ARIB STD-T63-23.042 V11.0.0

Compression algorithm for text messaging services

(Release 11)

Refer to “Industrial Property Rights (IPR)” in the preface of ARIB STD-T63 for Related Industrial Property Rights. Refer to “Notice” in the preface of ARIB STD-T63 for Copyrights.

3GPP TS 23.042 V11.0.0 (2012-09)

Technical Specification

3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Compression algorithm for text messaging services (Release 11)

The present document has been developed within the 3rd Generation Partnership Project (3GPP TM) and may be further elaborated for the purposes of 3GPP.

The present document has not been subject to any approval process by the 3GPP Organisational Partners and shall not be implemented.
This Specification is provided for future development work within 3GPP only. The Organisational Partners accept no liability for any use of this Specification.
Specifications and reports for implementation of the 3GPP TM system should be obtained via the 3GPP Organisational Partners' Publications Offices.

Keywords
UMTS, LTE, algorithm, text, terminal

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Copyright Notification

No part may be reproduced except as authorized by written permission.
The copyright and the foregoing restriction extend to reproduction in all media.

© 2012, 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TTA, TTC).
All rights reserved.

UMTS™ is a Trade Mark of ETSI registered for the benefit of its members
3GPP™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners
LTE™ is a Trade Mark of ETSI currently being registered for the benefit of its Members and of the 3GPP Organizational Partners
GSM® and the GSM logo are registered and owned by the GSM Association

Contents

Foreword 6
Introduction 6
1 Scope 8
2 References 8
2.1 Normative references 8
2.2 Informative references 8
3 Abbreviations 8
4 Algorithms 8
4.1 Huffman Coding 9
4.2 Character Groups 10
4.3 UCS2 10
4.4 Keywords 11
4.5 Punctuation 11
4.6 Character Sets 11
5 Compressed Data Streams 11
5.1 Structure 12
5.2 Compression Header 12
5.2.1 Compression Header - Octet 1 12
5.2.2 Compression Header - Octets 2 to n 13
5.2.2.1 Compression Header reserved extension types and values 15
5.2.3 Identifying unique parameter sets 15
5.3 Compressed Data 15
5.4 Compression Footer 17
6 Compression processes 17
6.1 Overview 17
6.1.1 Compression 18
6.1.2 Decompression 19
6.2 Character sets 20
6.2.1 Initialization 20
6.2.2 Character set conversion 21
6.2.3 Character case conversion 21
6.3 Punctuation processing 21
6.3.1 Initialization 22
6.3.2 Compression 23
6.3.3 Decompression 24
6.4 Keywords 24
6.4.1 Dictionaries 24
6.4.2 Groups 25
6.4.3 Matches 27
6.4.4 Initialization 28
6.4.5 Compression 28
6.4.6 Decompression 29
6.5 UCS2 29
6.5.1 Initialization 29
6.5.2 Compression 29
6.5.3 Decompression 29
6.6 Character group processing 29
6.6.1 Character Groups 30
6.6.2 Initialization 31
6.6.3 Compression 31
6.6.4 Decompression 33
6.7 Huffman coding 33
6.7.1 Initialization Overview 34

6.7.2 Initialization 35
6.7.3 Build Tree 36
6.7.4 Update Tree 36
6.7.5 Add New Node 36
6.7.6 Compression 37
6.7.7 Decompression 37

7 Test Vectors 37

Annex A (normative): German Language parameters 39

A.1 Compression Language Context 39

A.2 Punctuators 39

A.3 Keyword Dictionaries 40

A.4 Character Groups 44

A.5 Huffman Initializations 46

Annex B (normative): English language parameters 50

B.1 Compression Language Context 50

B.2 Punctuators 50

B.3 Keyword Dictionaries 51

B.4 Character Groups 55

B.5 Huffman Initializations 57

Annex C (normative): Italian Language parameters 61

Annex D (normative): French Language parameters 62

Annex E (normative): Spanish Language parameters 63

Annex F (normative): Dutch Language parameters 64

Annex G (normative): Swedish Language parameters 65

Annex H (normative): Danish Language parameters 66

Annex J (normative): Portuguese Language parameters 67

Annex K (normative): Finnish Language parameters 68

Annex L (normative): Norwegian Language parameters 69

Annex M (normative): Greek Language parameters 70

Annex N (normative): Turkish Language parameters 71

Annex P (normative): Reserved 72

Annex Q (normative): Reserved 73

Annex R (normative): Default Parameters for Unspecified Language 74

R.1 Compression Language Context 74

R.2 Punctuators 74

R.3 Keyword Dictionaries 74

R.4 Character Groups 74

R.5 Huffman Initializations 75

Annex S (informative): Change history 76

Foreword

This Technical Specification has been produced by the 3GPP.

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of this TS, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 Indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the specification;

Introduction

This clause introduces the concepts and mechanisms involved in the compression and decompression of a stream of data.

Overview

Central to the compression of a stream of data and the subsequent recovery of the original data is the that both sender and receiver have information that not only describes the content of the data stream, but how the stream is encoded.

