**NIST Special Publication 1500-19**

**DRAFT Micro Common Data Format Specification**

**Version 1.0**

John Dziurłaj

Second Author

Etc.

This publication is available free of charge from:

https://doi.org/10.6028/NIST.SP.1500-19



**NIST Special Publication 1500-19**

**Micro Common Data Format**

John Dziurłaj  
*The Turnout LLC*

First Author

Second Author

*Office of XXXXXXXXXX*

*Second Operating Unit*

This publication is available free of charge from:

https://doi.org/10.6028/NIST.SP.1500-19

May 2022



U.S. Department of Commerce

*Gina M. Raimondo, Secretary*

National Institute of Standards and Technology

*James K. Olthoff, Performing the Non-Exclusive Functions and Duties of the Under Secretary of Commerce*

*for Standards and Technology & Director, National Institute of Standards and Technology*

Certain commercial entities, equipment, or materials may be identified in this

document in order to describe an experimental procedure or concept adequately.

Such identification is not intended to imply recommendation or endorsement by the

National Institute of Standards and Technology, nor is it intended to imply that the

entities, materials, or equipment are necessarily the best available for the purpose.

**National Institute of Standards and Technology Special Publication 1500-19**

**Natl. Inst. Stand. Technol. Spec. Publ. XXX, NNN pages (Month YYYY)**

**CODEN: NSPUE2**

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

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.

Key words

Required, alphabetized, separated by semicolon, and end in a period.

Table of Contents

[1. Introduction to microCDF 1](#_Toc115265322)

[2. Logical and Physical Syntax 3](#_Toc115265323)

[2.1. Document conventions 3](#_Toc115265324)

[2.2. Reserved tokens 3](#_Toc115265325)

[2.3. Messages 4](#_Toc115265326)

[2.4. Segments 4](#_Toc115265327)

[2.5. Fields and Delimiters 4](#_Toc115265328)

[2.5.1. Field Population 5](#_Toc115265329)

[2.5.2. Field Position 5](#_Toc115265330)

[2.5.3. Data Types 5](#_Toc115265331)

[2.5.4. Repetition 5](#_Toc115265332)

[2.6. Supported Encodings 6](#_Toc115265333)

[2.7. Formatting Codes 6](#_Toc115265334)

[2.7.1. Internationalization and Reserved Tokens 6](#_Toc115265335)

[3. Segments defined by this standard 8](#_Toc115265336)

[3.1. NS1 (NS1 Header) 8](#_Toc115265337)

[3.2. DSC – continuation pointer segment 8](#_Toc115265338)

[4. Additional Features 9](#_Toc115265339)

[4.1. Continuations (Fragmentation) 9](#_Toc115265340)

[4.1.1. Fragment reassembly 9](#_Toc115265341)

[4.2. Canonicalization of mCDF 10](#_Toc115265342)

[5. Usage examples 11](#_Toc115265343)

[5.1. Mapping delimiters 11](#_Toc115265344)

[References 12](#_Toc115265345)

[Acronyms 13](#_Toc115265346)

[Appendix A: mCDF and CDFs Compared 14](#_Toc115265347)

[Appendix B: mCDF and HL7 v2 compared 15](#_Toc115265348)

[Appendix C: Mapping UML to mCDF Profiles 16](#_Toc115265349)

[5.2. UML Profile 17](#_Toc115265350)

[Appendix D: mCDF Specification Grammar 26](#_Toc115265351)

List of Tables

[Table 1 – Conventional delimiters in mCDF 3](#_Toc114553515)

[Table 2 – Supported escape sequences 6](#_Toc114553516)

[Table 3 – Example use of Unicode escape sequences 7](#_Toc114553517)

[Table 4 – NIST Segment 1 Header 8](#_Toc114553518)

[Table 5 – Continuation pointer segment 8](#_Toc114553519)

List of Figures

[Figure 1 – Railroad track diagram 5](#_Toc114553520)

# 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 information exchange.

In 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 B: mCDF and HL7 v2 compared.

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.

**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 are 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)[] 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)[3].

