



INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

V.44

(11/2000)

SERIES V: DATA COMMUNICATION OVER THE
TELEPHONE NETWORK

Error control

Data compression procedures

ITU-T Recommendation V.44

(Formerly CCITT Recommendation)

ITU-T V-SERIES RECOMMENDATIONS
DATA COMMUNICATION OVER THE TELEPHONE NETWORK

General	V.1–V.9
Interfaces and voiceband modems	V.10–V.34
Wideband modems	V.35–V.39
Error control	V.40–V.49
Transmission quality and maintenance	V.50–V.59
Simultaneous transmission of data and other signals	V.60–V.99
Interworking with other networks	V.100–V.199
Interface layer specifications for data communication	V.200–V.249
Control procedures	V.250–V.299
Modems on digital circuits	V.300–V.399

For further details, please refer to the list of ITU-T Recommendations.

Date compression procedures

Summary

This Recommendation describes a data compression algorithm for use in DCEs. It achieves a better performance than V.42 *bis* on many types of data. In addition to the normal stream method, the algorithm has a method which can compress data already contained in packets in an efficient way.

Source

ITU-T Recommendation V.44 was prepared by ITU-T Study Group 16 (2001-2004) and approved under the WTSA Resolution 1 procedure on 17 November 2000.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

© ITU 2001

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from ITU.

CONTENTS

Page

1	Scope.....	1
1.1	General.....	1
1.2	Requirements for error-correcting procedures.....	1
1.3	A DCE employing data compression.....	1
2	References.....	2
3	Definitions	2
4	Abbreviations.....	4
5	Functional description of a DCE	4
5.1	General.....	5
5.2	DTE/DCE interchange circuits	5
5.3	Signal converter	5
5.4	Control function.....	5
5.5	Error-control function	5
5.6	Data compression function	5
6	Procedures of the data compression function	6
6.1	Overview of the data compression function	6
6.2	Dictionary structure	6
6.2.1	Encoder dictionary.....	6
6.2.2	Decoder dictionary	7
6.3	Encoding.....	8
6.3.1	String-matching procedure	8
6.3.2	String-extension procedure.....	9
6.3.3	Creating string-segments	10
6.3.4	Encoding summary	11
6.4	Decoding.....	11
6.4.1	Processing codes.....	11
6.4.2	Creating new strings	12
6.5	Transparent mode	13
6.5.1	Transition from compressed mode to transparent mode	13
6.5.2	Transition from transparent mode to compressed mode	13
6.6	Transfer.....	13
6.6.1	Transfer of control codes, ordinals, and codewords.....	14
6.6.2	Transfer of string-extension length.....	14
6.6.3	Code prefixes.....	15
6.6.4	Example of Transfer	15

	Page
7 Operations of data compression.....	16
7.1 Communication between the control and data compression functions.....	16
7.2 Communications between peer data compression functions	17
7.3 Negotiation of V.44 capability.....	17
7.4 Negotiation of data compression parameters	17
7.4.1 Negotiation through XID.....	18
7.4.2 Negotiation after link establishment.....	18
7.5 Initialization of the data compression function.....	19
7.5.1 Initial state of the encoder dictionary	20
7.5.2 Initial state of the decoder dictionary	20
7.6 Establishment of error-controlled connection.....	20
7.7 Transfer of data between the DTE/DCE interface and the data compression function	20
7.8 Encoding	20
7.9 Transfer of data between the data compression function and the error-control function	20
7.10 Decoding.....	20
7.11 Autonomous adjustments.....	21
7.11.1 Ordinal size and STEPUP.....	21
7.11.2 Codeword size and STEPUP	21
7.11.3 Node-tree full.....	21
7.11.4 History full.....	21
7.11.5 Data compressibility monitoring	21
7.12 Dictionary reinitialization	22
7.13 Expedited data transfer and FLUSH.....	22
7.14 ESCAPE command sequence	22
7.15 Action on detection of C-ERROR	22
8 Parameters.....	23
Annex A – XID information field for negotiating V.44 capability when used with V.42	25
Annex B – Operation of V.44 in Packet Networks.....	26
B.1 Packet method operation of V.44	27
B.1.1 General description.....	27
B.1.2 Default values of data compression parameters for packet method	28
B.2 Multi-packet method of operation of V.44	28
B.2.1 General description.....	28
B.2.2 Default values of data compression parameters for multi-packet method	29

