INTERNATIONAL STANDARD

ISO/IEC 15475-3

First edition
2002-11-01

Information technology — CDIF transfer format —

Part 3: Encoding ENCODING.1

Technologies de l'information — Format de transfert CDIF —

Partie 3: Codage ENCODING.1

Reference number
ISO/IEC 15475-3:2002(E)

Contents

Table of Illustrations

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 15475 may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 15475-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and system engineering*.

ISO/IEC 15475 consists of the following parts, under the general title *Information technology — CDIF transfer format* :

— *Part 1: General rules for syntaxes and encodings*

— *Part 2: Syntax SYNTAX.1*

— *Part 3: Encoding ENCODING.1*

Annex A forms a normative part of this part of ISO/IEC 15475. Annex B is for information only.

Introduction

This standard will assist the vendors and users of modelling tools and metadata repositories in developing mechanisms for interchanging information. This standard specifies an element of a family of related standards. When used together, these standards specify a mechanism for transferring information between tools.

ISO/IEC 15474-1:2002, *Information technology — CDIF framework — Part 1: Overview* and ISO/IEC 15474-2:2002, *Information technology — CDIF framework — Part 2: Modelling and extensibility* should be read first when initially exploring CDIF. The first explains the overall CDIF Architecture and how the family of standards fits together. The second explains the scope, and modelling approach in CDIF. The CDIF meta-metamodel and extensibility mechanisms are also defined in that document.

This standard defines an encoding of SYNTAX.1. ISO/IEC 15475-1:2002, *Information technology — CDIF transfer format — Part 1: General rules for syntaxes and encodings* describes how CDIF supports multiple syntaxes and encodings. ISO/IEC 15475-2:2002, *Information technology — CDIF transfer format — Part 2: Syntax SYNTAX.1* defines the initial CDIF transfer format syntax, SYNTAX.1.

This standard has been developed with the wide support and participation of vendors, users, academia and government involved in or familiar with the CASE industry, its products and the general requirements associated with interchanging information between these products.

This document is organized into the following Clauses:

— Clause 6: Concepts and facilities

  This specifies the unique identifier for the encoding defined in this document and describes the concepts of clear text and character sets used in this document.

— Clause 7: Encoding structures and keywords in the CDIF transfer

  This Clause describes and gives examples of the format of transfer syntax terminals and lists the encodings of keywords.

— Annex A (normative): ENCODING.1 formal grammar

  This annex defines the encoding rules for all non-terminals in the encoding.

— Annex B (informative): Multibyte examples

  Supplementary Examples using multibyte codeset encoding are provided.

Information technology — CDIF transfer format —

Part 3:
Encoding ENCODING.1

1 Scope

The CDIF family of standards is primarily designed to be used as a description of a mechanism for transferring information between modelling tools. It facilitates a successful transfer when the authors of the importing and exporting tools have nothing in common except an agreement to conform to CDIF. The language that is defined for the transfer format also has applicability as a general language for Import/Export from repositories. The CDIF semantic metamodel defined for modelling tools also has applicability as the basis of standard definitions for use in repositories.

The standards, which form the complete family of CDIF Standards, are documented in ISO/IEC 15474-1:2002, *Information technology — CDIF framework — Part 1: Overview.* These standards cover the overall framework, the transfer format and the CDIF semantic metamodel.