For example, a simple rule such as "it's 8 bit data" is enough to transport any character value in the range 0 to 255 with 8 bits being required for each and every character. In contrast if both sender and receive know that some characters are more frequent than others, then the more frequent might be encoded in fewer bits while the less frequent in more - resulting in a net reduction of the total number of bits used to express the data stream.

This knowledge of the nature of the data stream can be established in two ways. Either both sender and receiver can agree some key aspects of the data stream *prior* to it being processed or key aspects of the data can be garnered *dynamically* during its processing.

The disadvantage of an approach based on "prior information" is that it must be known. It can either be carried as a header to the data stream, in which case it adds to the net size of the compressed stream. Or it can be fixed and known to the (de)compression algorithm itself in which case compression performance degrades as a given stream diverges in nature from these fixed and known states. In contrast, the disadvantage of "dynamic information" is that it must be discovered; typically this means a greater processing requirement for the (de)compressor. It also implies that compression performance is initially poor as the algorithm has to "learn" about the data stream before it can apply this knowledge. It will also require greater working memory to store its knowledge about the data stream.

The choice of compression algorithms is always a balancing of compression rate (in terms of fewer output bits), working memory requirements of the (de)compressor and CPU bandwidth. For the compression of SMS messages, there is the additional requirement that it should work well (in terms of compression rate) even on short data streams.

Compression / Decompression is an optional feature but when implemented, the only mandatory requirement is ‘Raw Untrained Dynamic Huffman’ . The default initialisation for the Huffman Encoder / Decoder operating in the Raw Untrained Dynamic Huffman mode are defined in annex R. (See also subclause 4.1.)

i.e. There is no need for any pre-defined attributes such as language dependency to be included. This is of particular significance for entities such as an MS which may have memory storage constraints.

1 Scope

The present document introduces the concepts and mechanisms involved in the compression and decompression of a stream of data.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

2.1 Normative references

[1] 3GPP TS 23.038: "Alphabets and language-specific information".

2.2 Informative references

[2] "The Data Compression Handbook 2nd Edition" by Mark Nelson and Jean-Loup Gailly, published by M&T Books, ISBN 1-22851-434-1.

3 Abbreviations

For the purposes of the present document, the following abbreviations apply.

CD	Compressed Data
CDS	Compressed Data Stream
CDSL	Compressed Data Stream Length
CF	Compression Footer
CG-ID	Character Group ID
CH	Compression Header
CLC	Compression Language Context
HI-ID	Huffman initialization ID
KD-ID	Keyword Dictionary ID
PU-ID	PUnctuator ID

4 Algorithms

The compression algorithm comprises a number of components that may be combined in a variety of configurations. The discrete algorithms are discussed in the following subclauses.

4.1 Huffman Coding

The base compression algorithm is a Huffman coder, whereby characters in the input stream are represented in the output stream by bit sequences of variable length. This length is inversely proportional to the frequency with which the character occurs in the input stream.

This is the only component of the whole compression algorithm that can be expected to be included in any implementation, all other components are optional.

There are two possible approaches here:

a) the (de)coder can be "pre-loaded" with a character frequency distribution, thus improving compression rate for streams that approximate to this distribution; or

b) the (de)coder can adapt the frequency distribution it uses to (de)code characters based on the incidence of previous characters within the input stream.

In both cases, the character frequency distribution is represented in a "tree" structure, an example of which is shown in figure 1.

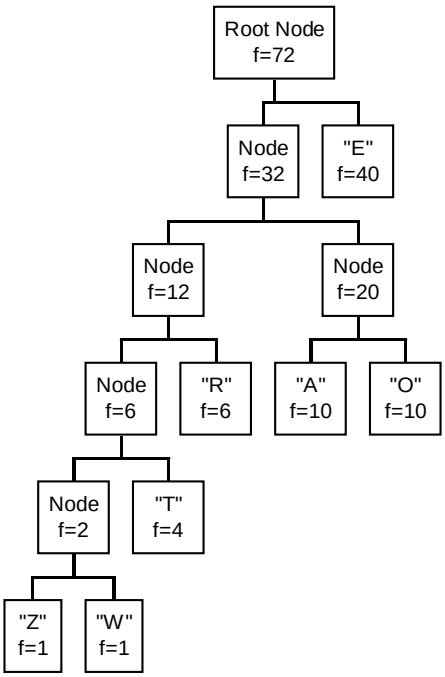

Figure 1: Character frequency distribution

The tree represents the characters Z, W, T, R, A, O and E which have frequencies of 1, 1, 4, 6, 10, 10 and 40 respectively. The characters may be coded as variable length bit streams by starting at the "character node" and ascending to the "root node". At each stage, if a left hand path is traversed, a 0 bit is emitted and if a right hand path is traversed a 1 bit is emitted. Thus the infrequent Z and W would require 5 bits, whereas the most frequent character E requires just 1 bit. The resulting bit stream is decoded by starting at the "root node" and descending the tree, to the left or right depending on the value of the current bit, until a "character node" is reached.

It is a requirement that at any time the trees expressing the character frequencies shall be identical for both coder and decoder. This can be achieved in a number of ways.

