undocumented requirements, or documented requirements that deviate from the standard. At minimum, however, one would like to know whether a system can exchange data with another system or not. Documentation is the key.

Claims such as "The system can support/speaks HL7" that are made by vendors are not informative or helpful to potential purchasers, nor is a statement like "the system is compliant to the guidance" useful, because each system participating in the interface may have been designed with different interface requirements. As an example, usage for an element could be set to "Required" in one of the systems. This usage setting is not a significant issue as long as this system is not acting as the receiver of information and the partner sending-systems are not transmitting this information; however, this scenario could result in a problematic mismatch between a sender and receiver if the sender transmits a message without required information and the receiver subsequently cannot process the message.

To mitigate the downside of not being able to perform live testing of two systems (Figure 9.1 and Table 9.1, Point 5), an alternative must be sought (Figure 9.1 and Table 9.1, Point 6). Table 9.1

gives possible avenues for making this assessment as long as sufficient documentation is provided.

9.2 Documentation Quality

A standard provides the foundation for implementers and includes many options, which can lead to multiple interpretations and implementations that exhibit different behaviors depending on the options chosen by the implementers. Given the possible variations in implementation behavior, it is essential that vendors' claims for conformance to the standard are backed by documentation that clearly describes the capabilities supported. This documentation can be articulated in varying degrees of completeness and quality. To emphasize the importance of documentation, we describe characteristics of documentation quality. Some aspects are hierarchical in nature, forming levels of quality. Other aspects are necessary to ensure completeness. Table 9.2 presents documentation quality levels.

Table 9.2: Documentation Quality Hierarchy

Documentation Claim	Description
Undocumented Unsubstantiated Claim	A developer of an implementation claims conformance to a given standard; however, the claim is unsubstantiated.
Documented Unsubstantiated Claim	A developer of an implementation provides evidence of a claim with documentation of the interface. The documentation can be in any format (e.g., a text document) and the contents of the claim are not substantiated. Note: For example, the provider of the documentation (e.g., a vendor) may copy paragraphs from the original standard; this approach represents the type of documentation at this level.
Documented Standard Unsubstantiated Claim	The documentation fulfills the requirements of the conformance profiling mechanism provided by the underlying standard. A text document of a state-level (e.g., guide for Immunization for Texas) profile is an example.
Documented Standard Machine Processable Unsubstantiated Claim	The documentation is machine process-able, such as HL7 v2.x XML message profile definition (See Section 10). Tools can aid in the development of machine process-able documentation. Documentation at this level enables automated comparison of specifications and implementations.
Documented Standard (Implementable Profile Level) Machine Process-able Unsubstantiated Claim	The documentation is a message profile fulfilling the criteria for implementable profiles in a machine process-able format. Such documentation defines precisely the capabilities of the implementation. Tooling, as mentioned previously, can provide the machine process-able documentation and also allows for verification that this claim is an implementable profile.

Documentation Claim	Description
Substantiated Claim	The implementation is verified to a claim (i.e., a claim in this list) made in the documentation. The verification is performed in a testing or certification program.

It is important to note that this quality hierarchy is designed for assessment of documentation and not for systems. The primary goal is to support assessment of the compatibility capabilities of proposed interface implementations. This determination of the quality of the associated documentation provides a first-level review prior to interoperability testing with implementations.

Table 9.2 represents a tiered structure of the steps for determining the quality of documentation related to vendors' conformance claims. Substantiating a claim is most meaningful when applied to an implementable profile.

10 COMPUTABLE DOCUMENT DEFINITIONS

10.1 Document Definitions Layout

The structure of the implementation guide, message profile, vocabulary definitions, and their descendants is expressed in a computable format using the XML schema language. Specific instances of each document type are expressed in XML. The document definitions are independent but have relationship among them. Additionally, ancillary documents are defined separately to facilitate document management and processing. The XML schema definitions can be accessed at https://v2.hl7.org/conformance/profileschemas (forthcoming).

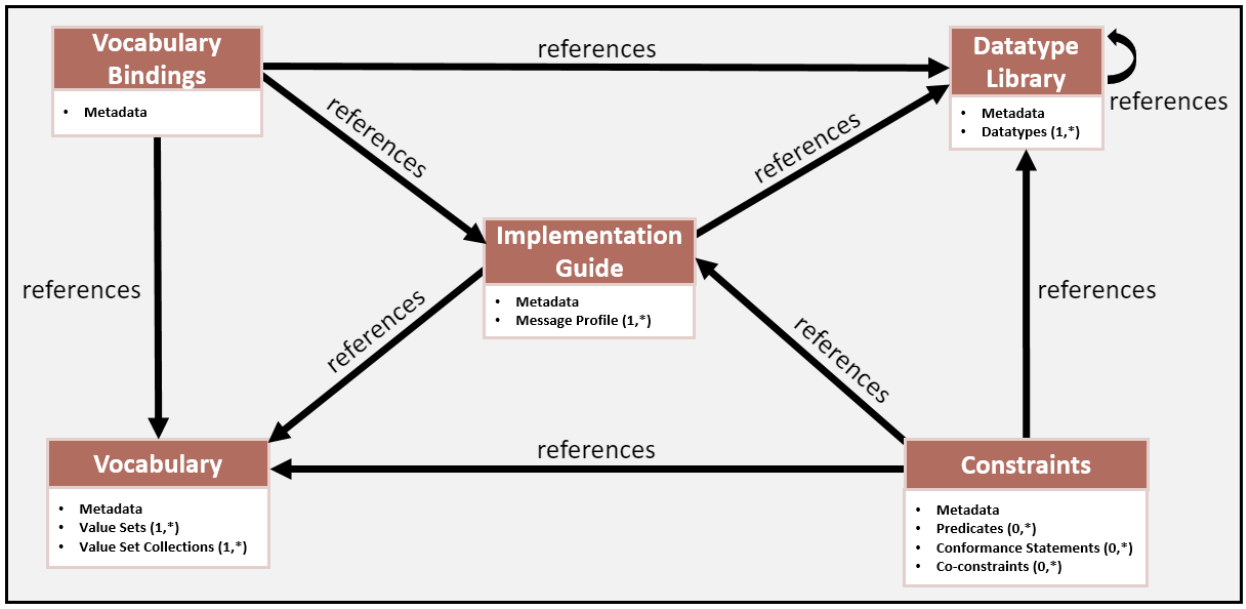

Figure 10.1: Implementation Guide Document Definition

Figure 10.1 provides a high-level overview of the implementation guide document definition. The definition contains meta data and one or more message profiles. The implementation guide document definition contains references to other document definitions, which allows reuse of the fundamental building block components such as data type definitions.

A message profile is normatively expressed as an XML instance document that contains the HL7 v2 message structure along with message element requirements. The XML instance document is governed by the message profile schema (XSD) document that provides the rules for expressing the requirements given in the normative clauses in this document.