	Page
Appendix I – Notes on Implementation.....	29
I.1 Selection of N_2 : the total number of codewords.....	29
I.2 Selection of N_7 : maximum string length	30
I.3 Selection of N_8 : data structures and length of history	30
I.4 Efficient compression of Unicode data.....	31
I.5 Applicability of transparent mode	31
I.6 Calculation of compression performance	31
I.7 Differences between V.44 and V.42 <i>bis</i>	31
Appendix II – Illustration of operation of V.44 algorithm	32
II.1 Compression and decompression of "ABCDEXABCDEYABCDE FF _H AC"	32
II.2 Compression & decompression of "CCCCCCCCCX"	45

ITU-T Recommendation V.44

Data compression procedures

1 Scope

1.1 General

This Recommendation describes a lossless data compression procedure for use with V-series data circuit-terminating equipment (DCEs).

The principal characteristics of the data compression procedure are:

- a) a compression procedure based on an algorithm that encodes strings of characters input from data terminal equipment (DTE) as binary codes of variable length;
- b) a decoding procedure that recovers the strings of characters from received binary codes of variable length;
- c) a string-building mechanism that rapidly extends existing strings;
- d) an automatically invoked transparent mode of operation when uncompressible data are detected.

Annex B describes the implementation of this data compression procedure in packet networks.

A summary of the set of parameters used in this Recommendation is given in clause 8.

Clause I.7 summarizes the main differences between the algorithms of ITU-T V.44 and V.42 *bis*.

This Recommendation contains examples for illustrative purposes; in any case in which an example appears to contradict the normative text, the normative text shall take precedence.

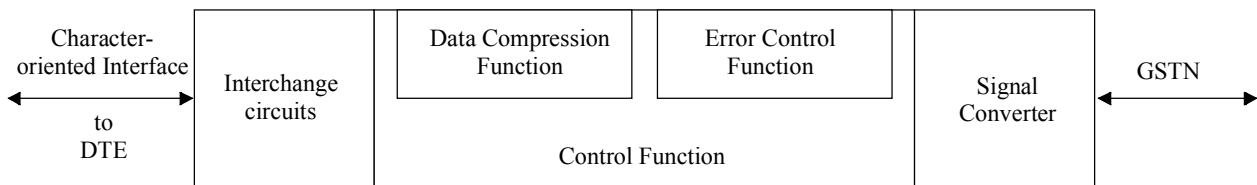
1.2 Requirements for error-correcting procedures

For correct operation of the data compression function, it is necessary that an error-correcting procedure be implemented between the two entities using this Recommendation. In the case of V-series Recommendations, the Link Access Procedure for Modems (LAPM) error-correcting procedures defined in ITU-T V.42 or the error-correcting procedures in ITU-T V.76 or V.120 must be implemented.

NOTE – Undetected bit errors will cause mis-operation of the data compression function. Use of a 32-bit frame check sequence (FCS), as defined in ISO/IEC 13239, substantially reduces the possibility of such errors, and may be desirable in environments with severe impairments. This 32-bit FCS is an option in V.42 LAPM.

1.3 A DCE employing data compression

The data compression function may be used with an error-correcting DCE, as shown in Figure 1 and described in clause 5. The elements of an error-correcting V-series DCE are specified in ITU-T V.42.



T1609040-00

Figure 1/V.44 – DCE employing data compression and error control

2 References

The following ITU-T Recommendations contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

- ITU-T V.42 (1996), *Error-correcting procedures for DCEs using asynchronous-to-synchronous conversion*.
- ITU-T V.42 bis (1990), *Data compression procedures for data circuit-terminating equipment (DCE) using error correction procedures*.
- ITU-T V.120 (1996), *Support by an ISDN of data terminal equipment with V-series type interfaces with provision for statistical multiplexing*.
- ISO/IEC 13239:2000, *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures*.

3 Definitions

This Recommendation defines the following terms:

3.1 alphabet: The set of all possible characters. In this Recommendation, the characters are contiguous from 0 to $N_4 - 1$ (see clause 8).