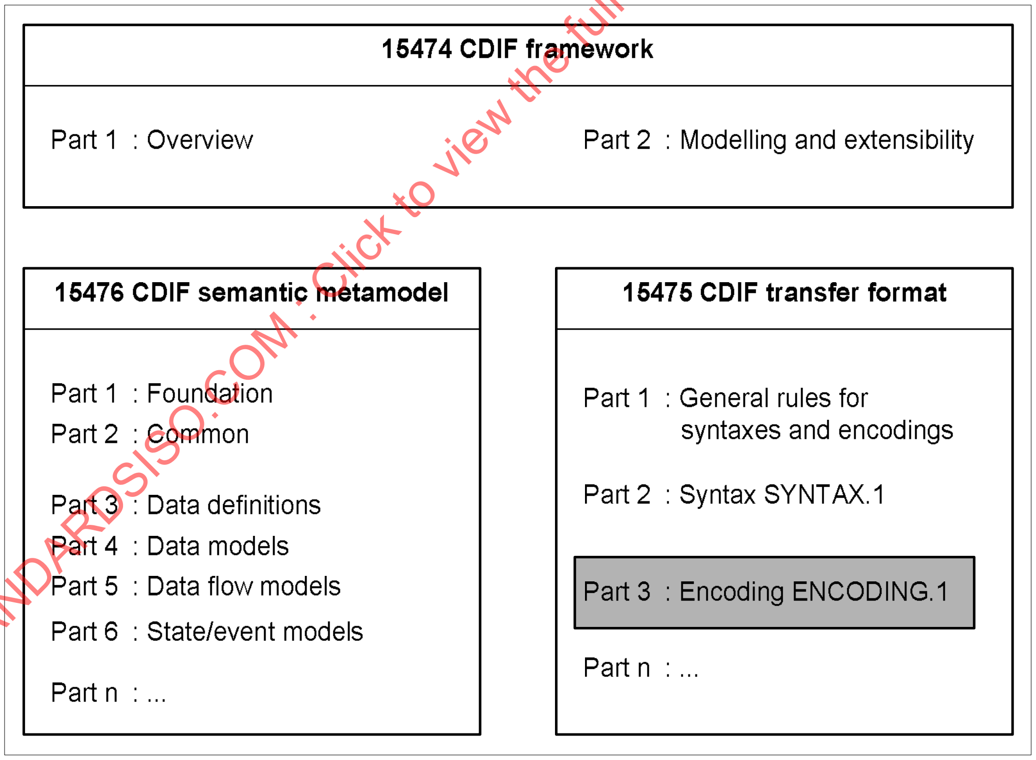
15474 CDIF framework
Part 1 : Overview
Part 2 : Modelling and extensibility
15476 CDIF semantic metamodel
Part 1 : Foundation
Part 2 : Common
Part 3 : Data definitions
Part 4 : Data models
Part 5 : Data flow models
Part 6 : State/event models
Part n : ...
15475 CDIF transfer format
Part 1 : General rules for syntaxes and encodings
Part 2 : Syntax SYNTAX.1
Part 3 : Encoding ENCODING.1
Part n : ...

Figure 1 – Position in the CDIF family of standards

The diagram in Figure 1 depicts the various standards that comprise the CDIF family of standards. The shaded box depicts this Standard and its position in the CDIF family of standards.

This document describes the standard CDIF Transfer Encoding for the standard CDIF Transfer Syntax as defined in ISO/IEC 15475-2:2002, *Information technology — CDIF transfer format — Part 2: Syntax SYNTAX.1*.

This document is intended to be used by anyone wishing to understand and/or use CDIF. This document provides an introduction to the entire CDIF family of standards. It is suitable for:

— Those evaluating CDIF,

— Those who wish to understand the principles and concepts of a CDIF transfer, and

— Those developing importers and exporters.

The documents ISO/IEC 15474-1:2002, *Information technology — CDIF framework — Part 1: Overview* and ISO/IEC 15474-2:2002, *Information technology — CDIF framework — Part 2: Modelling and extensibility* should be read first when initially exploring CDIF and before attempting to read other documents in the CDIF family of standards.

This document should be read in conjunction with ISO/IEC 15475-1:2002, *Information technology — CDIF transfer format — Part 1: General rules for syntaxes and encodings* and ISO/IEC 15475-2:2002, *Information technology — CDIF transfer format — Part 2: Syntax SYNTAX.1.*

While there are no specific prerequisites for reading this document, it will be helpful for the reader to have familiarity with the following:

— Entity-Relationship-Attribute modelling;

— Modelling (CASE) tools;

— Information repositories;

— Data dictionaries;

— Multiple meta-layer modelling;

— Formal grammars;

— Transfer formats.

2 Conformance

A product is standards conformant this standard if and only if the product obeys all definitions and rules in Annex A of this standard, and is also CDIF architecture conformant, as defined in Clause 2 of ISO/IEC 15474-1:2002, *Information technology — CDIF framework — Part 1: Overview*. A product can be either transfer format standards conformant or non-conformant. Partial standards conformance to a standard defining a part of the CDIF transfer format is not defined.

A product is standards conformant to a CDIF encoding standard only if it is standards conformant to Annex A of ISO/IEC 15475-3:2002, *Information technology — CDIF transfer format — Part 3: Encoding ENCODING.1* and also conformant to Annex A of ISO/IEC 15475-2:2002, *Information technology — CDIF transfer format — Part 2: Syntax SYNTAX.1*. A product is standard conformant to a CDIF syntax standard only if it is standards conformant to Annex A of ISO/IEC 15475-2:2002, *Information technology — CDIF transfer format — Part 2: Syntax SYNTAX.1*.