Firstly, both coder and decoder could use a fixed and pre-agreed frequency distribution that includes all possible characters but as noted above, this use of "prior information" suffers when a given input stream has a significantly different character frequency distribution.

Secondly, the coder may calculate the character frequency distribution for the entire input stream and prepend this information to the encoded bit stream. The decoder would then generate the appropriate tree prior to processing the bitstream. This approach offers good compression, especially if the character frequency information may itself be

compressed in some manner. Approaches of this type are common but the cost of the prepended information for a potentially small data stream makes it less attractive.

Thirdly, extend the algorithm such that although both coder and decoder start with known frequency distributions, and subsequently adapt these distributions to reflect the addition of each character in the input stream. One possibility is to have initial distributions that encompass all possible characters so that all that is required, as each input character is processed, is to increment the appropriate frequency and update the tree. However, the inclusion of all *possible* characters in the initial distribution means that the tree is relatively slow to adapt, making this approach less appropriate for short messages. An alternative is to have an initial distribution that does not include all possible characters and to add new characters to the distribution if, and when, they occur in the input stream.

To achieve the latter approach, the concept of a "special" character is required. A "special" character is one whose value is outside the range of the character set being used (e.g. 256 if the character set has a range 0 to 255). These characters therefore do not form part of the input stream being conveyed, but their existence in the compressed stream signals the need for the decoder to adjust its behaviour. Here a "special" character is used to signal that the following n bits (where n is a fixed value) represent a new character that needs to be added to the frequency distribution. In the example above this would be done by replacing the "character" node containing the character Z with a new node that had as its children the "character" nodes for Z and for the new character.

This is the approach taken here. It provides considerable flexibility, effectively enabling all of the foregoing approaches. The specific approach to be used for a given message is signalled in the header.

The algorithm uses an additional optimization in that 2 special characters are defined, one meaning that a 7-bit literal follows and the other for 8-bit characters. So for example:

- The initial tree can contain just the "new character follows" special character(s). In this case, the input stream "AAA" would result in:
  [1 bit = new character(7bit) special][7 bits = "A"][2 bits = "A"][1 bit = "A"]

- As can be seen from the above there is quite a high cost in adding a new character (the "special" plus literal). So if the initial tree contains a small subset of the generally most frequently used characters, the cost of character addition can be avoided for these characters.

- Given that we can signal in the header a specific initial frequency distribution, there is no reason why this distribution cannot contain all possible characters and frequency adaptation enabled or disabled as appropriate.

A detailed description of Huffman coding can be found in Chapter 4 of "The Data Compression Handbook 2nd Edition" by Mark Nelson and Jean-Loup Gailly, published by M&T Books, ISBN 1-22851-434-1.

4.2 Character Groups

Character grouping is an optional component that can effect an increase in compression performance of the Huffman coder. This technique groups characters that may be expected to occur together within the input stream and signals transitions between the groups rather than each individual character.

The algorithm derives benefit by;

a) reducing the need to add new characters to the frequency distribution; and

b) using a smaller overall tree. For example, assume that there is no pre-loaded distribution and a stream comprised the characters "abcdefABCDEF".

The capital letters can be encoded more efficiently by signalling the transition to "upper case" and then coding the extant lower case characters rather than introducing 6 new characters. "Special" characters are used to signal transitions between groups of characters.

4.3 UCS2

Input streams comprising 16bit UCS2 information are handled in a manner similar to Character groups. Both coder and decoder maintain knowledge of "the current" Basic Multilingual Plane row for characters in the input stream and the

row octet itself is then omitted from the output stream for sequences of characters within that row. Transitions between rows are signalled in the output stream by a "special" character.

Support for UCS2 is optional.

4.4 Keywords

The algorithm optionally supports the concept of dictionaries - essentially a list of key words or phrases of up to 255 characters in length. Dictionaries need to be known to both the coder and the decoder. The input stream is matched against entries in the dictionary and matching characters in the stream are replaced with a reference to the dictionary entry.

Again "special" characters are used to signal that the following sequence of bits describe a reference to a dictionary entry. So for example, if a dictionary contains the phrases "Please" and "meeting", an input stream "Please cancel the monthly meeting" would be rendered as:

[keyword special][10 bits = "Please"][.....][keyword special][10 bits = "meeting"]

Dictionary matches for long strings can result in very high compression rates.

4.5 Punctuation

The punctuation processor is distinct from the other algorithms in that it is non-symmetric so the decompressed stream may not be identical to the original. Its use is therefore mainly applicable to input streams comprising human readable sentences where it is sufficient to preserve the meaning of the content, but not the exact format. It is also applicable when the input stream is a "standard sentence" that is known to produce a symmetric result. The punctuation processor is applied before (on coding) and after (on decoding) any of the other algorithms. Its functions are:

- to remove leading and trailing spaces from the input stream;
- to replace repeated spaces within the stream with a single space;
- to remove (on coding) and insert (on decoding) spaces following certain punctuation characters;
- to decapitalize (on coding) and capitalize (on decoding) the first character of the stream, the first character following an appropriate punctuation character or a paragraph symbol and capitalized single character words such as "I";
- to remove (on coding) and insert (on decoding) a full stop if it is the last character of the stream.

