



TECHNICAL REPORT	EIA-IS-110™	
	Issued 1994-01 Stabilized 2015-07	
CDIF - Transfer Format Encoding - ENCODING.1		

NOTICE

This document has been taken directly from the original TechAmerica document and contains only minor editorial and format changes required to bring it into conformance with the publishing requirements of SAE Technical Standards. The release of this document is intended to replace the original with the SAE International document. Any numbers established by the original document remain unchanged.

The original document was adopted as an SAE publication under the provisions of the SAE Technical Standards Board (TSB) Rules and Regulations (TSB 001) pertaining to accelerated adoption of specifications and standards. TSB rules provide for (a) the publication of portions of unrevised specifications and standards without consensus voting at the SAE committee level, and (b) the use of the existing specification or standard format.

SAE Technical Standards Board Rules provide that: "This report is published by SAE to advance the state of technical and engineering sciences. The use of this report is entirely voluntary, and its applicability and suitability for any particular use, including any patent infringement arising therefrom, is the sole responsibility of the user."

SAE reviews each technical report at least every five years at which time it may be revised, reaffirmed, stabilized, or cancelled. SAE invites your written comments and suggestions.

Copyright © 2016 SAE International

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of SAE.

TO PLACE A DOCUMENT ORDER: Tel: 877-606-7323 (inside USA and Canada)
Tel: +1 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
<http://www.sae.org>

SAE WEB ADDRESS:

Full document details: [SAE EIAIS110](#)

SAE values your input. To provide feedback on this Technical Report, please visit <http://www.sae.org/technical/standards/EIAIS110>

Documents
Visit Document Sharing Hub

Interim Standard

EIA/IS-110

CDIF

CASE Data Interchange Format
A Division of Electronic Industries Association

CDIF
Transfer Format Encoding
ENCODING.1

January 1994

Electronic Industries Association
CDIF Technical Committee



Documents

Document Sharing Hub

NOTICE

EIA Engineering standards and publications are designed to serve the public interest through eliminating misunderstandings between manufacturers and purchasers, facilitating interchangeability and improvement of products, and assisting the purchaser in selecting and obtaining with minimum delay the proper product for his particular need. Existence of such standards and publications shall not in any respect preclude any member or nonmember of EIA from manufacturing or selling products not conforming to such standards and publications, nor shall the existence of such standards and publications preclude their voluntary use by those other than EIA members, whether the standard is to be used either domestically or internationally.

Recommended standards and publications are adopted by EIA without regard to whether or not their adoption may involve patents on articles, materials or processes. By such action, EIA does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standard or publication.

EIA INTERIM STANDARDS

EIA Interim Standards contain information deemed to be of technical value to the industry, and are published at the request of the originating Committee without necessarily following the rigorous public review and resolution of comments which is a procedural part of the development of an EIA Recommended Standard.

EIA Interim Standards should be reviewed on an annual basis by the formulating Committee and a decision made on whether to proceed to develop an EIA Recommended Standard on this subject. EIA Interim Standards must be canceled by the Committee and removed from the EIA Standards Catalog before the end of their third year of existence.

This EIA standard is considered to have international standardization implication, but the International Standardization Organization activity has not progressed to the point where a valid comparison between the EIA standard and the ISO work can be made.

COPYRIGHT 1994

Published by
ELECTRONIC INDUSTRIES ASSOCIATION
Engineering Department
2001 Pennsylvania Avenue N.W.
Washington, D.C. 20006

PRICE: Please refer to the current
Catalog of EIA & JEDEC STANDARDS & ENGINEERING PUBLICATIONS
or contact EIA Electronic Information Group
Phone: (202) 457-4960, FAX: (202) 457-4985

All rights reserved
Printed in U.S.A.

ISBN 0-7908-0015-2

PREFACE

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

EIA/IS-106 CDIF - CASE Data Interchange Format - Overview and EIA/IS-107 CDIF - Framework for Modeling 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 modeling approach in CDIF. The CDIF Meta-meta-model and extensibility mechanism are also defined in that document.