3 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 15475. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 15475 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 9075:1992, *Information technology — Database languages — SQL*

ISO/IEC 10646-1:1993, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*

ISO/IEC 10646-1:1993/Amd.2:1996, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane. Amendment 2: UCS Transformation Format 8 (UTF-8)*

ISO/IEC 13238-1:—[1], *Information technology — Data management export/import — Part 1: Standardization framework*

ISO/IEC 15474-1:2002, *Information technology — CDIF framework — Part 1: Overview*

ISO/IEC 15474-2:2002, *Information technology — CDIF framework — Part 2: Modelling and extensibility*

ISO/IEC 15475-1:2002, *Information technology — CDIF transfer format — Part 1: General rules for syntaxes and encodings*

ISO/IEC 15475-2:2002, *Information technology — CDIF transfer format — Part 2: Syntax SYNTAX.1*

4 Terms and definitions

For the purposes of this part of ISO/IEC 15475, the following definitions apply. Unless otherwise noted, the definitions are specific to this part of ISO/IEC 15475.

4.1 From other standards

4.1.1 ISO/IEC 15474-1

This part of ISO/IEC 15475 makes use of the following terms defined in ISO/IEC 15474-1:

CDIF
CDIF family of standards
CDIF graphical notation
CDIF semantic metamodel
CDIF meta-metamodel
CDIF transfer
CDIF transfer format
Character set
Encoding
ENCODING.1
Meta-attribute
Meta-entity
Metamodel
Meta-object
Meta-relationship
Model
Non-terminal symbol
Production rule
Subject area
Syntax
SYNTAX.1
Terminal symbol

[1] To be published.

Transfer
Transfer format

4.1.2 ISO/IEC 15475-1

This part of ISO/IEC 15475 makes use of the following terms defined in ISO/IEC 15475-1:

Metamodel section
Transfer header

4.1.3 ISO/IEC 13238-1

This part of ISO/IEC 15474 makes use of the following terms from ISO/IEC 13238-1:

Transfer file
CDIF transfer file
Export process
Exporter
Import process
Importer
Clear text file encoding

4.2 For this standard

For the purposes of this part of ISO/IEC 15475 new terms are defined when introduced. Double quotes are used to introduce new terms (e.g., "model layer").

5 Symbols (and abbreviated terms)

5.1 Naming and diagramming notations

All meta-objects and meta-meta-objects in CDIF (in metamodels and meta-metamodels) are named by concatenating all the words that name the meta-object or meta-meta-object; the first letter of each word is upper-case, the rest are lower-case (e.g., *MetaAttribute*, *AttributeDerivation*, *IsDrawnUsing*, *IsOptional*).

Full details of the CDIF Graphical notation used in the metamodel and the meta-metamodel can be found in the Framework document (ISO/IEC 15474-2:2002, *Information technology — CDIF framework — Part 2: Modelling and extensibility*).

5.2 BNF conventions

An extended Backus Naur Form (BNF) is used to define the structure and describe the sequence of data in the transfer file.

This document uses the conventions established in subclause 5.2 of ISO/IEC 15475-1:2002, *Information technology — CDIF transfer format — Part 1: General rules for syntaxes and encodings*.

5.3 Abbreviations

The following abbreviations are used in this part of ISO/IEC 15475:

BNF	Backus Naur Form
CDIF	CASE Data Interchange Format (originally)

6 CDIF transfer concepts and facilities

6.1 Encoding identifier

The encoding defined in this version of this standard shall have a CDIF standardized encoding identifier of "ENCODING.1" and a version of "15475-3:2002". This is used to define the standardized encoding used in a CDIF transfer that conforms to the CDIF standardized syntax with identifier "SYNTAX.1" and version "15475-2:2002" (see ISO/IEC 15475-2).

The syntax of the CDIF Encoding ID and Encoding Version is as follows:

```
<EncodingID> ::=          'ENCODING.1'
<EncodingVersion> ::=     '15475-3:2002'
```

6.2 Clear text

The objectives of this clear text encoding are that it should be:

— Machine writeable: A transfer file using such an encoding should be capable of being generated by computer software.

— Machine-readable: A transfer file using such an encoding should be capable of being parsed by computer software.

— Human readable: A transfer file using such an encoding should be capable of being read and understood by a human who is familiar with the concepts being transferred.

CDIF transfer files are not designed to be created or edited by humans, although it is not difficult to do so.

6.3 Character sets

Whitespace characters are used (outside strings, text strings and comments) to separate tokens and to assist in formatting the transfer to improve its legibility. After token identification, these characters are ignored by importers. The whitespace characters are space, horizontal tab, vertical tab, carriage return, line feed and form feed.