3.2 append: To create a new string using an unmatched character. If the unmatched character follows a string match, the new string consists of the string match with the unmatched character added to the end; if the unmatched character follows a character, the new string consists of both characters.

3.3 character: A single data element input from the DTE to the encoder using a predefined number of bits N_3 (see clause 8).

3.4 code: A sequence of bits output by the encoder that represents control or information. Defined code types are: control codes, ordinals, codewords, and string-extension lengths.

3.5 code prefix: One or two bits preceding a code that indicates the type of code that follows. These are defined in 6.6.3.

3.6 codeword: A binary number in the range 4 to $N_2 - 1$ that represents a string of consecutive characters. In the encoder, it corresponds to a particular string-segment. In the decoder, it corresponds to an entire string (see clause 8).

- 3.7 compressed mode:** The mode of operation in which data from the DTE are transmitted as binary codes.
- 3.8 compressed operation:** The state of the DCE in which the data compression function is active. Compressed operation has two modes: compressed mode and transparent mode. Transitions between these modes may be automatic, and based on the content of the data input from the DTE (see 6.5).
- 3.9 control code:** A binary number in the range 0 to 3, reserved for use in DCE-to-DCE signalling of control information related to the compression function while in the compressed mode of operation (see 7.2).
- 3.10 decoder:** The data compression subfunction that decompresses the output of an encoder.
- 3.11 dictionary:** The data structure that represents the strings created while encoding or decoding. For the encoder, it consists of the root array, node-tree, and history; for the decoder, it consists of the string collection and the history.
- 3.12 encoder:** The data compression subfunction that compresses data.
- 3.13 ESCAPE:** In transparent mode, the indication of the beginning of a command sequence to the decoder.
- 3.14 extend:** To create a new string, consisting of a string match with one or more characters added to the end.
- 3.15 flush:** The encoder completes processing of characters input up to that point, transfers any codes that result, updates the dictionary as appropriate, and establishes octet alignment. The decoder subsequently establishes octet alignment.
- 3.16 history:** The history buffer, herein called "the history", is the data structure that contains the characters input to the encoder (or equivalently, decoded by the decoder) since the most recent reinitialization of the data-structures. Characters are entered into the history in the order in which they are input (or decoded).
- 3.17 multi-packet method:** The compression of data that are packetized (or framed) such that both ends of the data transmission can identify packet (or frame) boundaries, and several packets, or portions of packets, are processed as continuation. The multi-packet method is described in Annex B.
- 3.18 node:** A node is a point in a tree-structure that represents a string-segment. It corresponds to a codeword in the encoder.
- 3.19 node-tree:** The data structure in the encoder that represents the set of string-segments and their interrelationships. Combined with the root array, it represents the encoder tree-structures. A codeword corresponds to a particular string-segment and to a particular node.
- 3.20 ordinal:** The ordinal of a character is the numerical equivalent of the character.
- 3.21 packet method:** The compression of data that are packetized (or framed) such that both ends of the data transmission can identify packet (or frame) boundaries, and each packet is processed separately. The packet method is described in Annex B.
- 3.22 parameter mode:** The mode of operation in which compression parameters are transferred between the data compression peer.
- 3.23 receive:** In parameter negotiation, the direction corresponding to an entity's decoder.
- 3.24 root:** A root is the point at the base of a tree-structure that represents, within the context of this Recommendation, the first character of a string (see Figure 2 and 6.2.1). Each character in the alphabet has a corresponding root.
- 3.25 root array:** The data structure that contains the set of all encoder roots.