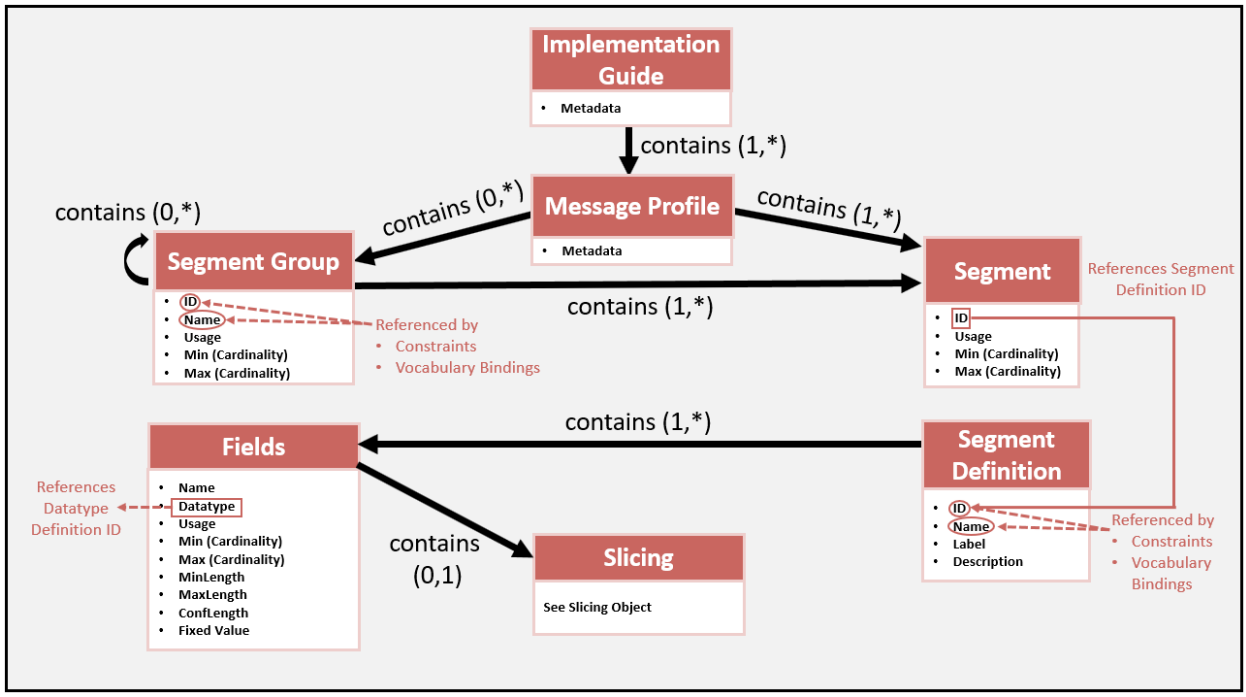

Figure 10.2: Message Profile Document Definition

Figure 10.2 illustrates the components of a message profile in the context of an implementation guide document definition. The message profiles contain segment groups and segments. The segment references segments definitions, which in turn contain the fields along with constraint attributes.

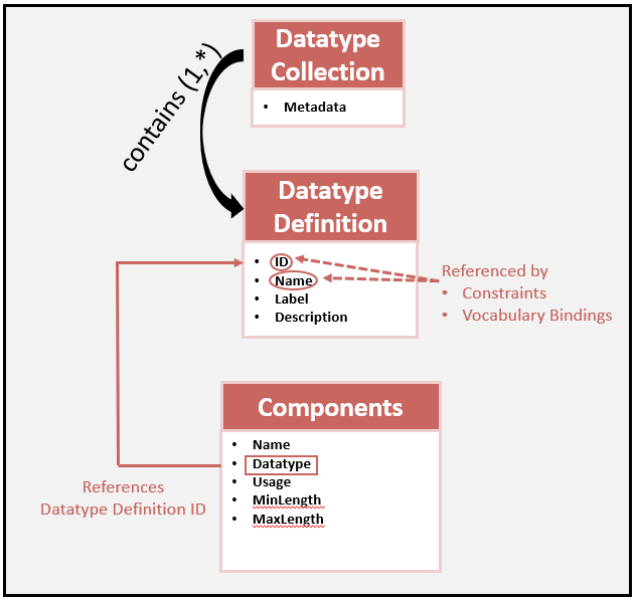

Figure 10.3: Data Type Library Document Definition

Figure 10.3 shows the definition of the data type library that is referenced in fields and components. The data type library contains base-level data types and data type flavors used in the implementation guide.

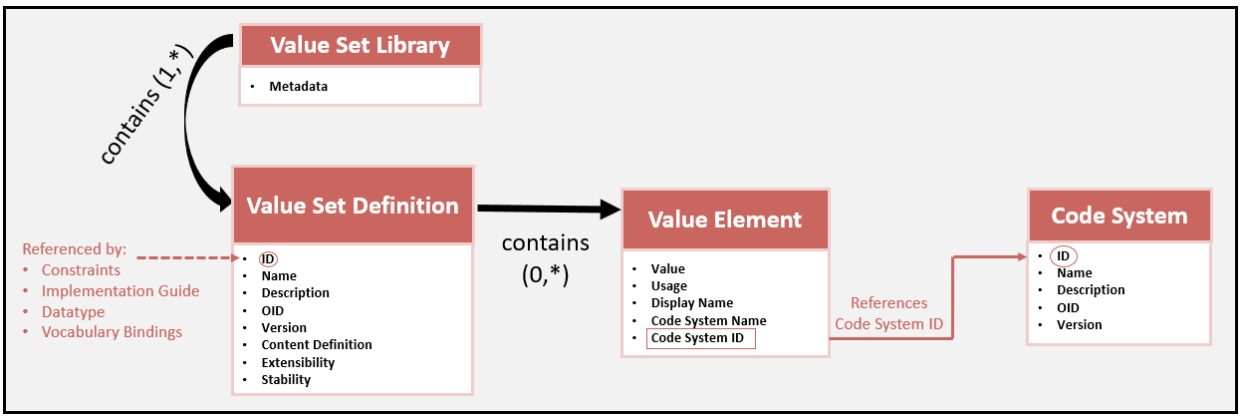

Figure 10.4: Value Set Library Document Definition

Figure 10.4 shows the value set library document definition. The value set library contains a list of value set definitions that provides the meta data and constraint attributes for each value set definition. Additionally, the value set definition contains the list of code values and their references to a code system.

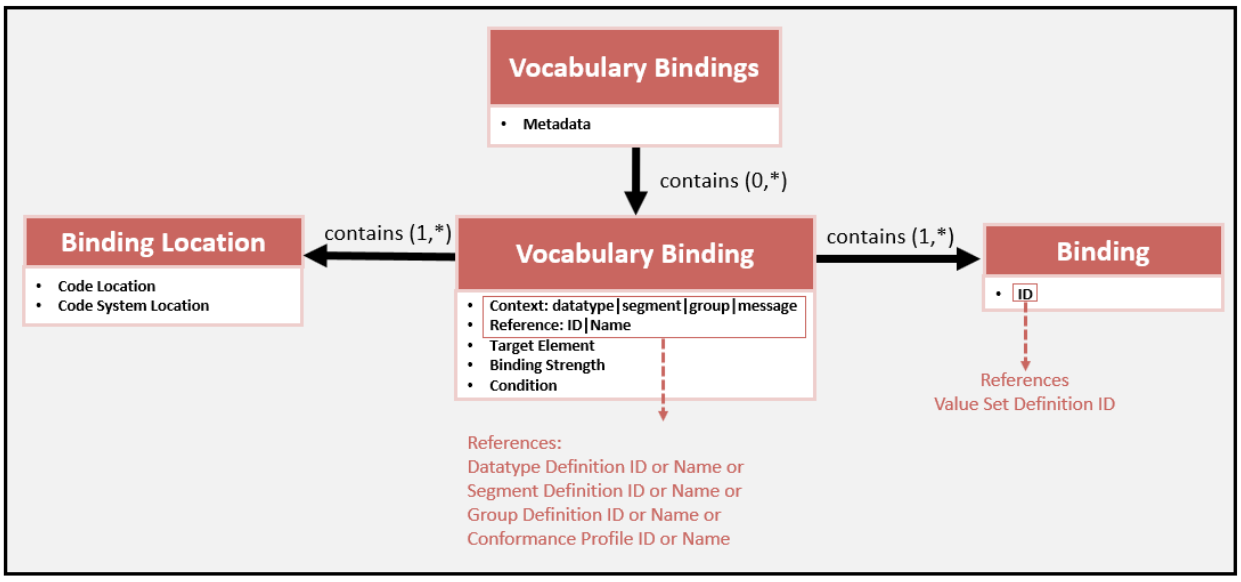

Figure 10.5: Vocabulary Binding Document Definition

Figure 10.5 shows the vocabulary binding document definition. The definition provides the link between the vocabulary definition and the element.