Whitespace characters and all other characters in a transfer file are encoded using the character set specified in the Transfer Header.

Whitespace characters must not appear anywhere other than as defined in the syntax defined in ISO/IEC 15475-2. This does not mean that a text string cannot have non-printable characters. For more details, see 7.2.11.

7 Encoding structures and keywords in the CDIF transfer

7.1 Introduction

A Transfer expressed in this clear text encoding consists of a stream of characters forming a sequence of tokens separated by whitespace. The rules for separation of tokens are given in ISO/IEC 15475-2.

7.2 Encoding structures

7.2.1 Decimal integer value

A decimal integer value is defined as follows:

```
<DecimalIntegerValue> ::= <DecimalRadix> <Integer>
<DecimalRadix> ::=        #D
<Integer> ::=             [ <Sign> ] <Digit> ...
```

The following are examples of valid and invalid decimal integer values:

```
(1)   #d12345        -    Valid: positive decimal
(2)   #d-12345       -    Valid: negative decimal
(3)   #d+12345       -    Valid: positive decimal
(4)   #d123e43       -    Invalid: 'e' is not a numeric digit
```

7.2.2 Binary value

A binary value is defined as follows:

```
<BinaryValue> ::=         <BinaryRadix> <BinaryNumber>
<BinaryRadix> ::=         #B
<BinaryNumber> ::=        [ <Sign> ] <BinaryDigit> ...
<BinaryDigit> ::=         0 | 1
```

The following are examples of valid and invalid binary values:

```
(1)   #b10101        -    Valid: positive binary
(2)   #b-10101       -    Valid: negative binary
(3)   #b11021        -    Invalid: '2' is not a binary digit
```

7.2.3 Octal value

An octal value is defined as follows:

```
<OctalValue> ::=          <OctalRadix> <OctalNumber>
<OctalRadix> ::=          #O
<OctalNumber> ::=         [ <Sign> ] <OctalDigit> ...
<OctalDigit> ::=          0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
```

The following are examples of valid and invalid octal values:

```
(1)   #o31727        -    Valid: positive octal
(2)   #o-31727       -    Valid: negative octal
(3)   #o31923        -    Invalid: '9' is not an octal digit
```

7.2.4 Hexadecimal value

A hexadecimal value is defined as follows:

```
<HexaDecimalValue> ::=      <HexRadix> <HexNumber>
<HexRadix> ::=              #H
<HexNumber> ::=             [ <Sign> ] <HexDigit> ...
<HexDigit> ::=              0 | 1 | 2 | 3 |4 | 5 | 6 | 7 | 8 | 9
                            | A | B | C | D | E | F
                            | a | b | c | d | e | f
```

The following are examples of valid and invalid hexadecimal values:

```
(1)    #h1e2cf          -    Valid: positive hexadecimal
(2)    #hffffffff       -    Valid: positive hexadecimal
(3)    #h-fffffff       -    Valid: negative hexadecimal
(4)    #h122g9          -    Invalid: 'g' not a valid hex digit
```

7.2.5 Float value

A float value is a decimal number with up to 16 decimal digits of precision within the range of 10^{-1023} to 10^{1023}.

A float value is defined as follows:

```
<FloatValue> ::=            <FloatDesignator> <Mantissa> <Exponent>
<FloatDesignator> ::=       #F
<Mantissa> ::=              [ <Sign> ] <Digit> ...
                            [ <DecimalPoint> [ <Digit> ... ] ]
<DecimalPoint> ::=          .
<Exponent> ::=              <Exp> <Integer>
<Exp> ::=                   E
```

Examples of valid and invalid Float values are:

```
(1)    #f4E9            -    Valid
(2)    #f123.45E2       -    Valid
(3)    #f-123.45E15     -    Valid
(4)    #f+0.23E5        -    Valid
(5)    #f0.23E-3        -    Valid
(6)    #f0.23E+3        -    Valid
(7)    #f123.45         -    Invalid: missing exponent
(8)    #f.23E3          -    Invalid: must follow radix with a sign or number
(9)    #f23.32-2        -    Invalid: missing 'E' exponent prefix
```

7.2.6 Date

A date is defined as follows:

```
<Date> ::=                  <YearValue> <DateDelimiter>
                            <MonthValue> <DateDelimiter>
                            <DayValue>
<YearValue> ::=             <Integer>
```

```
<MonthValue> ::=           <Integer>
<DayValue> ::=             <Integer>
<DateDelimiter> ::=        /
```

