Check for updates

NIST Special Publication 1500
NIST SP 1500-19

Micro Common Data Format Specification

Version 1.0

John Dziurłaj
Benjamin Long

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.1500-19

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Special Publication 1500
NIST SP 1500-19

Micro Common Data Format Specification

Version 1.0

John Dziurłaj
The Turnout LLC

Benjamin Long
Software and Systems Division
Information Technology Laboratory, NIST

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.1500-19

January 2023

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
Laurie E. Locascio, NIST Director and Under Secretary of Commerce for Standards and Technology

NIST SP 1500-19
January 2023

Publication History
Approved by the NIST Editorial Review Board on 2023-01-23

How to Cite this NIST Technical Series Publication
Dziurłaj J (2023) Micro Common Data Format Specification. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication 1500 (SP) NIST SP 1500-19. https://doi.org/10.6028/NIST.SP.1500-19

NIST Author ORCID iDs
Benjamin Long: 0000-0002-4340-7674

Contact Information

National Institute of Standards and Technology
Attn: Software and Systems Division, Information Technology Laboratory
100 Bureau Drive (Mail Stop 8970) Gaithersburg, MD 20899-8930
Email: voting@nist.gov

Abstract

This specification describes a data format for space-constrained environments, such as the placement of machine readable data on paper. The specification is responsive to a need for interoperability in several key voting system scenarios in which the use of other storage mechanisms are impractical or disallowed. The format's structure is described in prose and code examples.

Keywords

Common data format (CDF); elections; Health Level Seven (HL7); serialization; Unified Markup Language (UML); voting; VVSG.

Table of Contents

List of Tables

List of Figures

Acknowledgments

The authors wish to thank their colleagues of the National Institute of Standards and Technology Common Data Format (CDF) Ballot Styles Subgroup, who reviewed drafts of this document and contributed to its technical content.

The authors also gratefully acknowledge and appreciate the significant contributions from individuals and organizations in the public and private sectors, whose thoughtful and constructive comments improved the overall quality, thoroughness, and usefulness of this publication.

Executive Summary

This publication presents a messaging standard for conveying election data used in specialized scenarios. Such scenarios share characteristics such as limited storage space (e.g., paper), fewer code points that can be used for encoding data, and transmission over airgaps. This format is not intended for general use in environments that do not exhibit these characteristics.

The format seeks to support the following election scenarios:

1. Exchange of activation information between ballot activation devices and ballot marking devices;
2. exchange of contest option selections between ballot marking devices and ballot scanners;
3. exchange of ballot style identifier information between full-face paper ballots and scanners; and
4. other applications that require software independent (e.g., paper) information exchange.

The format, known as the Micro Common Data Format (mCDF) Version 1.0, describes a method of encoding data with less overhead than XML or JSON. This publication describes:

- The syntactic structures of the format;
- logical structures (segments) applicable to all mCDF messages;
- a method to derive mCDF messages from existing UML (Unified Modeling Language) models; and
- background information.

The mCDF is a serialization, not a schema, and this specification does not provide any data structures for a particular data exchange. Instead, profiles of common data formats, detailed outside of this specification will specify the use of mCDF for the scenarios. Separately provided profiles exist for scenarios (2) and (3) listed above.

The specification provides manufacturers of voting systems with standard methods for exchanging data in air-gapped scenarios that often use paper as the medium of exchange, thereby increasing interoperability among election devices that were never suited to formats such as JSON or XML. Interoperable data exchanges for these scenarios seek to increase the componentization of voting equipment and offer jurisdictions more choice in election equipment.

The specification is geared towards the following audiences:

- Voting equipment manufacturers
- Election officials
- Election-affiliated organizations and
- Election analysts and the general public

1. Introduction to microCDF

The Micro Common Data Format (stylized as "microCDF" and hereafter referred to as "mCDF") is a messaging standard for consistent data transfer across storage-constrained election information systems.

It is a textual, delimited, hierarchical messaging format intended to represent profiles of NIST Special Publication Series 1500 common data format (CDF) specification data in a compact manner. It is designed for environments that are storage-space constrained, such as printed materials.