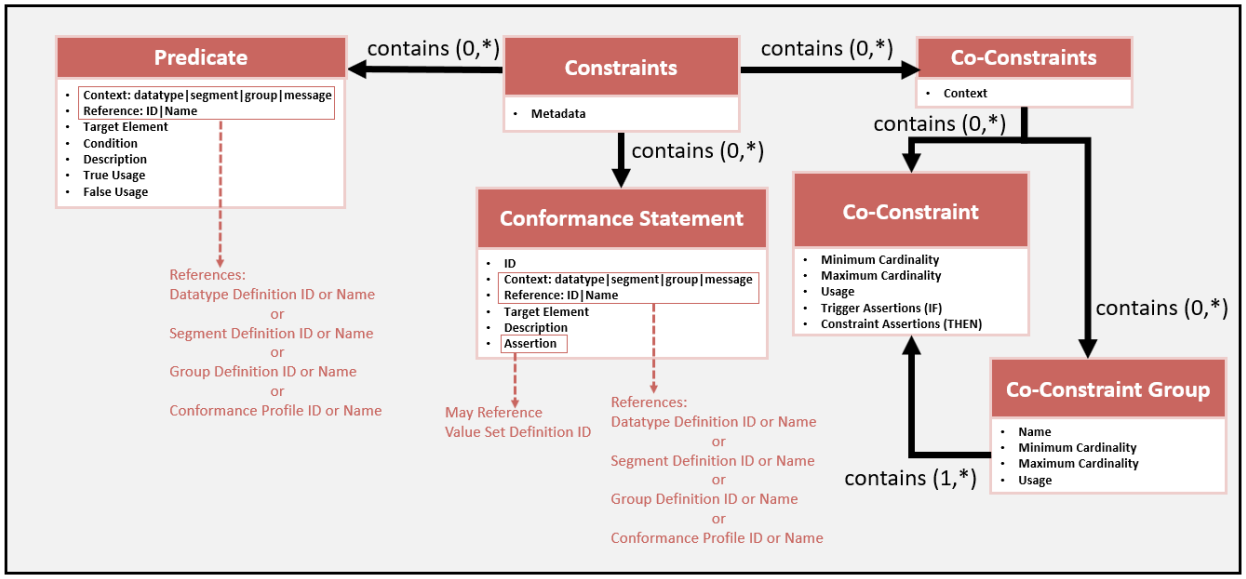

Figure 10.6: Condition Predicate, Conformance Statement and Co-constraints Document Definition

Figure 10.6 shows the document definition for the Condition Predicate, Conformance Statement, and Co-constraints. The definition provides a set of constraint types that can be applied to a given element at various levels in a message profile structure definition (e.g., segment).

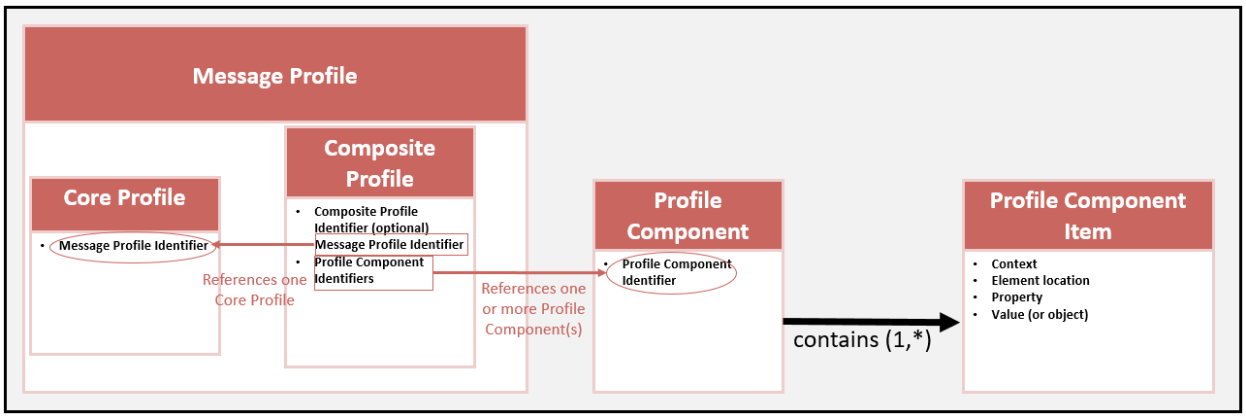

Figure 10.7: Profile Construction Components Relationships

Figure 10.7 shows the relationships among the message profiles and how a message profile can be constructed. A complete document definition can be contained in a message profile. However, a message profile may be constructed using various parts. A core profile and one or more profile components can be combined to create a composite profile, which is a message profile.

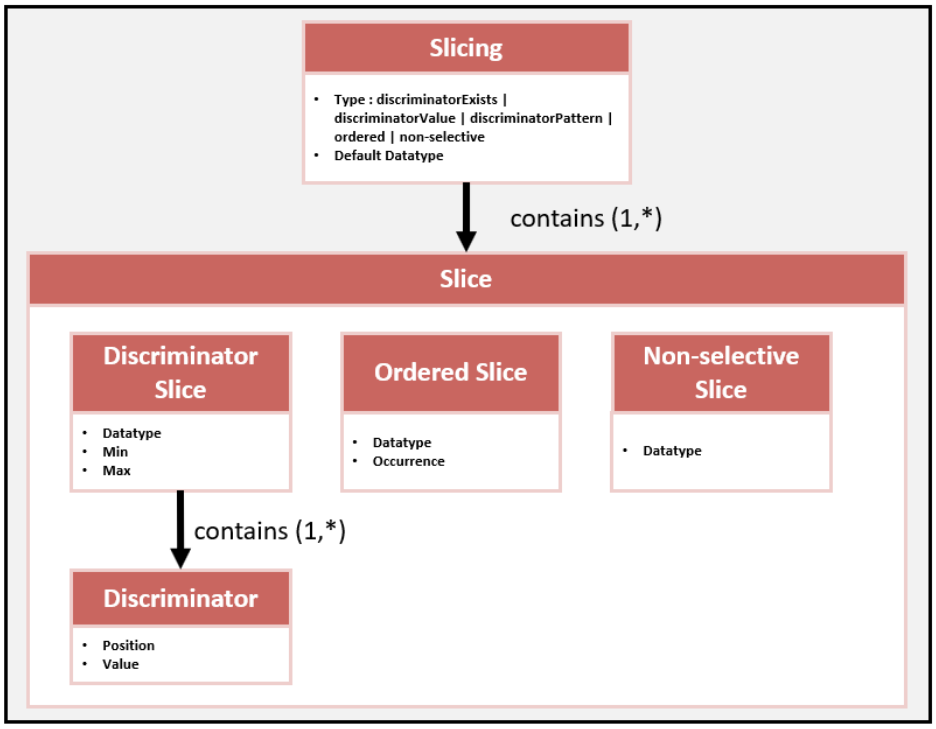

Figure 10.8: Slicing Document Definition

Figure 10.8 shows the document definition for the Slicing mechanism.

10.2 Meta Data

Meta data is provided for both an implementation guide and message profile.

10.2.1 IMPLEMENTATION GUIDE META DATA

Table 10.1 provides a list of the meta data and definition for an HL7 v2 implementation guide. The attribute name, definition, type, and use are given. Type indicates how the value of the attribute is represented. Use indicates whether the meta data is Required (R) Optional (O), or Conditional (C). Date Format is YYYYMMDD.

Table 10.1: Implementation Guide Meta Data

Attribute	Definition	Type	Use
Name	The natural language name of the implementation guide. Commonly used as the title of the implementation guide.	String	R
IG Version	The version of the implementation guide as decided by the author. There is no prescribed format.	String	O
Organization	The name/identification of the organization or owner of the implementation guide.	String	R

Author	List of authors that created the implementation guide.	String	O
HL7 Version	List of HL7 profile versions included in the implementation guide. Generally, most implementation guides include message profiles of the same HL7 version.	String	O
Create Date	The date the implementation guide was created.	Date Format	O
Last Update Date	The date the implementation guide was last updated.	Date Format	O
Status	The status of the implementation guide. Can be draft, published, superseded, or withdrawn.	Enum	R
Publish Date	If the status is "published", this is the date the implementation guide was published. Once published, the IG version is static.	Date Format	C
Description	Provides a short summary and intended use of the implementation guide.	String	O

Figure 10.9 shows the life cycle of an implementation guide.

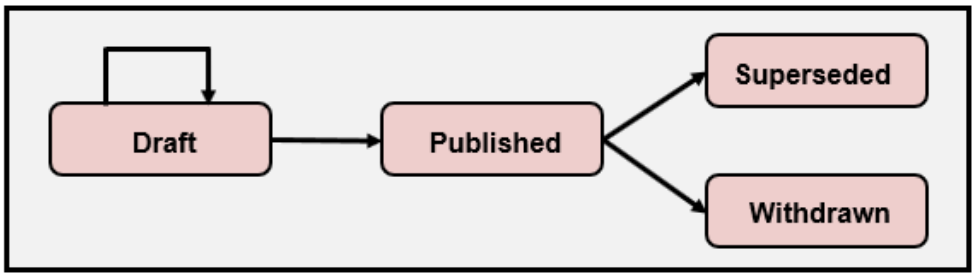

Figure 10.9: Implementation Guide Life Cycle

10.2.2 MESSAGE PROFILE META DATA

Table 10.2 provides a list of the meta data and definitions for an HL7 v2 message profile. The attribute name, definition, type, and use are given. Type indicates how the value of the attribute is represented. Use indicates whether the meta data is Required (R), Optional (O), or Conditional (C). Date Format is YYYYMMDD.