The following are examples of valid and invalid dates:

```
(1)   1940/12/07          -    Valid
(2)   0002/10/11          -    Valid
(3)   1941/Dec/12         -    Invalid: month must be an integer
(4)   1991/11/31          -    Invalid: day out of valid range for month.
```

7.2.7 Time

A time is defined as follows:

```
<Time> ::=                 <Hours> <TimeDelimiter>
                           <Minutes> <TimeDelimiter>
                           <Seconds>
<Hours> ::=                <Integer>
<Minutes> ::=              <Integer>
<Seconds> ::=              <Integer>
                           [ <DecimalPoint> <Digit>
                           [<Digit> [ <Digit> ] ] ]
<TimeDelimiter> ::=        :
```

The following are examples of valid and invalid times:

```
(1)   07:20:23            -    Valid
(2)   03:59:59            -    Valid
(3)   00:00:00.250        -    Valid
(4)   29:25:19            -    Invalid: illegal hour value 29
```

7.2.8 Identifier value

The syntax of the identifier value is:

```
<IdentifierValue> ::=      * <MultibyteIdentifier> *
                           !! <MultibyteIdentifier> must be a member of the list of
                           values given in the definition of the Meta-attribute.
```

The following are examples of valid identifier values:

```
(1)   *johnBrownsBody*    -    Valid
(2)   *sysrange*          -    Valid
(3)   *CUSTOMER-RECORD*   -    Invalid: embedded hyphen
(4)   *9system*           -    Valid
```

7.2.9 Enumerated value

The syntax of the enumerated value is:

```
<EnumeratedValue> ::=     < <MultibyteIdentifier> >
                          !! <MultibyteIdentifier> must be a member of the list of
                          values given in the definition of the Meta-attribute.
```

The following are examples of valid enumerated values:

```
(1)    <red>              -    Valid
(2)    <10>               -    Valid
```

7.2.10 String

A string is defined as follows:

```
<String> ::=              <StringDelimiter> [ <StringElement> ] ...
                          <StringDelimiter>
<StringDelimiter> ::=     "
<StringElement> ::=       <GeneralPrintableChar>
                          | ]
                          | |
                          | #
                          | <EscapeCharacter> <StringDelimiter>
                          | <EscapeCharacter> <EscapeCharacter>
```

If it is necessary to include a pattern in a string that matches the `<StringDelimiter>`, then this must be "escaped" by preceding each occurrence of the pattern by the `<EscapeCharacter>`. In order to embed a pattern that is the same as the `<EscapeCharacter>`, the pattern must be preceded by the `<EscapeCharacter>`. **]**, |, and **#** are explicitly listed as `<StringElement>`s because they are not `<GeneralPrintableChar>`s. `<GeneralPrintableChar>` does not include characters like **"**, **]**, |, and **#** because they are used as a delimiter (or part of a delimiter) in strings, text strings, and comments.

The following are examples of valid and invalid strings:

```
(1)    "This is a string"        -    Valid
(2)    "This is a "string"       -    Invalid: embedded delimiter not escaped.
(3)    "This is a \"string\""    -    Valid: embedded delimiters escaped.
(4)    "Here is an escape character \\ (backslash) in a string"
                              -    Valid
```

7.2.11 Text string

A text string represents a sequence of printable characters, which may include embedded "whitespace" characters. It is defined as follows:

```
<TextString> ::=          <OpenText>
                               [ <TextElement> ] ... [ ] ]
                          <CloseText>
<OpenText> ::=            #[
<TextElement> ::=         <GeneralPrintableChar>
                          | "
```

```
                                  | |
                                  | #
                                  | <WhiteSpace>
                                  | <EscapeCharacter> <CloseText>
                                  | <EscapeCharacter> <EscapeCharacter>
                                  | ] <TextElementExcludingHashSign>
                                  | <OtherChar>
<TextElementExcludingHashSign> ::=
                                  <GeneralPrintableChar>
                                  | "
                                  | |
                                  | <WhiteSpace>
                                  | <EscapeCharacter> <CloseText>
                                  | <EscapeCharacter> <EscapeCharacter>
                                  | ] <TextElementExcludingHashSign>
                                  | <OtherChar>
<OtherChar> ::=                   <EscapeCharacter> <HexaDecimalValue>
                                  <EscapeCharacter>
<CloseText> ::=                   ]#
```

The maximum size of a <TextString> is 1024 characters between the <OpenText> and <CloseText> delimiters.