- 3.26 stream method:** The compression of data that are transmitted continually over a link with guaranteed delivery. This method is the primary focus of ITU-T V.44.
- 3.27 string:** An unbroken sequence of two or more characters. In the encoder dictionary, it consists of a first character (root) followed by one or more string-segments.
- 3.28 string extension:** A sequence of one or more characters by which a string is extended to create a new and longer string.
- 3.29 string-extension length:** A binary code indicating the number of characters by which the string corresponding to the immediately preceding codeword is extended.
- 3.30 string-segment:** A section of a string that corresponds to a particular codeword in the encoder node-tree.
- 3.31 string collection:** The data structure in the decoder that represents the decoded strings.
- 3.32 transmit:** In parameter negotiation, the direction corresponding to an entity's encoder.
- 3.33 transparent mode:** The mode of operation in which characters from the DTE are transmitted in uncompressed form. The data compression function is active, but does not influence the data transmission.
- 3.34 tree-structure:** An abstract data structure in the encoder that is used in this Recommendation to represent a set of strings, all with the same initial character (see Figure 2 and 6.2.1).
- 3.35 uncompressed operation:** The state of the DCE in which the data compression function is inactive.
- 3.36 unmatched character:** A character that causes the termination of the string-matching or string-extension process because it does not match a specific character in the history. The unmatched character then becomes the first character of a possible new string match.
- 3.37 $\{\}$:** $\{A\}$ is the vector representation of a binary code. It has as many components as there are bits in the code. The component A_1 is the least significant bit of the binary code.

4 Abbreviations

This Recommendation uses the following abbreviations:

DCE	Data Circuit-terminating Equipment
DTE	Data Terminal Equipment
ECM	Enter Compressed Mode: a transparent mode command defined in 7.2
EID	ESCAPE In Data: a transparent mode command defined in 7.2
EPM	Enter Parameter Mode: a transparent mode command defined in 7.2
ETM	Enter Transparent Mode: a control code defined in 7.2

5 Functional description of a DCE

This clause describes a DCE employing a data compression function.

5.1 General

A DCE employing data compression, as depicted in Figure 1, contains the following components:

- a) DTE/DCE interchange circuits;
- b) a signal converter;
- c) a control function;
- d) an error-control function;
- e) a data compression function.

5.2 DTE/DCE interchange circuits

This component shall have capabilities as described in ITU-T V.42.

5.3 Signal converter

This component shall have capabilities as described in ITU-T V.42.

5.4 Control function

This component shall have capabilities as described in 6.2/V.42. Additionally, it shall perform the following aspects of operation:

- a) negotiation of modes of operation of the data compression function with the remote DCE, and negotiation of parameters associated with the operation of the data compression function;
- b) instigation of the initialization or reinitialization of the data compression function;
- c) coordination of the establishment of an error-controlled connection for use by the peer data compression functions;
- d) coordination of the delivery of data between the DTE/DCE interface and the data compression function, in accordance with the procedures defined in 6.2/V.42 and 8.4/V.42, including the provision of the flow-control procedures defined therein;
- e) coordination of the delivery of data between the data compression function and the error-control function;
- f) action on detection of an exception condition.

5.5 Error-control function

This component shall have capabilities as described in ITU-T V.42.

5.6 Data compression function

The data compression function shall implement the procedures defined in this Recommendation in order to efficiently encode data prior to transmission over an error-controlled connection. It shall perform the following aspects of operation:

- a) initialization and reinitialization of the encoder and decoder dictionaries;
- b) data compression encoding and decoding;
- c) switching between compressed and transparent modes of operation;
- d) configuration of encoder and decoder in accordance with the parameters negotiated by the control function.

6 Procedures of the data compression function

6.1 Overview of the data compression function

The data compression function consists of an encoder and a decoder. A data connection, in general, supports data transmission in both directions, and thus can support data compression in both directions. Thus, each data connection peer can have an encoder and a decoder. The encoder transfers compressed data to its peer decoder at the other end of the connection, and the decoder decompresses the compressed data received from its peer encoder. The encoder-decoder pair for each direction must have coordinated data compression parameter values, established through negotiation; however at any one end, the encoder and decoder parameter values may differ, since they refer to different directions of transmission.

Characters from the DTE input to the encoder are matched against any previously identified strings of characters. If a string is matched, the codeword representing the string is transferred, and then an attempt is made to extend the matched string to encode additional characters and create a new and longer string. If no string match is found, the ordinal corresponding to the first character is transferred.

The encoder continually adds to its set of strings available for matching, by placing input characters into the history and by adding nodes to the node-tree. When either of these becomes full, the dictionary is reinitialized, and the encoding operation continues as before.

The decoder creates strings that replicate the strings created by its peer encoder, using the same assigned codewords. By using these strings, it decompresses the received compressed data stream. The decoder dictionary is reinitialized upon receipt of a REINIT control code.

6.2 Dictionary structure

To describe in detail the data compression algorithm, it is useful to define the following data structures.