The use of the punctuation processor is optional.

4.6 Character Sets

The use of pre-loaded frequencies, key word dictionaries and the punctuation processor all require that a consistent character set is used by both coder and decoder. As there can be no assumption that the same character will be have the same value (or even be available) on the devices used to send and receive a compressed message, the algorithms are specified to operate on a known character set *to* which (prior to coding) and *from* which (post decoding) a device needs to convert an input stream to render it in the native character set of the device.

The handling of character sets is mandatory for all implementations.

5 Compressed Data Streams

This clause provides:

- A detailed specification of the algorithms and data structures that implement compression and decompression mechanisms.

5.1 Structure

A Compressed Data Stream (CDS) comprises three key components:

- a Compression Header (CH) containing a variable number of octets, the content of which defines the nature of the compressed data;
- the Compressed Data (CD) which is a bit stream of variable length;
- a Compression Footer (CF) which is used to signal the number of bits in the last octet of the CDS that form part of the compressed data.

5.2 Compression Header

The Compression Header (CH) comprises a variable number of octets that define the nature of the compressed data.

The compression header allows for a wide range of compression alternatives, however of these alternatives only one is defined as the basic mandatory form of compression that shall be supported by all implementations. This is the use of the basic Huffman algorithm initialized with no prior knowledge of character distribution. This case can be signalled directly by setting a single octet(octet 1) for the compression header with the value of 120 (decimal).

5.2.1 Compression Header - Octet 1

The first CH octet is mandatory and is defined as follows:

Table 1: CH octet

7	6	5	4	3	2	1	0	Description
0								There is no subsequent CH octet
1								A further CH octet follows
	n	**n**	**n**	**n**				The "Compression Language Context" this is described below
					0			Punctuation processing disabled
					1			Punctuation processing enabled
						0		Keyword processing disabled
						1		Keyword processing enabled
							0	Character group processing disabled
							1	Character group processing enabled

As noted in clause 4, the compression algorithms can be configured to operate in a variety of ways and may rely on end-to-end knowledge of "prior" information such as which key word dictionary is to be used.

A requirement that all configuration information be explicitly stated in the CH is less efficient (in terms of compression ratio) than if a default configuration is known and only variations from this need be signalled. However, a major determinant of configuration is the language in which the original message to be compressed is composed. For example, different keyword dictionaries would be required for French and opposed to German and character frequency distributions for English texts may vary greatly from those for Swedish texts. From this it can be seen that a universal "default" configuration would be of little value.

To address this, the Compression Language Context (CLC) allows a default configuration to be specified for each of the languages defined in 3GPP TS 23.038 [1] in relation to the Cell Broadcast Data Coding Scheme as follows:

- The CLC in bits 6 to 3 of the CH specify the language as per 3GPP TS 23.038 [1] in the case where bits 7 to 4 of the Cell Broadcast Data Coding Scheme octet are set to 0000.

- If and when required, higher order bits of the CLC can be signalled by a subsequent CH octet as described below.
- The CLC value 1111 (language unspecified) will indicate a "default" configuration that is language independent. This is specified in annex R and involves the basic Huffman (de-)coding with no initial character frequency distribution, see example below.

Table 2: Huffman (de-)coding with no initial character frequency distribution

7	6	5	4	3	2	1	0	Description
0	1	1	1	1	0	0	0	Basic Huffman (de-)coding only.

5.2.2 Compression Header - Octets 2 to n

Any second and subsequent CH octets are used to vary the configuration defaults established by the CLC. These octets all comprise a continuation bit followed by a Type, Value structure as follows:

Table 3: Value structure

7	6	5	4	3	2	1	0	Description
0								There is no subsequent CH octet
1								A further CH octet follows
	n	n	n					CH Extension Type
				n	n	n	n	CH Extension Value

The bits of the semi-octet CH Extension value are interpreted left to right, MSB to LSB. If the CH contains more than one octet of the same CH Extension type, the CH Extension value of a subsequent CH octet, is interpreted as being next most significant semi-octet of the composite value being signalled.

For example if the CLC in CH octet 1 indicates that the default Huffman Initialization ID is 1 (decimal) and the required HI-ID is 37 (decimal), then the following octets (in the range 2 to n) would also be required in the CH.

Table 4: CH extension octets (Example)

7	6	5	4	3	2	1	0	Description
1	0	1	1	0	1	0	1	The default HI-ID is replaced with the value 0101
0	0	1	1	0	0	1	0	The current HI-ID value (0101) is extended to 0010 0101

The following values are defined for the CH Extension Type:

000 Extend CLC. The CH Extension Value contains higher order bits that are to be pre-pended to the current CLC value.

NOTE: for 1st occurrence of the Extend CLC CH Extension Type in the CH, the value for the CLC specified in CH octet 1 is *not* replaced but rather the process of "extension" begins directly. Thus is the CLC to be used is 18, octets 1 and 2 of the CH would contain:

Table 5: CLC extension (Example)

7	6	5	4	3	2	1	0	Description
1	0	0	1	0				The least significant semi-octet of the CLC is 0010

0	**0**	**0**	**0**	**0**	**0**	**0**	**1**	The CLC value (0010) is extended to 0001 0010

001 Change Character Set. The CLC defines a default character set (UCS2 or otherwise) within which compression will operate. The Change Character Set CH Extension Type indicates that this should be overridden by the character set specified by the CH Extension Value. If a CH contains more than one Change Character Set CH Extension Type octet, the CH Extension Value contained in subsequent CH octets of this type contains higher order bits and are to be pre-pended to the value of the new character set.