If it is necessary to include a pattern in the text string that matches the <CloseText> delimiter, then this must be "escaped" by preceding each occurrence of the pattern by the <EscapeCharacter>. In order to embed a pattern that is the same as the <EscapeCharacter>, the pattern must be preceded by the <EscapeCharacter>. ", |, and # are explicitly listed as <TextElement>s because they are not <GeneralPrintableChar>s. <GeneralPrintableChar> does not include characters like ",], |, and # because they are used as a delimiter (or part of a delimiter) in strings, text strings, and comments.

In order to embed a character that is "not printable" (it is not a <GeneralPrintableChar> , ", #,], <EscapeCharacter> or <WhiteSpace>), use <OtherChar>. <OtherChar> encodes a character as an escaped sequence with an embedded hexadecimal number. This hexadecimal number is the hexadecimal equivalent of the binary number used to represent the character being transferred.

The following are examples of valid and invalid text strings[2]:

(1)
```
#[Program SumIntegers(Input,Output);
var  total,input_integer : Integer;
begin
     while not EOF(Input) do
          begin
          ReadLn(input_integer);
          total:=total+input_integer
          end;
```

2) These examples of the text string subclause have been laid out with whitespace characters, such as <CR>, transformed for presentation.

```
            WriteLn('Total =',total);
                                end.]#
                                -     Valid
(2)   #This is some text]#
                                -     Invalid: incomplete text delimiter
(3)   #[The end of text marker is "]#"]#
                                -     Invalid: ]# not escaped
(4)   #[Here is how to embed a \]# in the text]#
                                -     Valid
(5)   #[And here is how to embed two backslashes \\\\ in the text]#
                                -     Valid
(6)   #[This is some text] -     Invalid: missing # at the end
(7)   #[Control-Z is usually the character \#H1A\, which is not printable.]#
                                -     Valid
(8)   #[There is a right square bracket, ], embedded in this text.]#
                                -     Valid
(9)   #[There is a right square bracket, \], embedded (and escaped) in this text.]# --
                                      Invalid: \] not followed by #
```

7.2.12 Comment

A comment is defined as follows:

```
<Comment> ::=              <OpenComment> <CommentBody> <CloseComment>
<OpenComment> ::=          #|
<CommentBody> ::=          [ <CommentBodyChar> ] ... [ | ]
<CommentBodyChar> ::=      <WhiteSpace>
                           | <GeneralPrintableChar>
                           | "
                           | ]
                           | #
                           | <EscapeCharacter> <CloseComment>
                           | <EscapeCharacter> <EscapeCharacter>
                           | | <CommentBodyCharExcludingHashSign>
<CommentBodyCharExcludingHashSign> ::=
                           <WhiteSpace>
                           | <GeneralPrintableChar>
                           | "
                           | ]
                           | <EscapeCharacter> <CloseComment>
                           | <EscapeCharacter> <EscapeCharacter>
                           | | <CommentBodyCharExcludingHashSign>
<CloseComment> ::=         |#
```

`"`, `]`, and `#` are explicitly listed as `<CommentBodyChar>`s because they are not `<GeneralPrintableChar>`s. `<GeneralPrintableChar>` does not include characters like `"`, `]`, `|`, and `#` because they are used as a delimiter (or part of a delimiter) in strings, text strings, and comments.

The following are examples of valid and invalid comments, represented in the standard CDIF encoding[3]:

```
(1)   #| this is
                      a multi-line comment |#
                   -     Valid
(2)   (#d1 #d2 #| buckle my shoe |# #d3 #d4)
                   -     Valid (in line comment)
(3)   #| hello ... <eof>  -     Invalid (missing delimiter)
```

7.2.13 Meta-meta-object name

A meta-meta-object name is defined as follows:

```
<MetaMetaObjectName> ::=  <UpperCaseAlphabeticChar>
                          [ <UpperOrLowerCaseAlphabeticChar> ] ...
```

The following are examples of valid and invalid meta-meta-object names:

```
(1)   CollectableMetaObject-     Valid
(2)   Meta-Entity          -     Invalid (- is not an alphabetic character)
(3)   metaattribute        -     Invalid (does not start with a capital)
```

7.2.14 Meta-object name

A meta-object name is defined as follows:

```
<MetaObjectName> ::=      <MultibyteIdentifier>
```

The following are examples of valid and invalid meta-object names:

```
(1)   DataModel            -     Valid
(2)   Entity               -     Valid
(3)   DataModelAttribute   -     Valid
(4)   Security-Classification
                           -     Invalid (embedded hyphen)
(5)   role                 -     Invalid (does not start with a capital)
```