This standard defines an encoding of SYNTAX.1. *EIA/IS-108 CDIF - Transfer Format - General Rules for Syntaxes and Encodings* describes how CDIF supports multiple syntaxes and encodings. *EIA/IS-109 CDIF - Transfer Format - 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.

The material contained in this publication was formulated under the cognizance of the CDIF Technical Committee.

CONTENTS

1. SCOPE	1
2. INTRODUCTION.....	2
2.1 Audience	2
2.2 Structure of this Document.....	2
2.3 Conventions	4
2.3.1 Global Conventions.....	4
2.3.2 BNF Conventions.....	4
3. CONCEPTS AND DEFINITIONS.....	6
3.1 Encoding Identifier.....	6
3.2 Clear Text	6
3.3 Character Sets	6
4. ENCODING STRUCTURES AND KEYWORDS IN THE CDIF TRANSFER....	8
4.1 Introduction.....	8
4.2 Encoding Structures.....	8
4.2.1 Decimal Integer Value	8
4.2.2 Binary Value.....	8
4.2.3 Octal Value	9
4.2.4 Hexadecimal Value	9
4.2.5 Float Value	9
4.2.6 Date	10
4.2.7 Time	11
4.2.8 Identifier Value.....	11
4.2.9 Enumerated Value.....	12
4.2.10 String	12
4.2.11 Text String.....	13
4.2.12 Comment	14
4.2.13 MetaMetaObjectName	15
4.2.14 MetaObjectName	16
4.2.15 Identifier	16
4.3 Keywords	17
5. ENCODING FORMAL GRAMMAR.....	18
APPENDIX A TRANSFER EXAMPLES	25
A1. Introduction.....	25
A2. Example 1	25
A2.1 Introduction	25
A2.2 Layout and Content.....	25
A2.3 The Transfer	26

A3.	Example 2	31
A3.1	Introduction	31
A3.2	Overview	31
A3.3	SubjectArea Definition.....	32
A3.4	Definition of the Additions.....	33
	A3.4.1 MetaEntity Definitions.....	34
	A3.4.1.1 SecurityClassification	34
	A3.4.2 MetaRelationship Definitions.....	36
	A3.4.2.1 SemanticInformationObject. IsControlledBy.SecurityClassification	36
A3.5	Definition of the Extensions.....	37
A3.6	The Transfer	38
APPENDIX B QUESTIONS AND ANSWERS		47
GLOSSARY		49
REFERENCES		57
INDEX.....		59
RELATED EIA DOCUMENTS.....		INSIDE BACK COVER

FIGURES

Figure 1 Position in the CDIF Family of Standards.....	1
Figure A1 Semantic Objects in Example.....	26

1. SCOPE

The CDIF Family of Standards is primarily designed to be used as a description of a mechanism for transferring information between CASE 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 Integrated Meta-model defined for CASE 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 *EIA/IS-106 CDIF - CASE Data Interchange Format - Overview*. These standards cover the overall framework, the transfer format and the CDIF Integrated Meta-model.