The following Character Sets are defined:

0000 No character set defined. To be used where original message content is binary data and compression is solely via Huffman coding with no initial frequency training and thus there is no requirement to ensure consistent use of character set by coder and decoder.

0001 GSM 7 bit default alphabet (3GPP TS 23.038 [1])

0010 Codepage 437

0011 Codepage 850

All other values are reserved - see section 5.2.2.1

A Change Character Set to UCS2 codepoint is not defined here. Where the CLC indicates a character set other than UCS2 and there is a need to change to UCS2 then this is achieved using the Change UCS2 row parameter described below.

010 Change UCS2 Row. The CLC defines a default character set (UCS2 or otherwise) within which compression will operate. The Change UCS2 Row CH Extension Type indicates that this should be overridden by the use of UCS2 *and* the UCS2 row value for the first character in the input stream is that specified by the CH Extension Value. If a CH contains more than one Change UCS2 Row CH Extension Type octet, the CH Extension Value contained in subsequent CH octets of this type contains higher order bits for the initial UCS2 Row value and are to be pre-pended to the current value.

NOTE: Change UCS2 Row CH Extension Type octet effectively overrides any prior Change Character Set CH Extension Type octet and vice versa so these types are logically mutually exclusive within a given CH.

011 Change Huffman Initialization. The CLC defines a default set of parameters for the initialization of the Huffman (de)coder. The Change Huffman Initialization CH Extension Type indicates that this should be overridden by the set of initialization parameters identified by the Huffman Initialization ID contained in the CH Extension Value. If a CH contains more than one Change Huffman Initialization CH Extension Type octet, the CH Extension Value contained in subsequent CH octets of this type contains higher order bits for the initial Huffman Initialization ID value and are to be pre-pended to the current value.

100 Change Keyword Dictionary. The CLC defines a default set of parameters for the initialization of the Keyword (de)coder. The Change Keyword Dictionary CH Extension Type indicates that this should be overridden by the set of initialization parameters identified by the Keyword Dictionary ID contained in the CH Extension Value. If a CH contains more than one Change Keyword Dictionary CH Extension Type octet, the CH Extension Value contained in subsequent CH octets of this type contains higher order bits for the initial Keyword Dictionary ID value and are to be pre-pended to the current value.

101 Change Punctuator. The CLC defines a default set of parameters for the initialization of the punctuation (de)coder. The Change Punctuator CH Extension Type indicates that this should be overridden by the set of initialization parameters identified by the Punctuator ID contained in the CH Extension Value. If a CH contains more than one Punctuator CH Extension Type octet, the CH Extension Value contained in subsequent CH octets of this type contains higher order bits for the initial Punctuator ID value and are to be pre-pended to the current value.

110 Change Character Group. The CLC defines a default set of parameters for the initialization of the Character Group (de)coder. The Change Character Group CH Extension Type indicates that this should be overridden by the set of initialization parameters identified by the Character Group ID contained in the CH Extension Value. If a CH contains more than one Change Character Group CH Extension Type octet, the CH Extension Value contained in subsequent CH octets of this type contains higher order bits for the initial Character Group ID value and are to be pre-pended to the current value.

111 Reserved, see section 5.2.2.1

5.2.2.1 Compression Header reserved extension types and values

Any currently undefined values in the range 0 to 255 decimal are reserved.

Values above 255 are available for user to user requirements.

5.2.3 Identifying unique parameter sets

The four component compression algorithms (Huffman, Keywords, Character Groups and Punctuation) may all have a variety of initialization options. For each algorithm, a given set of initialization options needs to be identified for the processing of a given input stream.

Initialization and operation of the algorithms depends not only on the language in which the original source text is composed but also the character set (UCS2 or otherwise) that is to be used during processing. Thus the Huffman Initialization ID (HI-ID), Keyword Dictionary ID (KD-ID), Punctuator ID (PU-ID) and Character Group ID (CG-ID) only define unique values within the context of a given character set (the default established by the CLC or subsequently amended via Change Character Set or Change UCS2 Row CH Extension types) and within the context of the language indicated by the CLC.

5.3 Compressed Data

The Compressed Data (CD) is a stream bits of variable length that represent either an encoding of the content original input stream or control information indication that the operation of some algorithm should vary in some manner.

Control information is signalled within the CD by Huffman encoded symbols (characters) whose value is greater than 255 decimal. Huffman encoded symbols in the range 0 to 255 are of course characters from the original input stream.