7.2.15 Identifier

An identifier is defined as follows:

```
<Identifier> ::=          { <UpperOrLowerCaseAlphabeticChar>| <Digit> }
                          [ <UpperOrLowerCaseAlphabeticChar>
                          | <Digit> | <UnderScore> | <Hyphen> ] ...
```

Examples of valid and invalid identifiers:

```
(1)   johnBrownsBody       -     Valid
```

3) These examples of the comment subclause have been laid out with whitespace characters, such as <CR>, transformed for presentation. <eof> is used to represent the end of the file.

```
(2)   custRec.employeeBadgeId
                               -    Invalid: embedded dot
(3)   sysrange                 -    Valid
(4)   CUSTOMER-RECORD          -    Valid
(5)   9system                  -    Valid
(6)   John Brown               -    Invalid: embedded space
(7)   EMPLOYEE_RECORD          -    Valid
```

7.2.16 Multibyte identifier

A multibyte identifier is defined as follows:

```
<MultibyteIdentifier> ::= <MultibyteIdentifierChar> ...
<MultibyteIdentifierChar> ::=  !! Printable characters defined in the character set
                          specified by the Transfer Header, excluding '\','"','[', ']',
                          '|', '#', '<', '>', '*', ':', '+', '-', '(', ')', ',', '.',
                          and space.
```

7.3 Keywords

A keyword is encoded as a sequence of alphabetic, numeric, underscore ('_'), hyphen ('**-**') and colon ('**:**') characters using the character set specified by the transfer header. Alphabetic characters in keywords are case insensitive. The following are the encodings of keywords:

```
<BitmapKeyword> ::=                     :BITMAP
<DateKeyword> ::=                       :DATE
<ExtendMetaAttributeKeyword> ::=        :EXTENDMETA-ATTRIBUTE
<HeaderKeyword> ::=                     :HEADER
<HeightKeyword> ::=                     :HEIGHT
<IntegerListKeyword> ::=                :INTEGERLIST
<MetaModelKeyword> ::=                  :META-MODEL
<ModelKeyword> ::=                      :MODEL
<PointKeyword> ::=                      :POINT
<PointListKeyword> ::=                  :POINTLIST
<SubjectAreaReferenceKeyword> ::=       :SUBJECTAREAREFERENCE
<SummaryKeyword> ::=                    :SUMMARY
<TimeKeyword> ::=                       :TIME
<VersionNumberKeyword> ::=              :VERSIONNUMBER
<WidthKeyword> ::=                      :WIDTH
```

Annex A
(normative)

ENCODING.1 formal grammar

This annex specifies the encoding production rules. Some of the production rules were presented in 7 Encoding structures and keywords in the CDIF Transfer. This BNF alone is not a complete definition of ENCODING.1. There are additional definitions and restrictions expressed in 6 CDIF Transfer concepts and facilities and in the text in 7 Encoding structures and keywords in the CDIF Transfer.

The case of alphabetic characters in the encodings of <PrintableChar>, <UpperCaseAlphabeticChar> and <UpperOrLowerCaseAlphabeticChar> is significant, otherwise case is not significant.