6.2.1 Encoder dictionary

The encoder uses a dictionary consisting of three parts:

- 1) *root array*: This contains an entry for each character in the alphabet. The size of the alphabet is N_4 , and is determined by the size of the characters input from the control function: for an alphabet of 8-bit characters, there are 256 entries. Each entry is indexed by its character and serves as the root for a tree-structure; it contains a down-index to a node in the node-tree. This index is the starting point for climbing down the tree-structure.
- 2) *node-tree*: This contains the codewords created during the operation of the encoder, and the string-segments to which they correspond. The nodes in the node-tree define a set of tree-structures, and the corresponding string-segments. Each node contains a codeword, the position in the history of the first character of the string-segment, the number of characters in the string-segment, and two indices for linking to other nodes. The "down-index", if valid, points to a node that represents a string-segment that follows this string-segment. The "side-index", if valid, points to a node that represents a string-segment at the same level as this string-segment (i.e. that also follows the root or string-segment that precedes this string segment). The number of codewords cannot exceed N_{2T} .
- 3) *history*: This contains all the characters input to the encoder, in order, since the most recent dictionary reinitialization. String-segments in the node-tree are referenced by the position of their first character in the history and by their length. The number of characters in the history cannot exceed N_{8T} .

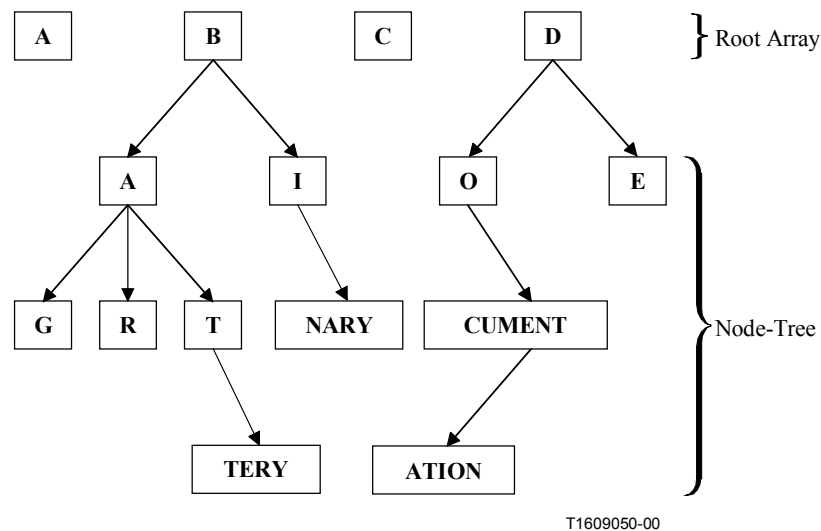


Figure 2/V.44 – Example Tree-structures

This figure shows only the tree-structures starting from roots A, B, C and D.

The tree-structures of Figure 2 show the string-segments, each represented by a node:

- A and I, following root character B;
- G, R, and T following string-segment A;
- TERY following string-segment T;
- NARY following string segment I;
- O and E following root character D;
- CUMENT following string-segment O;
- ATION following string-segment CUMENT.

The strings that are represented: BA, BAG, BAR, BAT, BATTERY, BI, BINARY, DO, DOCUMENT, DOCUMENTATION, DE. A complete string is matched by starting at the root using the next character to be encoded and matching successive string-segments down the tree using subsequently input characters.

6.2.2 Decoder dictionary

The decoder uses a dictionary consisting of two parts:

- 1) *string collection*: This defines the strings created during the decoding process that correspond to the string-segments created by the encoding process. Each entry has a codeword, the position in the history of the last character of the string, and the total length of the string. The number of codewords cannot exceed N_{2R} .
- 2) *history*: This contains all the characters decoded, in order, since the most recent reinitialization of the dictionary. It is identical to the encoder's history; however, it is generated using the characters resulting from the decoding process. The number of characters in the history cannot exceed N_{8R} .

6.3 Encoding

The encoding algorithm attempts to find a string of previously processed characters that matches the next characters to process; if this is found, the codeword representing the string is transferred. Subsequently, the decoder, upon receiving the codeword, can correctly regenerate this string. Data compression is successful if the encoded string requires fewer bits to transfer than the original uncompressed characters.