Table 10.2: Message Profile Meta Data

Attribute	Definition	Type	Use
Name	The natural language name of the message profile. Commonly used as the title of the implementation guide.	String	R
HL7 Version	The HL7 foundational version of the message profile; i.e., the version of the HL7 v2 abstract message definition. Corresponds to MSH-12.1. Value is obtained from table HL70104.	Enum	R
Organization	The name/identification of the organization or owner of the message profile.	String	O
Author	List of authors that created the message profile.	String	O
Message Code	The HL7 v2 Message Code (e.g., ADT). Corresponds to MSH-9.1. Value is obtained from table HL70076.	Enum	R
Trigger Event	The HL7 v2 Trigger Event (e.g., A04). Corresponds to MSH-9.2. Value is obtained from table HL70003.	Enum	R
Message Structure	The HL7 v2 Message Structure (e.g., ADT_A01). Corresponds to MSH-9.3. Value is obtained from table HL70354.	Enum	R
Level	The type (level) of message profiles (HL7, Constrainable, or Implementation).	Enum	R
Role	The role (perspective) of the message profile (Sender, Receiver, or Both).	Enum	O
Create Date	The date the message profile was created.	Date Format	O
Last Update Date	The date the message profile was last updated.	Date Format	O
Status	The status of the message profile. Can be draft, published, superseded, or withdrawn.	Enum	R
Publish Date	If the status is “published”, this is the date the message profile was published. Once published, the message profile version is static.	Date Format	C
Description	Provides a short summary and intended use of the message profile.	String	O

Message Profile Identifier	The Message Profile Identifier meta data is consistent with the definition of MSH-21 (Message Profile Identifier) specified in the message profile. The Message Profile Identifier Element may have an unlimited number of occurrences.	EI Data Type	R
Profile Type	Indicates the type of profile represented. Can be (profile, profile component, composite profile, or value set collection).	Enum	R

11 GLOSSARY

Term	Context	Definition
Accept Acknowledgement	Messaging	The acknowledgement message that is sent by the receiving system when the message is successfully received, indicating whether the received interaction can or cannot be accepted for further processing.
Actor	Use Case	An information system or a component of an information system that produces, manages, or acts on information associated with operational activities in an organization; has a specific role, performs specific functions, and is defined in an abstract manner. Actors exchange information through standard HL7 v2 messages and can be an initiator or a responder.
Adopted Object	Selective Adoption	An object from another version of the standard that is imported into a message profile.
Adopted Version	Selective Adoption	The HL7 v2 version from which an object was adopted.
Annotation	Constraint Type	Descriptive text that accompanies a standard element definition or concept and provides additional information pertaining to the use of the element (i.e., it is an elaboration of the concept as it relates to the use case to which it is being applied); may be associated at any defined element level in a message profile (e.g., a field).
Application Acknowledgement	Messaging	The acknowledgement message that is sent by the receiving system after the message is successfully and functionally processed by the receiving system.
Base Data Type	Slicing	The HL7 base data type for the field as defined in Chapter 2 of the base standard. The default data type and all the data type flavors associated with that field must derive from the base data type.
Base Standard	Standards Development	The base HL7 v2 standard.

Term	Context	Definition
Binding	Vocabulary Constraints	The association of a coded data element to a vocabulary. Depending on the level of the specification, the binding can be to a concept domain, code system, or value set. The HL7 v2 tables can represent any of these three vocabulary types depending on how the table is defined and used. At each specification (profile) level, the binding becomes increasingly specific, refining the data semantics of the element by limiting the vocabulary's content to a particular set of coded values.
Binding Parameters	Vocabulary Constraints	Define allowable options for code set modifications in derived profiles. Binding parameters include **extensibility** and **stability**.
Binding Strength	Vocabulary Constraints	Indicates the conformance of the binding, that is, whether the vocabulary must be used or not. There are two possible values: Required (R) and Suggested (S) (Recommended). All bindings are eventually required in implementations, and a suggested binding can be specified in a constrainable profile (i.e., the specification needs to be further constrained before implementation).
Cardinality	Constraint Type	Indicates the number of occurrences for an element by specifying the minimum and maximum bounds.
Co-Constraint	Constraint Type	A rule (constraint) used to express dependencies among a set of data values.
Coded Data Element	Vocabulary Constraints	An element with a data type definition that supports coded concepts. Examples include IS, ID, CE, CWE, and CNE and any data type flavors of coded data type definitions (e.g., CWE_01).
Code System	Vocabulary Constraints	A managed collection of codes that represent concepts used in a particular business or technical area; often there are relationships between the coded concepts.
Compatibility	Standards Development and Implementation	Declaration as to whether two specifications define sets of requirements that are harmonized with each other, allowing systems that implement them to *work* together, i.e., interoperate.

Term	Context	Definition
Compliance	Standards Development and Implementation	The degree to which a derived specification adheres to the requirements defined in the foundational specification (standard), i.e., follows the rules for adding constraints. Example of a usage compliance rule: a "required" element must remain "required" in any derived specifications (profiles).
Composite Data Type	Structure Definition	A data type that contains more than one element, which can be another composite data type or be a primitive data type. Also referred to as complex data type.
Composite Profile	Profile Construction	A profile created by combining a core profile component and one or more profile components.
Concept Domain	Vocabulary Constraints	An abstract notion that refers to a set of related ideas (concepts) that serve to help define the meaning of a particular data element (over and above just the name of the data element).
Conformance	Standards Development and Implementation	The relationship between a specification and an implementation; is an indication of how closely the software implements the requirements stated in the specification. An objective measure of how closely an implementation satisfies the stated requirements; is associated with the utilization of formal testing to verify adherence to the standard.
Conformance Clause	Implementation Guides	Defines the requirements, criteria, or conditions that must be satisfied by an implementation to claim conformance to the specification. The conformance clause identifies what must conform and how conformance can be met. This information typically is provided via a list of profiles or use cases that contain the profiles.
Conformance Constructs	Constraints	The key mechanisms for profiling; include usage, cardinality, data type, vocabulary, length, content, slicing, conformance statements, co-constraint, and semantic refinement.
Conformance Length	Constraint Type	The minimum for a maximum Length; sets the minimum number of characters that an implementation must support for an element.

Term	Context	Definition
Conformance Statement	Constraint Type	An explicit declaration conveyed in text or as a testable expression that defines a constraint.
Constraint	Standards Development and Implementation	A rule that is applied to narrow or focus the use of an element in accordance with a use case; is the central feature of the refinement process when creating profiles, as a constraint reduces the generality of the specification and focuses it on a particular requirement; enables implementers to understand how to use the various data structures and how to populate those structures with business-relevant information in accordance with the requirements detailed by the subject matter experts.
Content	Constraint Type	Defines the rules related to the data (value) for an element, such as a vocabulary, fixed value, or adherence to a format.
Constrainable Profile	Profile Levels	Is derived from either the standard profile or another constrainable profile. It further constrains the definition attributes in accordance with the profile compliance rules; for example, a U.S. state-level constrainable profile can be derived from a national-level profile by adding more constraints for a specific state.
Content Definition	Vocabulary Constraints	Indicates the method used to expand a value set definition into a set of codes. An *enumerated* value set definition is called an **Extensional** definition. A value set definition that is (defined) expanded based on an algorithm is called an **Intensional** definition.
Core Profile Component	Profile Construction	Is used to document the common set of requirements for a profile across the set of related definitions.
Data Type	Structured Definition and Constraint Type	Defines the data element structure and, at the primitive level, the type of data it may contain. Constraints include type substitution and specialization (when combined with other constraint types).
Data Type Flavor	Constraint Type	The mechanism for building on a data type for constraining components and sub-components; a specialization on a base data type.

Term	Context	Definition
Derived Profile	Implementation Guides	A "new" profile that is created and based on another profile. Profiling is an iterative process in which more detail is provided in each derived specification.
Discriminator Position	Slicing	The location of the discriminator element(s). The path is relative to the base datatype. All occurrences of the field have the same discriminator position. Discriminator position can be one or more elements (e.g., PID-11.6 (Country) and PID-11.7 (Address Type)).
Discriminator Value	Slicing	The content of the discriminator element. For a fixed value, the evaluation is based on a literal value. For exists, the evaluation is based on Boolean logic. For pattern, the evaluation is based on a regex (regular expression).
Empty Value	Messaging	An element lacking content. Indicated in er7 format for a field as ||.
Exception	Messaging	Is a general term to indicate that a conformance violation has occurred. The response to an exception is context dependent.
Extensibility	Vocabulary Profiling Mechanics	Indicates whether the value set definition can be extended (added to) or not in a derived (profiled) version of the specification. Extensibility can be either Open or Closed.
Extensional Definition	Vocabulary Constraints	In Content Definition, is an enumerated value set definition.
External Table	Vocabulary Constraints	A vocabulary definition created, maintained, and published by another standards organization. A specifier may reference an external table (code system) and bind that table to a data element in a message profile.
Fixed Value (Constant)	Constraint Type	Constrains the content to a single value, e.g., "A08" that indicates an update event type in ADT messages.
Foundational Version	Selective Adoption	The version of the base standard that serves as the core (starting) version of the message definition. All objects in the message profile will comply to this version unless explicitly overridden by an adopted object.