The following control symbols are defined:

Table 6: Compressed Data: control symbols

Decimal value	Significance
256	New 7 bit character. On encoding, if a character (octet) from the input stream in the range 0 to 127 does not exist in the Huffman tree, then the New 7 bit character symbol is Huffman encoded to the CD and bits 6 to 0 of the original octet are copied unchanged to the CD. The Huffman tree would then be updated to include the new character as described in the sections below. On decoding the New 7 bit character symbol, the symbol itself is discarded and the next 7 bits of the CD are copied unchanged to bits 6-0 of the octet to be output, bit 7 of which is zero. The Huffman tree would then be updated to include the new character.
257	New 8 bit character. The operation of this is identical to that of the New 7 bit character except that on encoding, the input character is in the range 128-255 and on decoding, bit 7 of the output character is set to 1.
258	Keyword. This symbol (Huffman encoded) prefixes a sequence of bits of variable length in the CD that define a representation of characters in the uncompressed stream by an entry in a keyword dictionary. On encoding, if a sequence of characters in the input stream can be represented by an entry in a keyword dictionary, the Keyword symbol is Huffman encoded to the CD followed by the bit sequence describing the keyword entry (this is described below). On decoding the Keyword symbol, the symbol itself is discarded and the bit sequence describing the keyword entry is passed to the Keyword processor to recovery the original character sequence to be placed in the output stream.
259 to 265	Character Group Transitions. These symbols signal transitions between groups of characters defined within the Character Group processor. For example, if 2 groups are defined to be the lower case and upper case characters then the input stream: "abcdefABCDEF" would become "abcdef<Change Group>abcdef" On encoding, Character Group Transition symbols are generated by the Character Group processor and simply passed to the Huffman processor for encoding. On decoding a Character Group Transition symbol, it is simply passed from the Huffman processor to the Character Group processor which takes the appropriate action based its current state and the group transition indicated.
266	New UCS2 Row. On encoding, if the next UCS2 character in the input stream has a "row octet" of a different value to that of the previous character in the input stream, the New UCS2 Row symbol is Huffman encoded to the CD and the 8 bit of the new row octet are copied unchanged to the CD. The new row octet is stored by the UCS2 processor as the "current row octet" and subsequent input characters within the current row are Huffman encoded as the 8 bit value of the character *within* the "current row". On decoding the New UCS2 Row symbol, the symbol is discarded and the next 8 bits are read from the CD and stored by the UCS2 processor as the "current row octet". Subsequent UCS2 characters are decoded by treating the 8 bit character values decoded by the Huffman processor as characters *within* the "current row".

5.4 Compression Footer

Although Compressed Data Stream Length (CDSL) - the total number of octets that contain the CDS - is known, the CD element of the CDS is a bit stream and therefore may not end on an octet boundary. The Compression Footer (CF) is used to indicate the end of the CD as follows:

- Calculate the number of meaningful bits in the last octet of the CD (i.e. total CD bits modulo 8).
- If the number of meaningful bits is >0 and <6 store the number of meaningful bits in bits 2 to 0 of the last octet. Otherwise extend the CD by adding 1 octet and store the number of meaningful bits in bits 2 to 0 of this new octet. In the case where the number of meaningful bits is 8 then bits 2 to 0 of the new octet are set to zero.

For example if there are 4 meaningful bits in the last CD octet, the CF will be constructed to occupy the shaded area in table 7.

Table 7: CF with >0 and <6 meaningful bits in last octet (Example)

0	7	6	5	4	3	2	1	0
X	X	X	X	X		1	0	0

Alternatively if there are 6 meaningful bits in the last CD octet, a new octet needs to be added. The CF will be constructed to occupy the shaded area in table 8.

Table 8: CF with >5 meaningful bits in last octet (Example)

0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X								1	1	0

If there are 8 meaningful bits in the last CD octet, a new octet needs to be added. The CF will be constructed to occupy the shaded area in table 8a.

Table 8a: CF with 8 meaningful bits in last octet (Example)

0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X						0	0	0

In all the tables above, the bits in the shaded area which have no bit value defined are set according to the particular bearer being used to transport compressed data. e.g. CBS. Where no particular reference is made regarding the value of those bits they may be set to any value.

6 Compression processes

This clause defines the detailed operation of the various compression algorithms.

6.1 Overview

This subclause describes how the various compression algorithms are combined.