Input characters are placed into the next available locations in the history and processed by the encoder as follows:

- a) Using the input character and the next character(s), the encoder attempts to find the longest matching string in the dictionary.
- b) If a string match is not found, the encoder transfers the ordinal corresponding to the input character, and returns to step a) using the next character.
- c) If a string match is found, the encoder transfers the codeword assigned to the last completely matched string-segment of the longest string match.
- d) Subsequently, the encoder attempts to extend the longest string match, up to the maximum string length. The encoder attempts to match the next character(s) to the character(s) in the history immediately following the last character of the last completely matched string-segment of the longest string match.
- e) If the longest string match is successfully extended by one or more characters, the encoder transfers a string-extension length indicating the number of characters by which the string has been extended.

Using the unmatched character (i.e. the first character that was not part of the string match or the extension) as the next character, the encoder returns to step a).

6.3.1 String-matching procedure

A string in the encoder dictionary consists of a first character followed by one or more string-segments of the node-tree; for example, in Figure 2, the string "BATTERY" has first character "B", followed by string-segment "A", followed by string-segment "T", followed by string-segment "TERY". The total length of the string cannot exceed N_{7T} .

The string-matching procedure uses the dictionary structure described in 6.2 to find a longest string match as follows:

- Each root in the root array has one down-index: if it is set to a valid index, the corresponding node provides the first string-segment for comparison.
- Each node in the node-tree has one side-index used by the encoder to search for another string-segment that also branches from the same previous node or root (another node at the same level).
- Each node in the node-tree has one down-index used by the encoder to search for a string-segment for continued comparison (a node at the next level).

In searching for a string match, the encoder examines the root corresponding to the first character input, and compares subsequently input characters to the string-segments corresponding to nodes directly branching from that root. If the subsequently input characters exactly match any complete string-segment (partial matches are not valid), the encoder compares additional input characters to the string-segments of the nodes branching from that node. Thus, the encoder continues down the tree-structure until it cannot find any further matches or until there are no more branching nodes.

The string-matching procedure fails if the root corresponding to the first character has no valid down-index; or if no second-level node matches the second character. In this case, the ordinal corresponding to the first character is transferred, and the string-matching procedure begins again with the next (unmatched) character.

Otherwise, the string-matching procedure finds a node that corresponds to the longest string in the dictionary that matches the input characters. The encoder transfers the codeword for this node. (Note that the codeword actually corresponds directly to the last completely matched string-segment of the longest string match, as the encoder does not explicitly associate codewords with complete strings.)

The encoder can transfer any codeword it has created. If the codeword value is equal to the decoder's value of C_1 , this indicates a codeword not yet created in the decoder; nonetheless, the decoder is required to correctly interpret all codeword values up through C_1 . See 6.4.1.

The string-matching procedure can also be terminated for the following reasons: the history is full; or a FLUSH request is received. The processing is similar to that when the next input character does not match any character in the current string-segment: the codeword for the last completely matched string-segment (if any) is transferred; then if there is a partially matched string-segment, the successfully matched characters are transferred as a string-extension length, and a corresponding node-tree entry is created.

NOTE 1 – In order for a string-segment to match the input characters, all characters must match: partial matches are not valid. However, a partial match of a string-segment can be used to extend the string, if no complete string-segment match is found at the same level. In this case, the string-matching procedure has already done the comparisons needed to find the longest possible string extension specified in 6.3.2, and this number is precisely the number of matched characters of the partially matched string-segment. Therefore, it is not necessary to initiate the string-extension procedure explicitly, as the encoder can simply transfer the string-extension length corresponding to the number of matched characters of the partially matched string-segment.

NOTE 2 – It is advantageous to order the nodes at one level that branch off a particular node, alphabetically by first character, to allow more efficient search for matching string-segments. There can be multiple string-segments, each with the same first character, branching off a node.

6.3.2 String-extension procedure

If the string-matching procedure is successful (i.e. a longest string match is found), and if the total length of the string is less than N_{7T} , the encoder initiates the string-extension procedure. The encoder attempts to extend the longest string match by comparing the next character(s) to the character(s) in the history that immediately follow the last character of the last completely matched string-segment.