Within the elections space, the following applications are considered:

- Exchange of activation information between ballot activation devices and ballot marking devices;
- exchange of contest option selections between ballot marking devices and ballot scanners;
- exchange of ballot style identifier information between full-face paper ballots and scanners; and
- other applications that require software independent (e.g., paper) information exchange.

In the context of the exchange of contest option selections, mCDF is meant to reconcile the need for interoperable data exchange throughout the election process with the Voluntary Voting System Guidelines 2.0 [1] Principle 9.1:

> *An error or fault in the voting system software or hardware cannot cause an undetectable change in election results.*

Thus, the mCDF is a format that supports the interoperability of software intendent (e.g., paper) vote records.

mCDF is not intended to offer an alternative to the JSON and XML serializations of the CDFs in environments that are not storage-space constrained.

Instead of developing a new serialization, research was conducted on a suitable existing format. A survey of existing formats showed that while none provided the exact features required, Health Level 7 (HL7) v2 [2] could serve as a starting point. mCDF extends and diverges from HL7 v2 in several areas; a full treatment of these differences is given in Appendix C.

This document represents novel work in interoperability in the voting space. It is anticipated that revisions to this specification will be made as vendors and others attempt to implement the standard.

1.1. Design Principles

- Compactness. Data is conveyed using as little overhead as possible.
    - Default values. Where possible, default values can be assumed when field values are omitted.
    - Delimited. Fields are separated by single character delimiters.
    - Ordering. Fields can be ordered so that required and commonly used fields appear first, avoiding some delimiter overhead.
    - Early termination. Segments can terminate without emitting delimiters for unused, trailing fields.
- Shared comprehension
    - Derivability from UML Model. mCDF Profiles are derived from the same data model as other CDFs, that is, the Unified Modeling Language (UML) [3] model.
    - ASCII. mCDF serializes as ASCII text [5], rather than binary. This allows its contents to be intelligible by a variety of readers.
- Flexibility. The mCDF should be able to be represented using existing print symbologies.
    - 7-bit ASCII. The mCDF uses a restricted subset of ASCII that is widely supported.
    - Digital signatures. All mCDF messages can be digitally signed.

mCDF is a serialization, not a schema and this specification is not built for any particular set of data. Instead, the data that may be conveyed using a mCDF is specified through mCDF profiles defined in other specifications (e.g., mCDF Profile for Contest Selection Capture) [4].

This document describes the syntax of the mCDF format and a method for mapping UML models to equivalent mCDF representations.

2. Logical and Physical Syntax

This document refers to the *logical syntax* of a message, and the *physical syntax* required to carry it. A single logical message can have many equivalent physical manifestations. A physical message can be converted to its logical form by canonicalizing it (see Section 4.2) and resolving all escaped characters.