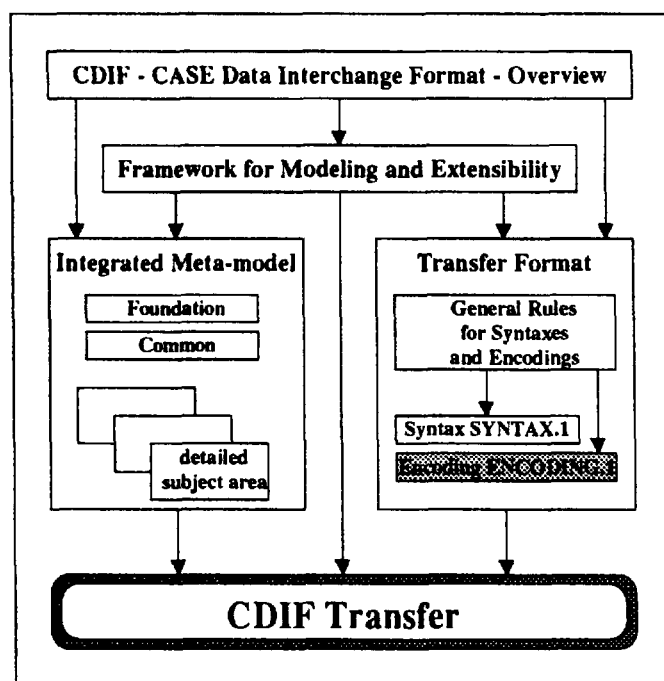


Figure 1
Position in the CDIF Family of Standards

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 *EIA/IS-109 CDIF - Transfer Format - Syntax SYNTAX.1*.

2. INTRODUCTION

2.1 Audience

This document is intended to be used by anyone wishing to understand and/or use CDIF. This document provides a definition of an encoding for CDIF transfers. It is suitable for:

- those evaluating CDIF
- those who wish to understand the principles and concepts of a CDIF transfer
- those developing importers and exporters.

The document *EIA/IS-106 CDIF - CASE Data Interchange Format - Overview* and the document *EIA/IS-107 CDIF - Framework for Modeling 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 *EIA/IS-108 CDIF - Transfer Format - General Rules for Syntaxes and Encodings* and *EIA/IS-109 CDIF - Transfer Format - 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 modeling
- CASE tools
- Information repositories
- Data dictionaries
- Multiple meta-layer modeling
- Formal grammars
- Transfer Formats.

2.2 Structure of this Document

This document is organized into the following sections:

Section 1 'Scope'

This describes the CDIF Family of Standards and positions the current document within the set of documents.

Section 2 'Introduction'

This describes the intended audience, structure and conventions used in the document.

Section 3 'Concepts and Definitions'

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.

Section 4 'Encoding Structures and Keywords in the CDIF Transfer'

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

Section 5 'Encoding Formal Grammar'

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

Appendix A 'Transfer Examples'

This gives two examples of a complete transfer, one with, and one without, extensibility.

Appendix B 'Questions and Answers'

This includes detailed responses to questions that have been raised by those reviewing this document. It is intended that the inclusion of these will aid the reader in understanding this document, and related issues concerning CDIF.

Glossary

The Glossary contains definitions of terms used within this Standard.

References

This contains details of other standards referenced within this Standard.

Index

A comprehensive index is provided.

2.3 Conventions

2.3.1 Global Conventions

The typographical and naming conventions defined below are used throughout the CDIF Family of Standards.

- Double quotes are used to introduce new terms (e.g., "model layer")
- Bold type is used for emphasis (e.g., this is **important**)
- All meta-objects and meta-meta-objects in CDIF (in meta-models and meta-meta-models) 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*)

2.3.2 BNF Conventions

The grammar in this document is defined using an extended version of "Backus Naur Form" (BNF).

In BNF, each syntactic element of the language is defined by means of a "production rule". A "production rule" defines a syntactic element in terms of an expression, ordered left to right, that can be used to form an instance of the element. The expression can consist of characters, character strings, and other syntactic elements.

The version of BNF used in this document has the following conventions:

<i>Symbol</i>	<i>Meaning</i>
< >	Angle braces delimit character strings that are the names of syntactic elements, i.e., non-terminal symbols.
::=	This is the definition operator. It is used in a production rule to separate the element defined by the rule from its definition. The element being defined is to the left of the operator and the expression that defines the element is to the right of the operator.
[]	Square braces enclose optional elements in an expression. The portion of the expression within the braces may be explicitly specified or may be omitted.
{ }	Curly braces group elements in an expression, so that the group can be treated as a single syntactic element.