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

# 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.

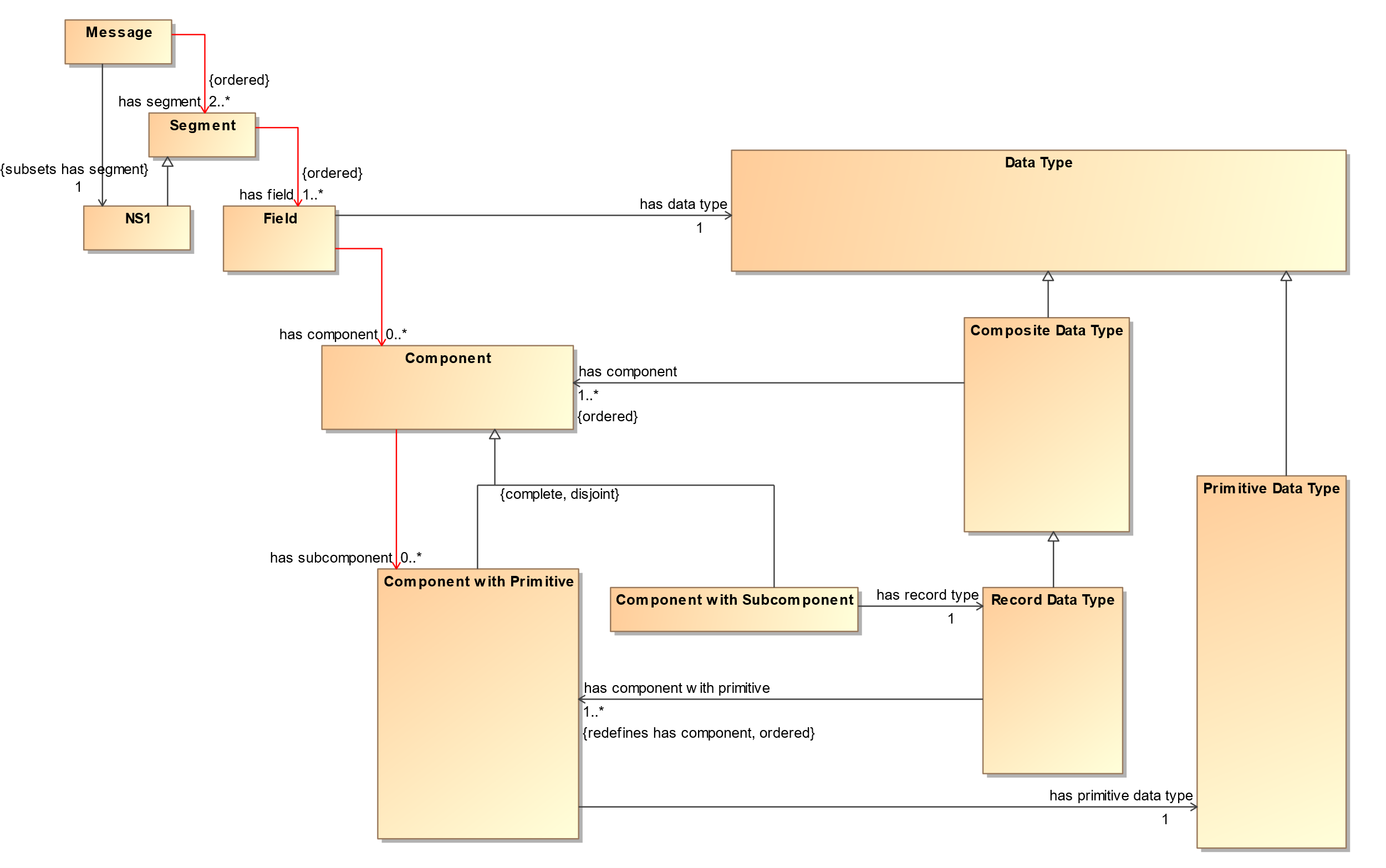


Figure - Diagram of mCDF Conceptual Structures

## 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.

## 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 |

## 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.

## 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.

## Fields and Delimiters

Diagram

Description automatically generated

Figure – Railroad track 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 the whether the separation is between fields, components or subcomponents.