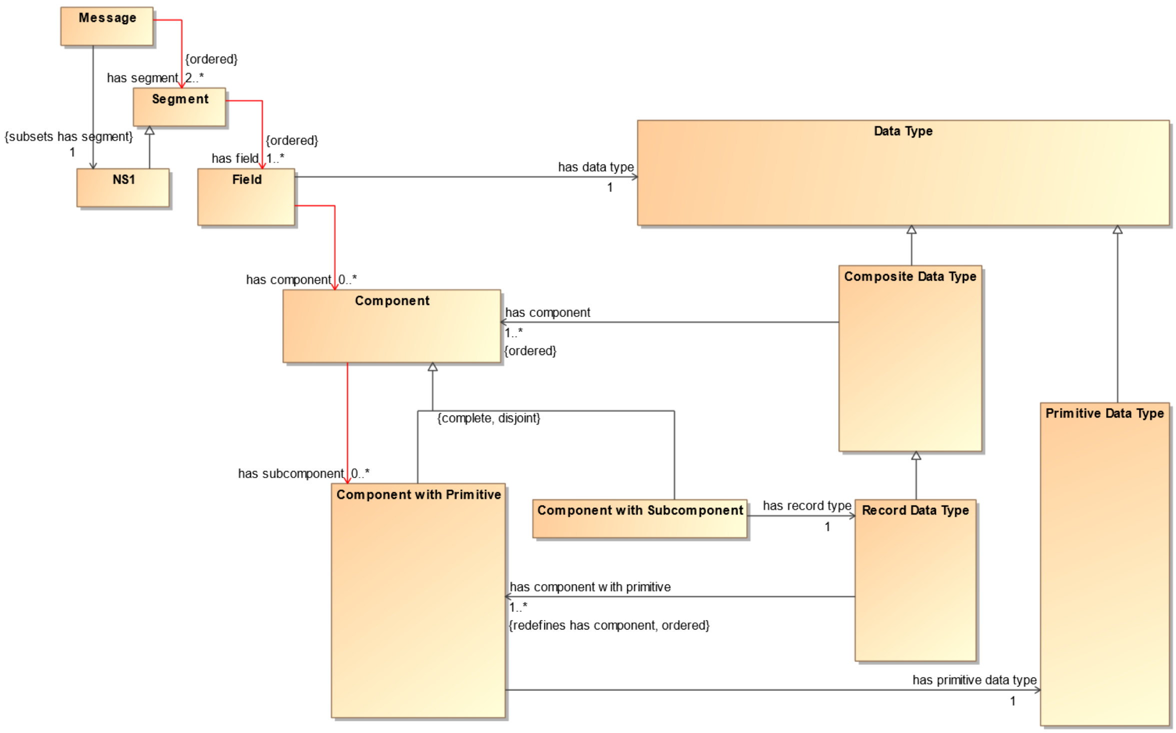

Figure 1 – Diagram of mCDF Conceptual Structures

2.1. Document conventions

This document refers to fields, components, and subcomponents thereof. When referring to fields, the following convention is used. The three digit segment identifier followed by a dash (-), followed by the sequence number is used. For example, NS1-1 refers to the “Field Separator” of the segment NS1.

2.2. Reserved tokens

mCDF is a highly flexible format, natively supporting restrictive character sets. Each mCDF message header specifies the reserved characters that are used to represent its syntactic constructs. One of the key features of the mCDF standard is that delimiters can be swapped out on a per physical message basis. This allows the mCDF to be used even when very few characters are available.

Table 1 lists the conventional delimiters used by mCDF. It is strongly recommended to use these delimiters unless a technical limitation precludes their use.

Table 1 – Conventional delimiters in mCDF

Delimiter	Meaning
\	Escape character
|	Field delimiter
^	Component delimiter
&	Subcomponent delimiter
~	Repetition delimiter
;	Segment delimiter

2.3. Messages

A message is a set of segments (described in Section 2.4) in a particular order that taken together form an information exchange. Each message is assigned a three character code that uniquely identifies it.

Messages are defined by specific common data format profiles and in terms of their logical syntax.

2.4. Segments

A segment is a particular grouping of fields (described in Section 2.5). Segments map to one or more classes in a CDF's UML class model. Each segment starts with its Segment ID, made up of three characters, followed by the *field delimiter* and one or more fields. Segments can be required or optional as defined by the message containing them, and some segments may repeat a number of times. Segments are conventionally delimited by the semicolon ; symbol.

2.5. Fields and Delimiters

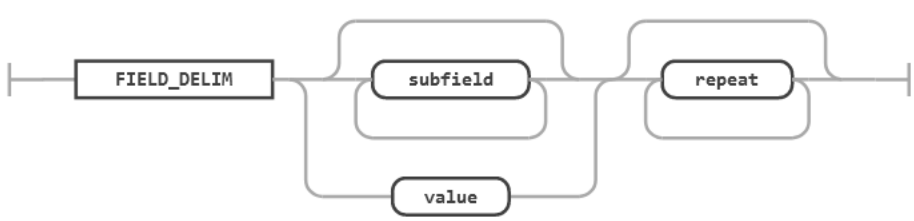

Figure 2 – Fields and delimiters diagram

Fields are the most fundamental concept in mCDF. A field contains data within a value space (defined by its type). Each field in a physical message is delimited, or separated using the tokens defined at the very beginning of the NS1 header segment (See Section 3.1). Fields can be further broken down into components and subcomponents. The delimiter used is dependent on whether the separation is between fields, components or subcomponents.