The following are the encoding rules:

```
<Absolute> ::=               ABSOLUTE
<AbsoluteUTC> ::=            ABSOLUTEUTC
<AbsoluteLocal> ::=          ABSOLUTELOCAL
<BinaryDigit> ::=            0 | 1
<BinaryNumber> ::=           [ <Sign> ] <BinaryDigit> ...
<BinaryRadix> ::=            #B
<BinaryValue> ::=            <BinaryRadix> <BinaryNumber>
<BitmapKeyword> ::=          :BITMAP
<CloseComment> ::=           |#
<CloseScope> ::=             )
<CloseText> ::=              ]#
<Comment> ::=                <OpenComment> <CommentBody> <CloseComment>
<CommentBody> ::=            [ <CommentBodyChar> ] ... [ | ]
<CommentBodyChar> ::=        <WhiteSpace>
                             | <GeneralPrintableChar>
                             | "
                             | ]
                             | #
                             | <EscapeCharacter> <CloseComment>
                             | <EscapeCharacter> <EscapeCharacter>
                             | | <CommentBodyCharExcludingHashSign>
<CommentBodyCharExcludingHashSign> ::=
                             <WhiteSpace>
                             | <GeneralPrintableChar>
                             | "
                             | ]
                             | <EscapeCharacter> <CloseComment>
                             | <EscapeCharacter> <EscapeCharacter>
                             | | <CommentBodyCharExcludingHashSign>
```

```
<Date> ::=                 <YearValue> <DateDelimiter>
                           <MonthValue> <DateDelimiter>
                           <DayValue>
<DateKeyword> ::=          :DATE
<DateDelimiter> ::=        /
<DayValue> ::=             <Integer>
<DecimalIntegerValue> ::= <DecimalRadix> <Integer>
<DecimalPoint> ::=         .
<DecimalRadix> ::=         #D
<Digit> ::=                 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<Dot> ::=                  .
<EnumeratedSeparator> ::= ,
<EnumeratedValue> ::=      < <MultibyteIdentifier> >
<EscapeCharacter> ::=      \
<Exp> ::=                  E
<Exponent> ::=             <Exp> <Integer>
<ExtendMetaAttributeKeyword> ::=
                           :EXTENDMETA-ATTRIBUTE
<FalseValue> ::=           -FALSE-
<FloatDesignator> ::=      #F
<FloatValue> ::=           <FloatDesignator> <Mantissa> <Exponent>
<GeneralPrintableChar> ::=!! Printable characters defined in the character set specified
                           by the Transfer Header, excluding '\', '"', ']', '|' and '#'.

!! Note                    The characters \, ", ], |, and # have been purposely left out
                           of <GeneralPrintableChar> because \ is used as an escape
                           character and the others are used as or in delimiters.
<HeaderKeyword> ::=        :HEADER
<HeightKeyword> ::=        :HEIGHT
<HexaDecimalValue> ::=     <HexRadix> <HexNumber>
<HexDigit> ::=             0 | 1 | 2 | 3 |4 | 5 | 6 | 7 | 8 | 9
                           | A | B | C | D | E | F
                           | a | b | c | d | e | f
<HexNumber> ::=            [ <Sign> ] <HexDigit> ...
<HexRadix> ::=             #H
<Hours> ::=                <Integer>
<Hyphen> ::=               -
<Identifier> ::=           { <UpperOrLowerCaseAlphabeticChar>
                                | <Digit> }
                             [ <UpperOrLowerCaseAlphabeticChar>
                                | <Digit> | <UnderScore>
                                | <Hyphen> ] ...
<IdentifierValue> ::=      * <MultibyteIdentifier> *
<Integer> ::=              [ <Sign> ] <Digit> ...
```

```
<IntegerListKeyword> ::=  :INTEGERLIST
<ListSeparator> ::=       ,
<LowerCaseAlphabeticChar> ::=
                          a | b | c | d | e | f | g | h | i | j | k
                          | l | m | n | o | p | q | r | s | t | u
                          | v | w | x | y | z
<Mantissa> ::=            [ <Sign> ] <Digit> ...
                               [ <DecimalPoint> [ <Digit> ... ] ]
<MetaModelKeyword> ::=    :META-MODEL
<MetaMetaObjectName> ::=  <UpperCaseAlphabeticChar>
                               [ <UpperOrLowerCaseAlphabeticChar> ] ...
<MetaObjectName> ::=      <MultibyteIdentifier>
<Minutes> ::=             <Integer>
<ModelKeyword> ::=        :MODEL
<MonthValue> ::=          <Integer>
<MultibyteIdentifier> ::= <MultibyteIdentifierChar> ...
<MultibyteIdentifierChar> ::=
                          !! Printable characters defined in the character set specified
                          by the Transfer Header, excluding '\', '"','[', ']', '|', '#',
                          '<', '>', '*', ':', '+', '-', '(', ')', ',', '.', and space.
<OctalDigit> ::=          0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
<OctalNumber> ::=         [ <Sign> ] <OctalDigit> ...
<OctalRadix> ::=          #O
<OctalValue> ::=          <OctalRadix> <OctalNumber>
<OpenComment> ::=         #|
<OpenScope> ::=           (
<OpenText> ::=            #[
<OtherChar> ::=           <EscapeCharacter> <HexaDecimalValue> <EscapeCharacter>
<PixelIntensity> ::=      0 .. 255
<PixelSeparator> ::=      ,
<PointKeyword> ::=        :POINT
<PointListKeyword> ::=    :POINTLIST
<PointSeparator> ::=      <Space>
<PositiveInteger> ::=     [ + ] <Digit> ...
<RelativeNegative> ::=    RELATIVENEGATIVE
<RelativePositive> ::=    RELATIVEPOSITIVE
<Seconds> ::=             <Integer>
                          [ <DecimalPoint> <Digit>
                          [<Digit> [ <Digit> ] ] ]
<Sign> ::=                + | -
<Space> ::=               !! Space character as defined in the character set specified
                          by the Transfer Header.
<String> ::=              <StringDelimiter> [ <StringElement> ] ...
                          <StringDelimiter>
```