### Field Population

Diagram

Description automatically generated

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 populated field

|  |
| --- |
| SEG|TEXT; |

### 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).

### 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.

### 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 components and subcomponents of a field cannot repeat.

## 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.

## Formatting Codes

Diagram

Description automatically generated

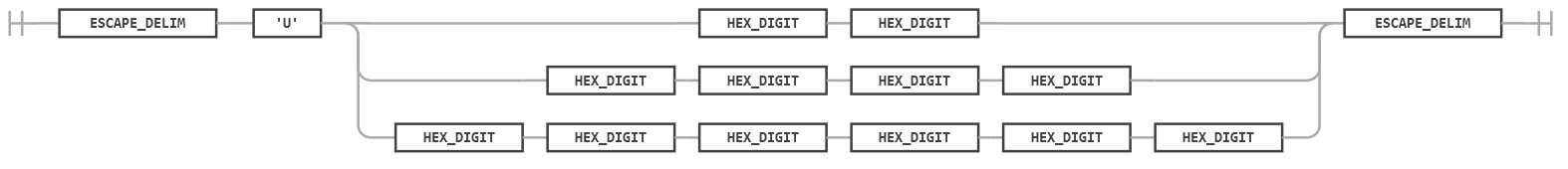
mCDF supports the use of formatting codes for when a message’s defined delimiters (in NS1-1 and NS1-2) must be used as part of a field’s character literal or if the character cannot be represented using the physical message’s encoding. The escape delimiter character is whichever ASCII character is specified 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.

### Internationalization and Reserved Tokens



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. The 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** |
| José Miguel Ramón Adaucto Fernández y Félix | Jos\UE9\ Miguel Ram\UF3\n Adaucto Fern\UE1\ndez y F\UE9\lix |
| 蔡英文 | \U8521\\U82F1\\U6587\ |

The escape sequence should reference the code point, not the UTF-\* representation. For example, 蔡 is represented as E8 94 A1 in UTF-8, but the unicode code point is 85 21. A separate escape is required per code point.

# 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 5.1 for additional information).

## NS1 (NS1 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. |

## DSC – continuation pointer segment

The continuation pointer segment is only used in messages that are fragmented. See Section 5.1.

Table – 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 |

# Additional Features

## 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 4.1) 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 4.2).

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 Message Control ID. 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).

### 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.

## 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 reserved tokens)

Note that all escaped characters remain escaped.

# Usage examples

## 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:

|  |  |  |
| --- | --- | --- |
| 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, <http://www.hl7.org>
3. Object Management Group (OMG), UML Specification version 1.1 (OMG document ad/97-08-11) September 22, 2011, http://omg.org/ [accessed 02/01/2019].
4. [Placeholder for mCDF Profile for Contest Selection Capture]
5. [ASCII Placeholder]
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>

# 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 |
| UTF | Unicode Transformation Format |

# Appendix A: mCDF and CDFs Compared

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

# Appendix B: 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 supports a special trailer segment for metadata and digital signatures
* mCDF only supports character escapes in terms of UTF codepoints
* mCDF supports 7-bit ASCII only

# Appendix C: 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. However, 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.

## UML Profile

**Stereotypes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Stereotype** | **Metaclass** | **Properties** | **Constraints** | **Description** |
| [mCDFDatatype](#_ca8517e8d4cc00e87b23fa921258985b) | Element | Datatype Name  Documentation |  | Causes a mCDF Data Type with the given Datatype Name to be generated. |
| [mCDFInclude](#_3d7ec92d9657c2b9025c48c078f69959) | Property | Default Value  Documentation  Mappable  Order  Rename  Repeatable  Required  Subsume |  | Causes the property to be included into as a mCDF segment or component. |
| [mCDFLookup](#_aa7284c58d8c56faab461cf753615457) | EnumerationLiteral | Identifier |  | Specifies a short (normally numeric) identifier to stand in place of the enumeration literal. |
| [mCDFMessage](#_cd7b68aca2021d8037fa64d725a8bd2f) | Element | ComposingClasses  Documentation  Message Name |  | Causes a message with the given Message Name to be generated. ComposingClasses will be output in order, as segments. |
| [mCDFSegment](#_5c17d9b0454cfd412970d8562a2f9484) | 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](#_cc2fb5c19c3cf8fd17951df67be1c6d5) | Enumeration | Documentation  Table Name |  | Causes a data table to be generated for the enumeration. |

**Stereotypes Properties**

**mCDFDatatype**

|  |  |  |  |
| --- | --- | --- | --- |
| **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 data type. |

**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. |

**mCDFLookup**

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

**mCDFMessage**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **Multiplicity** | **Description** |
| ComposingClasses | Class | ”0”, “\*” | The mCDFSegments that make up the message. |
| Documentation | String | ”1”, “1” | mCDF specific description of the message. |
| Message Name | String | ”1”, “1” | A three character code that will identify the message in mCDF syntax. |

**mCDFSegment**

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

**mCDFTable**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Type** | **Multiplicity** | **Description** |
| Documentation | String | ”1”, “1” | mCDF specific description of the table. |
| Table Name | String | ”1”, “1” | A three digit code that will identify the table in mCDF syntax. |