2.5.1. Field Population

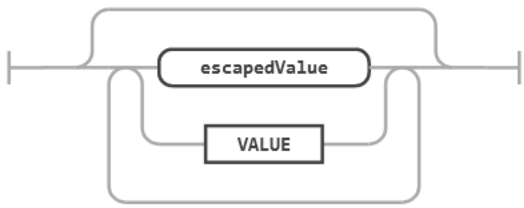

Figure 3 – Field population diagram

A field is said to be populated if there is character data (including spaces) in it. If there is an absence of any character data, then the consuming system may assume a default value applicable to the field.

A segment with a single, unpopulated field

```
SEG|;
```

A segment with a single populated field

```
SEG|TEXT;
```

2.5.2. Field Position

A field appears in a particular position relative to other fields in its segment. A field's location in a segment is indicated by an integer starting at 1 and increasing monotonically by 1. This number is used as shorthand to refer to the data field in mCDF profiles (e.g., NS1-2 to refer to the second field of the NS1 segment).

2.5.3. Data Types

mCDF derives its data types from UML. When a UML primitive data type (e.g., String, Integer) is encountered, character data is expected for the field. When a UML class is encountered, a component or subcomponent is expected. When a UML enumeration is encountered, a mapped value (i.e., an integer standing in for the enumeration literal) is expected.

Further constraints specified in UML (such as particular formats for dates) are expected to be honored in mCDF's physical syntax.

2.5.4. Repetition

Some fields can repeat multiple times within a segment. Repetition is indicated by using the *repetition delimiter*.

Whether a field can repeat is specified by the mCDF profile's documentation. For example, if the upper cardinality of a field is greater than 1, then the field can repeat up to the upper cardinality value. Note that because fields contain composite UML class data structures (e.g., nestable component and subcomponent values), only fields can be repeated using mCDF syntactic operators. Field content values, even when composite, are always treated as contained literals in this specification.

2.6. Supported Encodings

The mCDF is built to support the use of print symbologies that may support a limited character set. Thus, all messages must use a restricted version of 7-bit ASCII (i.e., code points 32-126) or subsets thereof.

2.7. Formatting Codes

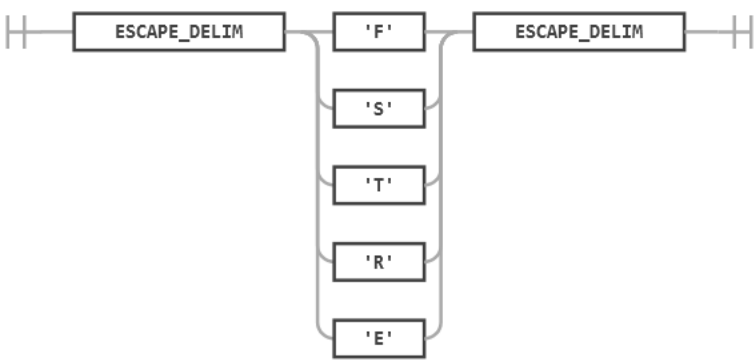

Figure 4 – Formatting codes diagram

mCDF supports the use of formatting codes for when a message's defined delimiters (specified in NS1-1 and NS1-2) must be used as part of a field's character literal or if the character cannot be represented using the physical message's encoding. The *escape delimiter* character is whichever ASCII character is specified in NS1-2 (3rd position). For purposes of this section, the character \ will be used to represent the escape delimiter. An escape sequence consists of the escape delimiter character followed by an escape code ID of one character, zero or more data characters, and another occurrence of the escape character. The following escape sequences are defined:

Table 2 – Supported escape sequences

Escape Sequence	Escape Name
\F\	Field Separator
\S\	Component Separator

\T\	Subcomponent Separator
\R\	Repetition Separator
\E\	Escape Character
\Uxx\	8-bit Unicode Escape
\Uxxyy\	16-bit Unicode Escape
\Uxxyyzz\	24-bit Unicode Escape

Escape sequences SHALL NOT contain nested escape sequences.