	This is the alternative operator. The vertical bar indicates that the portion of the expression to the right of the bar is an alternative to the portion to the left of the bar. If the vertical bar appears at a position where it is not enclosed in curly or square braces, it specifies a complete alternative for the element defined by the production rule. If the vertical bar appears in a portion of the expression enclosed in curly or square braces, it specifies alternatives for the contents of the innermost pair of such braces.
...	The ellipsis indicates that the element to which it applies in a formula may be repeated any number of times (but note that optionality is covered by the square braces convention). If the ellipsis appears immediately to the right of a closing curly or square brace, ")" or "]", then it applies to the portion of the expression enclosed between that closing brace and the corresponding opening brace, "{" or "[". If an ellipsis appears immediately to the right of any other element, then it applies only to that element.
!!	This is the comment operator. This operator introduces normal English text. It is used as a comment in the BNF and when the definition of a syntactic element is not expressed further in BNF.
<TokenName>	Anything emboldened and surrounded by angle braces is, a token , a BNF non-terminal symbol that is not further defined in the syntax. The detailed representation of a token is specified in the encoding.
Terminal	Anything emboldened, and not surrounded by angle braces, is a literal value (i.e., it represents the character string itself).
x .. y	This is the range operator. It refers to any symbol in the range x to y. For example, 0..255 refers to all integers 0 to 255 inclusive.

3. CONCEPTS AND DEFINITIONS

3.1 Encoding Identifier

The encoding defined in this version of this standard shall have a CDIF Standardized Syntax Identifier of "ENCODING.1" and a Version of "02.00.00". 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 "02.00.00" (see *EIA/IS-109 CDIF - Transfer Format - Syntax SYNTAX.1*).

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

<EncodingID> ::= "ENCODING.1"

<EncodingVersion> ::= "02.00.00"

3.2 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.

3.3 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. Whitespace characters must not appear anywhere other than as defined in the syntax defined in *EIA/IS-109 CDIF - Transfer Format - Syntax SYNTAX.1*.

The whitespace characters are space, horizontal tab, vertical tab, carriage return, line feed and form feed. Space is encoded using the 8 bit character code identified in ISO 8859-1[1]. Horizontal tab (HT), vertical tab (VT), carriage return (CR), line feed (LF) and form feed (FF) are encoded using the 8 bit character codes identified in ISO 6429 [2].

All other characters in a transfer file are encoded using ISO 8859-1[1].

This does not mean that a text string can not have other characters. For more details, see section 4.2.11 Text String.

4. ENCODING STRUCTURES AND KEYWORDS IN THE CDIF TRANSFER

4.1 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 *EIA/IS-109 CDIF - Transfer Format - Syntax SYNTAX.1*.

4.2 Encoding Structures

4.2.1 Decimal Integer Value

A Decimal Integer 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 |

4.2.2 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 |

4.2.3 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 |

4.2.4 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) | #-hffffffff | - | Valid: negative hexadecimal |
| (4) | #h122g9 | - | Invalid: 'g' not a valid hex digit |

4.2.5 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 |

4.2.6 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 date values:

- | | | | |
|-----|-------------|---|--|
| (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. |

4.2.7 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 time values:

- | | | | |
|-----|--------------|---|--------------------------------|
| (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 |

4.2.8 Identifier Value

The syntax of the Identifier Value clause is:

```

<IdentifierValue> ::=
    • <Identifier> *
```

<Identifier> 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* | - | Valid |
| (4) | *9system* | - | Valid |

4.2.9 Enumerated Value

The syntax of the Enumerated Value clause is:

`<EnumeratedValue> ::= < <Identifier> >`

`<Identifier>` 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

4.2.10 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

4.2.11 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 strings¹:

- (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;
 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 #

4.2.12 Comment

A Comment is defined as follows:

<Comment> ::=	<OpenComment> <CommentBody> <CloseComment>
<OpenComment> ::=	#
<CommentBody> ::=	[<CommentBodyChar>] ... []

¹These examples of the TextString clause have been laid out with whitespace characters, such as <CR>, transformed for presentation. <eof> is used to represent the end of the file.