Term	Context	Definition
HL7-Defined Table	Vocabulary Constraints	Provides an initial table identifier, code values, and certain base standard requirements. Code values defined in an HL7 table must not be redefined in a message profile; and if a concept needed for a use case is not defined, the table definition can be extended.
Implementation Profile **(Implementable Profile)**	Profile Levels	Defines all elements such that all optionality and openness have been removed; typically, is derived from a constrainable profile, but it may be derived by further constraining another implementable profile. All interfaces deployed in a production setting are implementable profiles whether they are documented (explicitly stated) or not documented (implicitly stated).
Implementation Tolerance	Vocabulary Constraints	Allows for unknown (undocumented) codes to be exchanged at runtime with impunity, because the complete code set is not known to the implementers. For a binding that supports implementation tolerance, an implementation must process unknown codes without raising an exception.
Indigenous Object	Selective Adoption	An object from the foundational version of the message profile. Indigenous objects do not have an explicit version attribute. The version of the object is the version of the Foundational Version.
Initiator	Use Case	An entity (actor) that starts an action; is also referred to as a **Sender, Producer,** or **Client**.
Intensional Definition	Vocabulary Constraints	In Content Definition, is a means used to expand a value set definition into a set of codes include using regular expressions, characterizing specific attributes for a specific code system (e.g., all laboratory codes in LOINC), or using hierarchical associations that include all specializations of the codes in question.
Interaction	Messaging	A conversation in which two systems or applications (generically called Actors) exchange data.
Interoperability	Standards Development and Implementation	The ability of implementations to exchange data and to use that data as intended to accomplish a desired task.

Term	Context	Definition
Known Data	Usage	Content that exists and is available for a given element.
Length	Constraint Type	Defines the number of characters that may be present in one occurrence of an element. Can specify a maximum limit or both the minimum and maximum bounds.
Local Table	Vocabulary Constraints	A table with a non-HL7 assigned table-identifier that contains a set of locally or site-defined values. Local tables typically are used for elements with a CWE data type. During the process of profiling, a specifier will create and bind local tables.
Non-empty Value	Usage	A value in which at least one character is a non-whitespace character. Two double quotes (""), which represents a “delete indicator”, qualifies as a non-empty value.
Null Flavor (Null Value)	Messaging	A concept for representing “unknown” content. HL7 v2 does not explicitly provide a mechanism for null flavors; however, an equivalent outcome can be achieved for coded data elements by including a null flavor coded value in a value set.
Object	Selective Adoption	An object from the HL7 v2 standard; can be any element (group, segment, field), data type, or table.
Optionality	Constraint Type	An attribute that is used to define the presence of an element in a message definition. The openness permitted for an element; through the application of conformance rules to the base standard, the openness permitted by the base standard is restricted. Sometimes used interchangeably with “usage”.
Pattern Restricted Value	Constraint Type	A value that constrains the content of the element based on the specified pattern matching algorithm (e.g., a Medical Record Number format). A pattern-restricted value is specified using the conformance statement mechanism.

Term	Context	Definition
Process	Usage	A general term to indicate that an action must be performed; is an important concept for understanding receiver side requirements. Message usage requirements alone cannot fully indicate the scope of processing requirements. Additional functional requirements for how data elements should be process can be defined.
Profile Component	Profile Construction	Defines a part or a certain aspect of a profile; is used to aggregate correlated requirements and/or to differentiate requirements from another profile or profile component; provides a mechanism to support a set of reusable requirements and can be applied to any construct or section of a profile. Profiles and profile components can be combined to develop and manage other profiles. A profile component in a family of profiles can be used to identify different levels of requirements for the same use case or to identify the differences in requirements for different, but closely related, use cases.
Profiling	Standards Development and Implementation	The process of placing additional constraints on a message definition in accordance with defined profiling compliance rules to meet requirements stated in a use case. The terms "*derived profile*" and, more generically, "*derived specification*" are used to denote a technical documentation that is based on another technical document. Profiling is an iterative process in which more detail is provided in each derived specification.
Message Profile	Profile Construction	A complete message structure definition with additional constraints applied to it.
Message Interaction	Implementation Guides	The process by which a message is sent from a Sender application to a Receiver application.
Message Transaction	Implementation Guides	The process by which a message is sent from a Sender application to a Receiver application and one or more response(s) is sent from the Receiver application back to the Sender application.
Responder	Use Case	An entity (actor) that is reacting to the action of the Initiator; is also referred to as a **Receiver, Consumer,** or **Server**.

Term	Context	Definition
Semantic Refinement	Constraint Type	Allows for refinement of the semantics of a data element based on the use case.
Single Code Element Binding	Vocabulary Constraints	A vocabulary binding in which the binding to an element is to a single value selected from a vocabulary definition.
Slice Min	Slicing	An attribute that determines the minimum number of occurrences that must contain "discriminator value" and, therefore, conform to “slice data type flavor”.
Slice Max	Slicing	An attribute that determines the maximum number of occurrences that must contain "discriminator value" and, therefore, conform to “slice data type flavor”.
Slicing	Constraint Type	Defines a set of different constraints that are possible for a field instance (occurrence).
Specifier	Standards Developer	A person who is creating the implementation guide or message profile.
Stability	Vocabulary Profiling Mechanics	Indicates whether the content of a value set might change (often the value set is dependent on an external code system that may be updated or replaced after publication of the interoperability specification). Stability can be Static and Dynamic.
Standard Profile	Profile Levels	Represents the base standard definitions and constraints as-is and establishes the framework for a specific type of event (e.g., an unsolicited vaccination record update – VXU V04 message event). At this level, the overall structure (“template”), including the data type definitions, is established; however, the full declaration of requirements has yet to be specified, and, therefore, considerable openness still exists. Additional profiles are subsequently derived from the standard profile.
Table	Vocabulary Constraints	An HL7 v2 vocabulary mechanism to define coded concepts. A pre-defined table has an identifier and may contain codes; sometimes acts like a code system, value set, and/or a concept domain depending on how it is defined and used. Tables in HL7 v2 have two types: HL7 Table and User Table.

Term	Context	Definition
Transaction	Messaging	A sequence of events comprising a *roundtrip* conversation.
Usage	Constraint Type	Indicates requirements for the presence (appearance) of an element in a message instance, governing the expected behavior of the sending application and the receiving application with respect to the element. Determines, from an implementation perspective, whether an element must be supported. From an operational perspective, usage determines whether the element must be present, can be present, or must not be present in a message instance for the sender; and for the receiver it influences the processing of the element. Sometimes used interchangeably with "optionality".
Use Case	Standards Development	Describes a system capability and the associated sequences of actions that a system performs to achieve an intended outcome; can include the actors involved, the interactions (including the message exchanges) between the actors, system functional requirements, acknowledgement requirements, and any other pertinent information that aids in the understanding of an implementer, such as example messages or message excerpts. Can be considered a *container* that organizes the pieces needed to specify a system capability.
User-Defined Table	Vocabulary Constraints	Provides an initial table identifier and, in some cases, suggested code values; are recommendations and, therefore, place no initial requirements on the vocabulary definition. In a message profile, a specifier may use, extend, modify, or replace the suggested codes and descriptions.
Value (verb)	Messaging	To place a non-empty value (noun) in an element location.

Term	Context	Definition
Value Set	Vocabulary Constraints	The collection of codes that may be used to populate an element in a message instance; the codes usually are targeted for a specific use. A value set draws codes from one (usually) or more code systems; the result is a set of codes that constrains the possible content of a data element. A key distinction between a value set and a code system is that a code system is used as a reference source of coded meaning, whereas a value set is a specific constraint for a set of explicit business uses.
Value Set Binding	Vocabulary Constraints	Refers to assigning a set of coded concepts to a particular data element in order to implement the concept domain associated with the data element.
Value Set Definition	Vocabulary Constraints	Provides meta data and the codes. The meta data includes the name of the value set, the identifier, and value set properties ‘extensibility’ and ‘stability’.
Vocabulary	Constraint Type	Defines the vocabulary binding and vocabulary definitions. Vocabulary indicates the allowable values (content) for a coded element. Vocabulary is a term used to convey the notions of concept domain, code system, and value set in a general sense.