2.7.1. Internationalization and Reserved Tokens

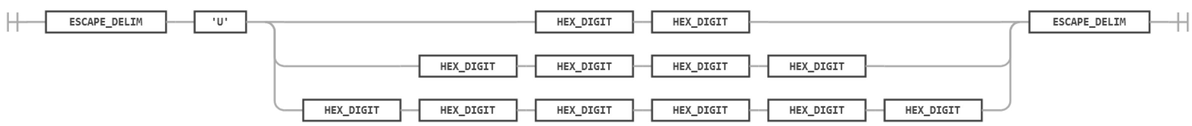

Figure 5 – Internationalization and reserved tokens diagram

mCDF uses a subset of 7-bit ASCII encoding. If there is a requirement to support characters beyond this, they must be escaped. This is facilitated by the use of one of the \Uxxyyzz\ escape sequence for Unicode [6] code points. Table 3 shows names in the Latin alphabet using diacritics (e.g., accents) and logographic characters (e.g., Chinese characters) mapped using mCDF.

Table 3 – Example use of Unicode escape sequences

Logical Character Literal	**Physical Character Literal**
Sebastián Ibáñez	Sebasti\UE1\n Ib\UE1\\UF1\ez
李嘉诚	\U674E\\U5609\\U8BDA\

The escape sequence should reference the code point, not the UTF-* representation. For example, 李 is represented as E6 9D 8E in UTF-8, but the Unicode code point is 67 4E. A separate escape is required per code point.

3. Segments defined by this standard

There are two segments defined by the mCDF standard itself. They include NS1, which serves as the header for all mCDF messages, and DSC, a segment for splitting a single logical message into multiple physical message fragments (see Section 4.1 for additional information).

3.1. NS1 - NIST Segment 1 (Header)

The NS1 header segment is required for each mCDF message and appears before any other segment.

Table 4 – NIST Segment 1 Header

Order	Datatype	Multiplicity	Attr Name	Description
1	String	1	Field Separator	Identifies the character that will separate fields, normally |
2	String	1	Segment Encoding Characters	Maps various delimiters, in order of component separator, repetition separator, escape character, subcomponent separator, and end-of-segment character, normally ^~\&;
3	String	1	Message Type	The three-character identifier for the message.
4	String	1	Message Version	Version of the message. Check the CDF profile for this value
5	String	1	Packet Serial	Each packet must have its own serial number. This is to distinguish it from others during packet reassembly
6	String	0..1	Continuation ID	Only used when fragment reassembly is required. Used to determine which order to reassemble fragments

3.2. DSC – continuation pointer segment

The continuation pointer segment is only used in messages that are fragmented. See Section 4.1.

Table 5 – Continuation pointer segment

Order	Datatype	Multiplicity	Attr Name	Description
1	String	1	Continuation Pointer	For associating a physical message with the next physical message that makes up the larger logical message

4. Additional Features

4.1. Continuations (Fragmentation)

mCDF instances can be broken into fragments. This is useful when a mCDF instance cannot be expressed using a single machine readable encoding. Data in the NS1 header (described in Section 3.1) is used to allow reassembly of instances from fragments and distinguish fragments from other mCDF instances.

The process for creating fragmented instances is given below:

Each fragment must include a NS1 header. A fragment cannot end or start in the middle of a segment (i.e., each segment must be wholly contained within a single fragment). Each fragment (except the final fragment) must contain a DSC segment (described in Section 3.2).

1. The logical message is split after an arbitrary segment.
2. A DSC segment is placed after the split in the first fragment. The DSC-1 Continuation Pointer field will contain a unique value that is used to match a subsequent fragment containing the same value in its header.
3. The DSC terminates the first fragment of the logical message.
4. The subsequent message will contain a NS1-6 Continuation ID, a value that matches that from DSC-1 (The presence of a value in NS1-6 indicates that the message is a fragment of an earlier message). Each subsequent message will have its own unique value for NS1-5 Packet Serial. Coordination between the between the DSC-1 Continuation Pointer and the subsequent message's NS1-6 Continuation ID is used to link the fragments in the proper order.
5. The logical message is the concatenation of the contents of the first message (which while having no value in NS1-6, did end with DSC, and hence was actually a message fragment), plus all subsequent fragments (as identified by values in NS1-6).