6.1.1 Compression

Table 9: Compression

Input	1) The nature of the compression to be performed. 2) The input stream of characters to be compressed.
Step 1	Construct the Compression Header so as to fully describe the nature of the compression to be performed as requested by higher software layers. Note that it is the responsibility of higher software layers that use the compression algorithms to ensure that only those aspects of the compression algorithms that are supported by a particular implementation are requested.
Step 2	Initialize as defined by the CH the following components: 1) Character Set Converter 2) Punctuation Processor 3) Keyword Processor 4) UCS2 Processor 5) Character Group Processor 6) Huffman Processor
Step 3	If the Character set in which input stream is composed is different from that specified in the CH, convert the input stream so that it is rendered in the Character set (UCS2 or otherwise) specified in the CH. Note that if characters in the input stream cannot be rendered in the character set specified in the CH, it is the responsibility of higher software layers that use the compression algorithms to detect this situation and take appropriate action.
Step 4	If the Punctuation Processor is enabled, use it to encode the character set converted input stream produced by Step 3 above.
Step 5	Set the current character position to the start of the character stream produced as the output of Step 4 above.
Step 6	If the Keyword processor is *not* enabled goto Step 7. Examine the sequence of characters starting at the current character position in the input stream and determine if they can be represented by an entry in the keyword dictionary. If an appropriate keyword is *not* found goto Step 7. If the Character Group processor is enabled, pass it the Keyword symbol and Huffman encode to the CD the sequence of symbols output by it. Huffman encode the Keyword symbol to the CD and then copy the bit sequence describing the keyword entry to the CD. Goto Step 10.
Step 7	If the input stream is *not* UCS2 goto Step 8. If the character at the current character position in the input stream has a different UCS2 row octet from the previous character Huffman encode the New UCS2 Row symbol to the CD and then copy the new row octet to the CD. Remove the row octet from the character at the current character position in the input stream which will subsequently treated as an 8 bit value.
Step 8	If the Character Group processor is *not* enabled goto Step 9. Pass the character at the current character position in the input stream to the Character Group processor and Huffman encode to the CD the sequence of symbols output by it. Goto Step 10.
Step 9	Huffman encode the character at the current character position in the input stream.
Step 10	Increment the current character position by the number if input characters processed in steps 6 to 9 above. If the entire input stream has *not* been processed goto Step 6 above.
Step 11	Construct the Compression Footer.
Output	The completed Compressed Data Stream. Note that the possibility exists that the CDS may be larger than the original input stream. In this case it is the responsibility of higher software layers that use the compression algorithms to detect this situation and take appropriate action.

6.1.2 Decompression

Table 10: Decompression

Input	The Compressed Data Stream
Step 1	Interpret the Compression Header to determine the nature of the decompression to be performed. Note that it is the responsibility of higher software layers that use the decompression algorithms to handle appropriately the case where the nature of the decompression to be performed is not supported by a particular implementation.
Step 2	Initialize as defined by the CH the following components: 1) Character Set Converter 2) Punctuation Processor 3) Keyword Processor 4) UCS2 Processor 5) Character Group Processor 6) Huffman Processor
Step 3	Interpret the Compression Footer to determine the total number of significant bits in the Compressed Data (CD). Set the total number of bits processed to zero.
Step 4	Read bits from the CD passing them to the Huffman decoder to generate the "current symbol". The bits should be read in the order bit 7 to bit 0 within each CD octet. CD octets are processed in the order 1 to n.
Step 5	If the Keyword processor is *not* enabled, goto Step 6. If the "current symbol" is the Keyword symbol, read the bit sequence describing the keyword entry from the CD. Pass the keyword entry description to the Keyword processor for decoding and add the resulting sequence of characters representing the keyword to the output stream. Goto Step 9.
Step 6	If the Character Group processor is *not* enabled goto Step 7. If the "current symbol" is a Character Group Transition symbol, pass it to the Character Group processor so that the current group can be updated and goto Step 9. If the value of the "current symbol" is in the range 0 to 255 (i.e. not a control symbol), pass the "current symbol" to the Character Group processor and set the new value of the "current symbol" to that returned by the Character Group processor.
Step 7	If the output stream is *not* UCS2 goto Step 8. If the "current symbol" is the New USC2 Row symbol, read the new "current UCS2 row octet" from the CD and goto Step 9. Pre-pend the "current UCS2 row octet" to the 8 bit value of the "current symbol" to produce a 16 bit UCS2 character.
Step 8	Add the "current symbol" to the output stream.
Step 9	Increment the total number of bits processed by the number of bits read from the CD in steps 4 to 8 above. If the total number of bits processed is less than the total number of significant bits in the CD goto Step 4.
Step 10	If the Punctuation Processor is enabled, use it to decode output stream produced by steps 3 to 9 above.
Step 11	If the Character set (UCS2 or otherwise) specified in the CH, is different from that required by higher level software layers, convert the output stream produced by step 10 above so that it is rendered in the Character set (UCS2 or otherwise) required by higher level software layers. Note that if characters in the stream cannot be converted, it is the responsibility of higher software layers that use the compression algorithms to detect this situation and take appropriate action.
Output	The decompressed original input stream.

6.2 Character sets

The need for character set conversion arises in that a number of the compression algorithms operate on the basis of "prior information" about the nature of human readable texts. For example Huffman frequency initializations may specify the an initial relative frequency for the letter "e" as opposed to the letter "x". Similarly, a keyword dictionary may contain the word "meeting".

Consider the case where a keyword dictionary contains the entry "£10,000" composed using the Code Page 850 character set. If an input stream containing the string "£10,000" also composed in Code Page 850 is processed, the string will be replace in the CD by a reference to the keyword entry. In contrast if the input string is composed using the GSM 7 bit default alphabet (3GPP TS 23.038 [1]) than a match between the input string and the keyword entry will not be found as the value of the "£" symbol in Code Page 850 is 156 decimal whereas in the GSM 7 bit default alphabet it is 2 decimal.

There can be no assumption that higher level software layers responsible for composing the original input stream to be compressed and displaying the resulting decompressed output stream use the same character set.