12 APPENDIX A: PREDICATE DEFINITION LANGUAGE

Elements assigned with a declared conditional usage require an associated predicate that is computable and based on other elements in the message. The conformance methodology specification does not prescribe a language to define the predicate. A recommended pseudo language[54] that provides the mechanisms to construct predicate statements is given in this section. The language is HL7 v2-specific and is grounded on predicate definitions that have emerged over time in the specification of HL7 v2 implementation guides. The goal is to provide specifiers with a simple language that affords consistency within and across HL7 v2 implementation guides, affords readability of a natural language, and is machine computable. The language is not comprehensive (nor formal) and is intended for use in HL7 v2 for convenience.

Examples of Condition Predicates:

IF CWE.1 (Identifier) is valued.

IF RXA-20 (Completion Status) contains one of the values in the list: {'CP', 'PA'}.

The pseudo language for describing predicate definition is composed of building blocks that are linked together to create the predicates. The building blocks include:

- Element Location
- Proposition Declarative Statements
  - Occurrence
  - Content
  - Content Comparative
- Context
- Verbs
- Connectors

An example template for the condition predicate builder follows this general pattern:

Condition Predicate Example using a template pattern:

IF LOCATION (DESCRIPTION) VERB one of the values in the list: {'VALUE 1', 'VALUE 2', 'VALUE N'}.

Example:

IF RXA-20 (Completion Status) contains one of the values in the list: {'CP', 'PA'}.

[54] Developed by NIST for use in the Implementation Guide Management and Authoring Tool (IGAMT). IGAMT facilitates the creation of implementation guides and message profiles.

Table 12.1 shows the template pattern broken up into its building-block parts. Additionally, a usage is given for each building block. Some building blocks are a part of all condition predicates (required), and others may be part of the condition predicates (optional). Subsequent sections explain the details for each of the building blocks.

Table 12.1: Condition Predicate Template

Example Template Pattern	Template Example	Example	Usage
IF	IF	IF	Required
Occurrence Declarative Statement	N/A	N/A	Optional
Location	LOCATION (DESCRIPTION)	RXA-20 (Completion Status)	Required
Context	N/A	N/A	Optional
Verb	VERB	contains	Required
Content Declarative Statement OR Content Comparison Declarative Statement	contains the value one of the values in the list: {'VALUE 1' 'VALUE 2', 'VALUE N'}.	one of the values in the list: {'CP', 'PA'}.	Required

12.1 IF Statement

Every condition predicate starts with an "IF" statement.

12.2 Occurrence-Declarative Statement

A condition predicate may refer to a specific instance of an element or to a group of element instances. Identifying an instance or instances is accomplished with the occurrence-declarative statement. Table 12.2 shows the list of pre-defined occurrence-declarative statement patterns and their meanings.

Table 12.2: Occurrence-Declarative Statement

Occurrence-Declarative Statement	Description
at least one occurrence of	One or more occurrence(s) of a repeating element can satisfy the predicate.
the 'INSTANCE' occurrence of	The occurrence indicated by the INSTANCE of a repeating element must satisfy the predicate. Other instances may also satisfy the predicate.
no occurrence of	No occurrence of a repeating element can satisfy the predicate.
exactly one occurrence of	One and only one occurrence of a repeating element can

Occurrence-Declarative Statement	Description
	satisfy the predicate.
'COUNT' occurrences of	Exactly COUNT occurrences of a repeating element can satisfy the predicate.
All occurrences of	All occurrences of a repeating element must satisfy the predicate.

12.3 Element Location

The Element Location indicates the location that is being addressed in the message. The location can reference a group, segment, field, component, or sub-component and includes a description. For details of how an element location is represented, see Section 1.2.3.

12.4 Context

The meaning of a condition predicate can be different depending on the context or scope to which it applies. Table 12.3 shows the list of pre-defined context patterns and their meanings. Note that the context of "LOCATION" indicates the location at any level. To indicate a specific context, the condition predicate may be enhanced with the context provided in Table 12.3.

Table 12.3: Condition Predicate Context

Context (Scope/Location)	Description
in the same GROUP group	"same group", e.g., "in the same ORDER group"

12.5 Verbs

Table 12.4 indicates the set of verbs that can be used to construct predicate inquires. The verbs are paired with certain Proposition Content Declarative Statements or Content Comparison Declarative Statements.

Table 12.4: Condition Predicate Verbs

Verbs
is
is not
contains
does not contain
matches
does not match

12.6 Content-Declarative Statements

Proposition content-declarative statements ask whether an element contains certain content. The values can be arbitrary values, coded values, or restricted values based on a format. The values

may or may not contain an associated description. Table 12.5 lists each of the statements along with the verbs they can be paired with.

Table 12.5: Content-Declarative Statement

Declarative Statement	Verb Pairs
valued.	is/is not
the value ‘VALUE’.	contains/does not contain
the value ‘VALUE’ (DESCRIPTION).	contains/does not contain
the value ‘VALUE’ drawn from the code system 'CODE SYSTEM'.	contains/does not contain
the value ‘VALUE’ (DESCRIPTION) drawn from the code system 'CODE SYSTEM'.	contains/does not contain
one of the values in the list: { ‘VALUE 1’, 'VALUE 2', 'VALUE N' }.	contains/does not contain
one of the values in the list: { ‘VALUE 1’ (DESCRIPTION), 'VALUE 2' (DESCRIPTION), 'VALUE N' (DESCRIPTION) }.	contains/does not contain
one of the values in the list: { ‘VALUE 1’, 'VALUE 2', 'VALUE N' } drawn from the code system 'CODE SYSTEM'.	contains/does not contain
one of the values in the list: { ‘VALUE 1’ (DESCRIPTION), 'VALUE 2 (DESCRIPTION)', 'VALUE N' (DESCRIPTION) } drawn from the code system 'CODE SYSTEM'.	contains/does not contain
the regular expression ‘REGULAR EXPRESSION’.	matches/does not match

For ‘REGULAR EXPRESSION’, content is required to match the regular expression. Regular expressions of “MR\d{5}” indicate that the content must start with “MR” and be followed with 5 digits. Example is “MR83452”.

12.7 Comparison-Content Declarative Statement

Proposition comparison-content declarative statements ask whether the content of one element matches that of another element based on a comparator. Table 12.6 lists each of the statements along with a description and examples. All Content Comparison Declarative Statement are paired with the verbs is/is not.

Table 12.6: Content Comparison Declarative Statement

Comparative Statement	Description	Examples
be identical to LOCATION 2 (DESCRIPTION).	Content is the same in meaning and representation (a literal identical to). Applies to general, coded values, and data/time content. This concept can also be applied at the complex element level (e.g., compare ORC-12 and OBX-16).	General: 3 is identical to 3. Coded Value: The LOINC code of 30963-3 is identical to 30963-3. Date/Time: 201103041023-0400 is identical to 201803041023-0400 (unlikely to be used for time). Complex element: Each constituent part of the complex

Comparative Statement	Description	Examples
		element is compared for identical content.
be earlier than LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs before the date/time of the element it is compared to. The comparison is a "meaning" comparison.	Date/Time: 201803041021-0400 is earlier than 201803040823-0600.
be earlier than or equivalent to LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs before or at the date/time of the element it is compared to. The comparison is a "meaning" comparison.	Date/Time: 201803041021-0400 is earlier than 201803040823-0600. Date/Time: 201803041023-0400 is equivalent to 201803040823-0600.
be truncated earlier than LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs before the date/time of the element it is compared to. The comparison is a "meaning" comparison and is at the coarsest level (i.e., the more detailed element is truncated to the resolution of the less detailed element).	Date/Time: 20180302 is earlier than 201803040823-0600.
be truncated earlier than or truncated equivalent to LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs before or at the date/time of the element it is compared to. The comparison is a "meaning" comparison and is at the coarsest level (i.e., the more detailed element is truncated to the resolution of the less detailed element).	Date/Time: 20180304 is equivalent to 201803040823-0600. Date/Time: 20180302 is earlier than 201803040823-0600.
be equivalent to LOCATION 2 (DESCRIPTION).	Content is the same in meaning but not representation. Applies to general, coded values, and data/time content. This concept can also be applied at the complex element level (e.g., compare ORC-12 and OBX-16).	General: 3.00 is equivalent to 3. Coded Value: C38288 (NCIT) is equivalent to PO (HL70162) (Oral - administered by mouth.) Note: A concept map must be specified. Date/Time: 201803041023-0400 is equivalent to 201803040823-0600. Complex element: Each constituent part of the complex