4.1.1. Fragment reassembly

Fragments must be reassembled in order. A reassembled instance should not contain interleaving header (NS1) segments. Instead those segments should be removed as redundant.

Fragment 1:

```
NS1|^~\&;|XXX|1|1;{Message specific data 1};DSC|123;
```

Fragment 2:

```
NS1|^~\&;|XXX|1|2|123;{Message specific data 2};
```

Reassembled instance:

```
NS1|^~\&;|XXX|1|1;{Message specific data 1};{Message specific
data 2};
```

Note that the headers are not duplicated in the reassembled message. The consuming application is responsible for converting the physical syntax to its logical equivalent.

4.2. Canonicalization of mCDF

Canonicalization is a method of putting a mCDF into a normal form, so that it can be easily compared for equivalence. This is particularly important for cryptographic applications. The canonicalization algorithm for mCDF is as follows:

1. If mCDF instance is fragmented, reassemble all fragments according to the protocol in the section “Fragment reassembly”.
2. Remap all delimiters to their default values (see Section 2.2)

Note that all escaped characters remain escaped.

5. Usage examples

5.1. Mapping delimiters

There may be situations where it might be beneficial to use delimiters other than the conventional ones. For example, QR Codes[7] are most efficient when operating in "Alphanumeric" mode, but this also limits the delimiters available. The following example shows a mCDF instance using alternative delimiters, as given below:

Table 6 – Mapping delimiters

Conventional	Alternative Alphanumeric Subset	Meaning
\	$	Escape character
|	+	Field delimiter
^	*	Component Delimiter
&	%	Subcomponent delimiter
~	-	Repeat of group
;	.	End of segment

Message using conventional delimiters:

```
NS1|^~\&;|BSI|1|1;BAL|112115|1|1|1-ess;ELE|331332219|26-
37520^1;
```

Message using alternative alphanumeric subset:

```
NS1+*-
$%.+BSI+1+1.BAL+112115+1+1+1$R$ess.ELE+331332219+26$R$37520*1.
```

Note in the above example that the fourth value in the `BAL` segment has changed from `1-ess` to `1$R$ess.` This is because the - character has been remapped in this message to be the reserved delimiter for repetitions and so it must be escaped. We must now use an escape delimiter, which has been mapped in this message to `$`, and the standard character for the repeat delimiter defined in Table 2, which is `R`. Thus, `-` is represented as `$R$`.

References

[1] Voluntary Voting Systems Guidelines, Version 2.0, February 10, 2021, Available at https://www.eac.gov/sites/default/files/TestingCertification/Voluntary_Voting_System_Guidelines_Version_2_0.pdf

[2] Health Level Seven International, Inc. Health Level 7 (HL7) Standard Version 2.7, ANSI/HL7, January, 2011, https://www.hl7.org

[3] Object Management Group (OMG), UML Specification version 1.1 (OMG document ad/97-08-11) September 22, 2011, https://omg.org/ [accessed 02/01/2019].

[4] mCDF Profile for Contest Selection Capture, Available at https://github.com/usnistgov/CastVoteRecords/tree/feature/mCDF/mCDF%20Profile%20for%20Contest%20Selection%20Capture [accessed 12/19/2022].

[5] ANSI INCITS 4-1986[R2017]: "Information Systems - Coded Character Sets - 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)"

[6] The Unicode Consortium. The Unicode Standard, Version 15.0.0, (Mountain View, CA: The Unicode Consortium, 2022. ISBN 978-1-936213-32-0) https://www.unicode.org/versions/Unicode15.0.0/

[7] ISO/IEC (2015) 18004:2015 Information technology — Automatic identification and data capture techniques — QR Code bar code symbology specification (ISO/IEC, Geneva, Switzerland) https://www.iso.org/standard/62021.html

Appendix A. Acronyms

Selected acronyms and abbreviations used in this document are defined below:

ASCII	American Standard Code for Information Interchange
CDF	Common Data Format
CVR	Cast Vote Record
DSC	Continuation Pointer Segment
JSON	JavaScript Object Notation
NS1	NIST Segment 1 Header
UTF	Unicode Transformation Format
XML	Extensible Markup Language

Appendix B. mCDF and CDFs Compared

Table 7 – mCDF and CDFs Compared

Factor	CDF	mCDF
Serialization	JSON, XML	Custom (HL.7 derivative)
Data Model	Hierarchical, Network	Hierarchical, Record Based
Default Values	Not supported	Supported
Enumeration values	String literals	Integer literals

Appendix C. mCDF and HL7 v2 compared

- Both share the concept of segments
- Both share 3 character segments and message IDs
- Fields are sequenced in both
- mCDF does not have fixed length fields
- mCDF does not allow Z segments
- mCDF does not use newlines for segment termination
- mCDF header is much different than a HL7 header
- mCDF only supports character escapes in terms of UTF codepoints
- mCDF supports 7-bit ASCII only

Appendix D. Mapping UML to mCDF Profiles

All published Special Publication Series 1500 common data formats covering elections are based on a high level model developed in the Unified Modeling Language. A benefit of this approach is that multiple implementation formats (e.g., JSON, XML) can be derived from a single logical model. This approach is extended to the mCDF format. Like the transformation to JSON and XML, mCDF requires additional annotations be added to the UML models to describe how the mCDF transformation should occur. These annotations indicate:

- Which classes should map to a mCDF segment
- Which properties of the class should be carried forward into mCDF segments as fields (profiling)
- Which properties should have default values
- Which properties should be made required or optional (overriding the requirements in the UML Model)
- Which classes should be collapsed into their parents

Annotations are applied using UML profiles and stereotypes. A UML profile is a collection of stereotypes and tags that can be applied to UML elements, such as classes, attributes, associations, among others. A UML Profile and a mCDF profile should not be confused.

D.1. UML Profile Stereotypes

Table 8 – UML profile stereotypes

Stereotype	Metaclass	Properties	Description
mCDFDatatype	Element	DataType Name Documentation	Causes a mCDF Data Type with the given Datatype Name to be generated.
mCDFInclude	Property	Default Value Documentation Mappable Order Rename Repeatable Required Subsume	Causes the property to be included as a mCDF segment or component.
mCDFLookup	EnumerationLiteral	Identifier	Specifies a short (normally numeric) identifier to stand in place of the enumeration literal.
mCDFMessage	Element	Composing Classes Documentation Message Name	Causes a message with the given Message Name to be generated. ComposingClasses will be output in order, as segments.
mCDFSegment	Element	Documentation Segment Name	Causes a mCDF Segment with the given Segment Name to be generated. Class properties tagged with «mCDFInclude» will be output as fields.
mCDFTable	Enumeration	Documentation Table Name	Causes a data table to be generated for the enumeration.

D.2. Stereotypes Properties

D.2.1. mCDFDatatype

Table 9 – Stereotypes properties

Property	Type	Multiplicity	Description
Datatype Name	String	”1”, “1”	A three character code that will identify the datatype in mCDF syntax.
Documentation	String	”1”, “1”	mCDF-specific description of the datatype.

D.2.2. mCDFInclude

Table 10 – mCDFInclude

Property	Type	Multiplicity	Description
Default Value	String	”0”, “1”	A default value to assume if no value is provided.
Documentation	String	”1”, “1”	mCDF-specific description of the included property.
Mappable	Boolean	”0”, “1”	Whether the property should map back to the UML model.
Order	Integer	”1”, “1”	The order the property should appear, relative to other properties at the same level.
Rename	String	”0”, “1”	Provides a disambiguating name.
Repeatable	Boolean	”0”, “1”	Whether the property may repeat.
Required	Boolean	”0”, “1”	If set, required value comes from here instead of UML property.
Subsume	Boolean	”1”, “1”	If set, indicates that the target class properties should be directly incorporated into the source class.

D.2.3. mCDFLookup

Table 11 – mCDFLookup

Property	Type	Multiplicity	Description
Identifier	String	”1”, “1”	The identifier used to stand in for the enumeration literal.