If a C-FLUSH is received from the control function immediately following a successful string match and before the string-extension procedure has processed a character, the string-extension procedure is terminated. The character immediately following the C-FLUSH, when received, starts a new string match and the dictionary is updated as if that character were unmatched.

For instance, suppose that the string "BATTERY" is the longest string match. The encoder compares the next character to the character in the history following the "TERY" string-segment. There are two possibilities:

- The next character does not match the character in the history immediately following the "Y" of the "TERY" string-segment. The string-extension procedure terminates unsuccessfully, and this next character is appended to the "TERY" to create a new and longer string, and also becomes the first character of a possible new string match.

- The next character does match the character in the history immediately following the "Y" of the "TERY" string-segment. The string-extension procedure continues by comparing the subsequent character input to the second character in the history following the "Y" of the "TERY" string-segment; and the next character input to the third character in the history following the "Y"; and so forth. The number of characters successfully matched is transferred as a string-extension length. The encoder creates a new string-segment in the node-tree, consisting of the string extension (the sequence of characters matched after the longest string match). This string-segment is assigned the next available codeword, and has string-segment length equal to the length of the extension. Its history index references the position in the history of the first of the most recent input characters used to extend the string.

The string-extension procedure can also be terminated for the following reasons: the history is full; a FLUSH request; the length of the total string has reached N_{7T} . The processing is handled in a way similar to that when the next input character does not match the next character in the history: any characters that have been successfully matched to extend the string are transferred as a string-extension length, and a corresponding node-tree entry is created.

6.3.3 Creating string-segments

The encoder continually creates new strings for possible future string matches. The two methods for creating new string-segments are:

Append

String-matching failure:

- If the string match fails because the first character has no valid down-index in the root array, a 1-character string-segment is created, linking the next character to the root of the first character.
- If the string match fails because the next character does not match any existing nodes branching from the first character's root, a 1-character string-segment is created using the unmatched character, branching from the root of the first character.

String-extension failure:

- A 1-character string-segment is created linking the unmatched character to the last completely matched string-segment of the longest string match.

Extend

String-extension success:

A string-segment is created extending the longest string match by one or more characters. The string-segment thus created has length equal to the number of characters of the extension, and is linked to the last completely matched string-segment.

6.3.4 Encoding summary

Table 1/V.44 – Encoding procedures

Procedure	Result	Code transferred	New string-segment	Use of unmatched character
String-Matching	Failure	Ordinal corresponding to first character	Append unmatched character to root of first character	As first character for String-Matching
String-Matching	Success	Codeword for last completely matched string-segment	None	For String-Extension
String-Extension	Failure	Nothing	Append unmatched character to last completely matched string-segment	As first character for String-Matching
String-Extension	Success	String-extension length	All extension characters, linked to last completely matched string-segment	As first character for String-Matching

Table 1 summarizes the encoding procedures. Appendix II illustrates the operation of the algorithm explicitly in terms of the dictionary structure.

6.4 Decoding

A string in the decoder is defined by the position in history of the last character of the string, and by the total length of the string. It corresponds to a particular codeword that, by construction, corresponds in the encoder to the final string-segment of that string.

In compressed mode, the binary codes received by the decoder are parsed into control codes, ordinals, codewords and string-extension lengths, by examining the code prefixes. The decoder must remain synchronized with the strings and codewords that are created by the encoder, updating the string collection and history based upon codes received. The decoder is simpler than the encoder, because the encoder must search for string matches and create new and longer strings, whereas the decoder needs only to keep track of the strings it has created.

The decoder shall be capable of operation in both transparent and compressed modes.

6.4.1 Processing codes

In compressed mode, the decoding function shall operate as follows:

- 1) On receipt of an ordinal, the decoder shall place the corresponding character into the decompressed output and into the history.
- 2) On receipt of a codeword less than C_1 , the decoder shall copy the string represented by the codeword into the decompressed output and into the history.
- 3) On receipt of a codeword equal to C_1 , if the previous code received was a codeword, the decoder shall copy the string represented by that previous codeword into the decompressed output and into the history; and shall then place the first character of the string represented by that previous codeword into the decompressed output and into the history.
- 4) On receipt of a codeword equal to C_1 , if the previous code received was an ordinal, the decoder shall process the corresponding character according to step 3) above, as if it were a string of length one.