Comparative Statement	Description	Examples
		element is compared for identical content.
be truncated equivalent to LOCATION 2 (DESCRIPTION).	Content is the same in meaning but not represented in the same manner and/or having the same resolution. Applies to general and data/time content. The comparison is at the coarsest level (i.e., the more detailed element is truncated to the resolution of the less detailed element).	General: 3.56 is truncated equivalent to 3. Date/Time: 20110304 is equivalent to 201803040823-0600.
be equivalent to or later than LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs after or at the date/time of the element it is compared to. The comparison is a "meaning" comparison.	Date/Time: 201803041023-0400 is equivalent to 201803040823-0600. Date/Time: 201803041025-0400 is later than 201803040823-0600.
be later than LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs after the date/time of the element it is compared to. The comparison is a "meaning" comparison.	Date/Time: 201803041025-0400 is later than 201803040823-0600.
be truncated equivalent to or truncated later than LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs after or at the date/time of the element it is compared to. The comparison is a "meaning" comparison and is at the coarsest level (i.e., the more detailed element is truncated to the resolution of the less detailed element).	Date/Time: 20180304 is equivalent to 201803040823-0600. Date/Time: 20180305 is later than 201803040823-0600.
be truncated later than LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs after the date/time of the element it is compared to. The comparison is a "meaning" comparison and is at the coarsest level (i.e., the more detailed element is truncated to the resolution of the less detailed element).	Date/Time: 20180305 is later than 201803040823-0600.

12.8 Complex Condition Predicates

Condition predicate statements can be combined to make complex condition predicates with a set of connectors (See Table 12.7). Multiple connectors can be used.

Table 12.7: Condition Predicate Connectors

Connectors	Description
AND	Both predicates must be satisfied
OR	One predicate must be satisfied; both may be satisfied
XOR	Exclusive OR—one predicate must be satisfied; both must not be satisfied

An example of a complex condition predicate is:

Example:

If RXA-9.1 (Identifier) contains the value '00' AND RXA-20 (Completion Status) contains one of the values in the list: {CP, PA}.

13 APPENDIX B: CONFORMANCE STATEMENT DEFINITION LANGUAGE

Certain conformance-statement patterns have emerged over time and can be documented in a "pseudo" language. The pseudo language[55] provides the mechanisms to construct conformance statements in a manner that provides the readability of a natural language while being machine computable. The outcome is a set of conformance-statement patterns and building blocks that provides a mechanism for specifiers to create and use a set of conformance statements. Using the conformance statements language provides consistency within and across HL7 v2 implementation guides. The pseudo language is not comprehensive (nor formal) and is intended for use in HL7 v2 for convenience. Constraints that cannot be represented within the pseudo language are written in "free-text". Free-text conformance statements are not conducive to automated machine computability. Computability greatly enhances processing of the constraints used, for example, in validation tools. The HL7 v2 Pseudo Constraint Language is the preferred constraint language for HL7 v2 message profiles but is not required.

The HL7 v2 Pseudo Constraint Language is composed of building blocks that are linked together to create conformance statements. The building blocks include:

- Element Location
- Propositions
- Context
- Conformance Verbs
- Declarative Statements
    - Occurrence Declarative Statement (ODS)
    - Content Declarative Statement (CDS)
    - Comparison Content Declarative Statement (CCDS)
- Connectors

An example template for the conformance-statement builder is given below:

Conformance Statement Example using a template pattern:

IF LOCATION (DESCRIPTION) contains the value 'VALUE', THEN at least one occurrence of LOCATION (DESCRIPTION) of the SEGMENT segment VERB contain the value 'VALUE'.

Instance Example:

IF MSA-1 (Acknowledgment Code) contains the value 'AR', THEN at least one occurrence of ERR-4 (Severity) of the ERR segment SHALL contain the value 'E'.

[55] Developed by NIST for use in the Implementation Guide Management and Authoring Tool (IGAMT). IGAMT facilitates the creation of implementation guides and message profiles.

Table 13.1 shows the template pattern broken up into its building-block parts. Additionally, a usage is given for each building block. Some building blocks are a part of all conformance statements (required), and others may be part of the conformance statement (optional).

Table 13.1: Conformance Statement Template

Pattern Part	Template Example	Example	Usage
Proposition	IF LOCATION (DESCRIPTION) contains the value 'VALUE', THEN	IF MSA-1 (Acknowledgment Code) contains the value 'AR', THEN	Optional
ODS	at least one occurrence of	at least one occurrence of	Optional
Location	LOCATION (DESCRIPTION)	ERR-4 (Severity)	Required
Context	of the SEGMENT segment	of the ERR segment	Optional
Verb	VERB	SHALL	Required
CDS or CCDS	contain the value 'VALUE'.	contain the value 'E'.	Required

An example of a conformance statement with the minimal required building blocks is:

Example:
OBX-11 (Observation Result Status) SHALL contain the value 'F'.

13.1 Element Location

The Element Location indicates the location that is being addressed in the message. The location can reference a group, segment, field, component, or sub-component and includes a description. For details of how an element location is represented, see Section 1.2.3.

13.2 Propositions

A conformance statement may be predicated on a condition, which is a building block called a proposition. Table 13.2 shows the list of pre-defined proposition patterns and their meanings.

Table 13.2: Conformance Statement Propositions

Propositions (Predicates)	Description
IF LOCATION (DESCRIPTION) is valued, THEN	"is valued" predicate
IF LOCATION (DESCRIPTION) is not valued, THEN	"is not valued" predicate
IF LOCATION (DESCRIPTION) contains the value 'VALUE', THEN	"dependent value" predicate
IF LOCATION (DESCRIPTION) contains the value 'VALUE' (DESCRIPTION), THEN	"dependent value with description" predicate
IF LOCATION (DESCRIPTION) does not contain the value 'VALUE', THEN	"not dependent value" predicate
IF LOCATION (DESCRIPTION) does not contain the value 'VALUE' (DESCRIPTION), THEN	"not dependent value with description" predicate
IF LOCATION (DESCRIPTION) contains one of the values in the list {	"dependent value in list"

Propositions (Predicates)	Description
'VALUE 1', 'VALUE 2', 'VALUE N' }, THEN	predicate
IF LOCATION (DESCRIPTION) contains one of the values in the list { 'VALUE 1' (DESCRIPTION) , 'VALUE 2' (DESCRIPTION), 'VALUE N' (DESCRIPTION) }, THEN	"dependent value in list with description" predicate
IF LOCATION (DESCRIPTION) does not contain one of the values in the list { 'VALUE 1', 'VALUE 2', 'VALUE N' }, THEN	"not dependent value in list" predicate
IF LOCATION (DESCRIPTION) does not contain one of the values in the list { 'VALUE 1' (DESCRIPTION) , 'VALUE 2' (DESCRIPTION) , 'VALUE N' (DESCRIPTION) }, THEN	"not dependent value in list with description" predicate

The key phrases in the proposition are “is valued” and “contains the value”; a brief definition follows:

is-valued: Any non-empty value satisfies the predicate proposition.

contains the value: A value that is an exact match of value satisfies the predicate proposition. An indication of whether the exact match is case sensitive can be noted.

13.3 Occurrence-Declarative Statement

A conformance statement may refer to a specific instance of an element or to a group of element instances. Identifying an instance or instances is accomplished with the occurrence-declarative statement. Table 13.3 shows the list of pre-defined occurrence-declarative statement patterns and their meanings.

Table 13.3: Occurrence Declarative Statement

Occurrence Declarative Statement	Description
At least one occurrence of	One or more occurrence(s) of a repeating element can satisfy the predicate.
The 'INSTANCE' occurrence of	The occurrence indicated by the INSTANCE of a repeating element must satisfy the predicate. Other instances may also satisfy the predicate.
No occurrence of	No occurrence of a repeating element can satisfy the predicate.
Exactly one occurrence of	One and only one occurrence of a repeating element can satisfy the predicate.
'COUNT' occurrences of	Exactly COUNT occurrences of a repeating element can satisfy the predicate.
All occurrences of	All occurrences of a repeating element must satisfy the predicate.

An example of a conformance statement using an occurrence-declarative statement is:

Example:

Exactly one occurrence of MSH-21.1 (Entity Identifier) SHALL contain the value 'Z22r1.0'.

13.4 Context

The meaning of a conformance statement can be different depending on the context or scope to which it applies. Table 13.4 shows the list of pre-defined context patterns and their meanings. Note that the context of "LOCATION" indicates the location at any level. To indicate a specific context, the conformance statement may be enhanced with the contexts provided in Table 13.4.

Table 13.4: Conformance Statement Context

Context (Scope/Location)	Description
of the SEGMENT segment	"segment", e.g., "ERR segment"
in the same GROUP group	"same group", e.g., "in the same ORDER group"

13.5 Verbs

Conformance statements indicate the strength of the constraint with the use of the conformance verbs (See Table 13.5). The "SHALL" and "SHALL NOT" verbs indicate that the constraint must be implemented. The other verbs indicate constraints that may be applied. The "SHOULD" and "SHOULD NOT" verbs indicate constraints that are recommended but are not mandated.

Table 13.5: Conformance Statement Verbs

Verbs	Description
SHALL	Requirement to implement the constraint statement
SHALL NOT	Requirement not to implement the constraint statement
SHOULD	Recommendation to implement the constraint statement
SHOULD NOT	Recommendation not to implement the constraint statement
MAY	An indication that the constraint statement can be implemented

13.6 Content-Declarative Statement

Content-declarative statements ask whether an element contains certain content. The values can be arbitrary values, coded values, or restricted values based on a format. The values may or may not contain an associated description.

Table 13.6: Content-Declarative Statement

Declarative Statement	Description
contain the value 'VALUE'.	Simple Value
contain the value 'VALUE' (DESCRIPTION).	Simple Value with Description
contain the value 'VALUE' drawn from the code system 'CODE SYSTEM'.	Code Value from Code System

Declarative Statement	Description
contain the value 'VALUE' (DESCRIPTION) drawn from the code system 'CODE SYSTEM'.	Code Value with Description from Code System
contain one of the values in the list: { 'VALUE 1', 'VALUE 2', 'VALUE N' }.	Simple Value in List
contain one of the values in the list: { 'VALUE 1' (DESCRIPTION), 'VALUE 2' (DESCRIPTION), 'VALUE N' (DESCRIPTION) }.	Simple Value with Description in List
contain one of the values in the list: { 'VALUE 1', 'VALUE 2', 'VALUE N' } drawn from the code system 'CODE SYSTEM'.	Code Value in List from Code System
contain one of the values in the list: { 'VALUE 1' (DESCRIPTION), 'VALUE 2 (DESCRIPTION)', 'VALUE N' (DESCRIPTION) } drawn from the code system 'CODE SYSTEM'.	Code Value with Description in List from Code System
match the regular expression 'REGULAR EXPRESSION'.	Content is required to match the regular expression. Regular expressions of "MR\d{5}" indicate that the content must start with "MR" and be followed with 5 digits. Example is "MR83452".
contain a *positive integer*.	Content is required to be a positive integer. This is a specific instance of a regular expression and provided for convenience.
be valued sequentially starting with the value '1'.	Content of the first occurrence of an element must be '1', subsequent occurrences are required to increase by 1. Used for identifying instances of segments using the Set ID field.
be valued with an *ISO-compliant OID*.	Content is required to be a regular expression that represents the format of an ISO-compliant OID. This is a specific instance of a regular expression and provided for convenience.

13.7 Comparison-Content Declarative Statement

Comparison-content declarative statements ask whether the content of one element matches that of another element based on a comparator. Table 13.7 lists each of the statements along with a description and examples.

Table 13.7: Comparison-Content Declarative Statement

Comparative Statement	Description	Examples
be identical to LOCATION 2 (DESCRIPTION).	Content is the same in meaning and representation (a literal	General: 3 is identical to 3.

Comparative Statement	Description	Examples
	identical to). Applies to general, coded values, and date/time content. This concept can also be applied at the complex element level (e.g., compare ORC-12 and OBX-16).	Coded Value: The LOINC code of 30963-3 is identical to 30963-3. Date/Time: 201103041023-0400 is identical to 201803041023-0400 (unlikely to be used for time). Complex element: Each constituent part of the complex element is compared for identical content.
be earlier than LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a data/time that occurs before the date/time of the element it is compared to. The comparison is a "meaning" comparison.	Date/Time: 201803041021-0400 is earlier than 201803040823-0600.
be earlier than or equivalent to LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs before or at the date/time of the element it is compared to. The comparison is a "meaning" comparison.	Date/Time: 201803041021-0400 is earlier than 201803040823-0600. Date/Time: 201803041023-0400 is equivalent to 201803040823-0600.
be truncated earlier than LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs before the date/time of the element it is compared to. The comparison is a "meaning" comparison and is at the coarsest level (i.e., the more detailed element is truncated to the resolution of the less detailed element).	Date/Time: 20180302 is earlier than 201803040823-0600.
be truncated earlier than or truncated equivalent to LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs before or at the date/time of the element it is compared to. The comparison is a "meaning" comparison and is at the coarsest level (i.e., the more detailed element is truncated to the resolution of the less detailed element).	Date/Time: 20180304 is equivalent to 201803040823-0600. Date/Time: 20180302 is earlier than 201803040823-0600.
be equivalent to LOCATION 2 (DESCRIPTION).	Content is the same in meaning but not representation. Applies	General: 3.00 is equivalent to 3. Coded Value: C38288 (NCIT) is

Comparative Statement	Description	Examples
	to general, coded values, and data/time content. This concept can also be applied at the complex element level (e.g., compare ORC-12 and OBX-16).	equivalent to PO (HL70162). Oral-administered by mouth. Note: A concept map must be specified. Date/Time: 201803041023-0400 is equivalent to 201803040823-0600 Complex element: Each constituent part of the complex element is compared for identical content.
be truncated equivalent to LOCATION 2 (DESCRIPTION).	Content is the same in meaning but not represented in the same manner and/or having the same resolution. Applies to general and date/time content. The comparison is at the coarsest level (i.e., the more detailed element is truncated to the resolution of the less detailed element).	General: 3.56 is truncated equivalent to 3. Date/Time: 20180304 is equivalent to 201803040823-0600.
be equivalent to or later than LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs after or at the date/time of the element it is compared to. The comparison is a "meaning" comparison.	Date/Time: 201803041023-0400 is equivalent to 201803040823-0600. Date/Time: 201803041025-0400 is later than 201803040823-0600.
be later than LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs after the date/time of the element it is compared to. The comparison is a "meaning" comparison.	Date/Time: 201803041025-0400 is later than 201803040823-0600.
be truncated equivalent to or truncated later than LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time that occurs after or at the date/time of the element it is compared to. The comparison is a "meaning" comparison and is at the coarsest level (i.e., the more detailed element is truncated to the resolution of the less detailed element).	Date/Time: 20180304 is equivalent to 201803040823-0600. Date/Time: 20180305 is later than 201803040823-0600.
be truncated later than LOCATION 2 (DESCRIPTION).	Applies to date/time content only and indicates a date/time	Date/Time: 20180305 is later than 201803040823-0600.

Comparative Statement	Description	Examples
	that occurs after the date/time of the element it is compared to. The comparison is a "meaning" comparison and is at the coarsest level (i.e., the more detailed element is truncated to the resolution of the less detailed element).	

13.8 Complex Conformance Statements

Conformance statements can be combined to make complex conformance statements with a set of connectors (See Table 13.8). Multiple connectors can be used, and connectors can be used in propositions and/or content-declarative statements and comparison-declarative statements.

Table 13.8: Conformance Statement Connectors

Connectors	Description
AND	Both predicates must be satisfied
OR	One predicate must be satisfied; both may be satisfied
XOR	Exclusive OR—one predicate must be satisfied; both must not be satisfied

An example of a complex conformance statement is:

Example:

IF RXA-20 (Completion Status) contains one of the values in the list { 'CP', 'PA' } AND RXA-9.1 (Administration Notes) contains the value '00' THEN exactly one occurrence of OBX-3.1 (Entity Identifier) in the same ORDER Group SHALL contain the value "30963-3" (Funding Source) drawn from the LN code system.

13.9 Free-Text Conformance Statements

The pseudo language presented is not intended to cover all possible constraints a specifier may wish to provide. In situations in which the language does not support a constraint, the specifier may use a free-text expression. The expression is not conducive to automatic translation into a machine-processable assertion.