Thus:

- The character set used to compose initialization parameter sets and used for the compression of a given input stream shall be the same for both compression and decompression.
- Where an input stream is composed using a character set that is different from that used for compression it shall be converted prior to compression.
- Where an output stream is required in a character set that is different from that used for compression it shall be converted after decompression.

There is an additional requirement in that a number of the compression algorithms perform upper / lower case conversions upon the characters within the character set used for compression. The mapping between "lower" and "upper" case characters needs to therefore be known.

6.2.1 Initialization

Initialization of character set conversion processing will typically involve identifying and loading the appropriate tables to a) convert between character sets and b) convert between upper and lower case characters.

As the character set(s) in which uncompressed data is required to be rendered is largely an implementation specific matter, so is the precise specification of the tables to convert these to/from the character set specified for compression. However, they need to be sufficient to support the following functions:

6.2.2 Character set conversion

Table 11: Character set conversion

Input	1) The value of the source character. 2) The character set in which the source character is rendered. 3) The character set in which the source character is to be rendered.
Output	1) The value of the converted character. 2) A Boolean value indicating whether a successful conversion has been performed.
Process	If the source character can be rendered in the target character set, its value in the target characterset is returned and a successful conversion is indicated. Otherwise, the value of the source character is returned unchanged, a conversion failure is indicated and higher software layers need to take appropriate action. For example: - The character "A", 65 decimal in Code Page 850 is rendered in the GSM 7 bit default alphabet also as 65 decimal so this value is returned and a successful conversion is indicated. - The character "£", 156 decimal in Code Page 850 is rendered in the GSM 7 bit default alphabet as 1 decimal so the value 1 is returned and a successful conversion is indicated. - The character "Û" 234 decimal in Code Page 850 cannot be rendered in the GSM 7 bit default alphabet so the value 234 is returned unchanged and a conversion failure is indicated.

6.2.3 Character case conversion

Conversion between upper and lower case for characters within the character set used for compression will also typically be supported by conversion tables that indicate for each character in the character set, the value of any lower case or upper case equivalent character such that the following function can be supported.

Table 12: Character case conversion

Input	1) The value of the source character. 2) The case (lower or upper) in which the source character is to be rendered.
Output	1) The value of the case converted character.
Process	If the character can be rendered in the case requested and the value of this case converted character is different from that of the source character, the value of the case converted character is returned. Otherwise (i.e. the source character is already in the requested case or the character does not have upper and lower case equivalents), the value of the source character is returned unchanged.

6.3 Punctuation processing

The punctuation processor achieves compression by using the "prior information" that the uncompressed stream is human readable and is constructed of sentences that conform to a known set of punctuation rules. Essentially this means that certain characters within the input stream, of themselves imply information about subsequent characters and this may therefore be omitted from the compressed stream. In this way the algorithm achieves some significant compression in a very simple manner.

However, because the algorithm operates on information about sentence structure rather than the exact sequence of characters used to render this, it is non-symmetric. In other words, although the overall meaning of the human readable input stream is preserved between compression and decompression, the exact sequence of characters is not. Higher level software layers or even user inspection may therefore be required to determine if the use of this processor is appropriate for a given input stream.

In addition to the ability to handle the conversion of characters between upper and lower case (as described in the previous subclause), the processor requires that certain characters (expressed in the character set to be used for compression) are assigned special attributes. These are:

Table 13: special attributes

Attribute	Description
PU-IWS	Inter-word separator. A character with this attribute is that typically used to separate words within the input stream. Only one character in the character set may have this attribute. This attribute is typically set for the "space" character (32 decimal).
PU-LST	Last Sentence Terminator. A character with this attribute is that typically used to terminate the last sentence in the input stream. Only one character in the character set may have this attribute. This attribute is typically set for the "." full stop character (46 decimal).
PU-WSF	Word Separator Follows. A character with this attribute is expected to be followed by one or more characters which have the PU-IWS attribute set. Any number of characters within the character set may have this attribute. Examples of characters that would normally have this attribute set are the exclamation mark (!), comma (,), full stop (.), colon (:), semi-colon (;) and question mark (?).
PU-UCF	Upper Case Follows. A character with this attribute is expected to be followed by an upper case character such as occurs at the start of a sentence or paragraph. Any number of characters within the character set may have this attribute. Typically, characters with this attribute set will also have the PU-WSF attribute set. Examples are the exclamation mark (!), full stop (.), and question mark (?). Other examples associated with new paragraphs might include the carriage return (13 decimal) and line feed (10 decimal) symbols.
PU-UCW	Upper Case Word. A character with this attribute set is expected to be upper case if it is a word i.e. if it is both preceded and succeeded by character with the PU-IWS attribute set. Any number of characters within the character set may have this attribute. An example in the English language is the letter "I".
PU-NSI	No Separator Insertion. A character with this attribute set is does not have the PU-IWS attribute set but is none the less expected to be preceded by a character for which the PU-WSF attribute is set. Any number of characters within the character set may have this attribute. Typically, characters with this attribute set will be numeric digits so that the case can be resolved where characters which have the PU-WSF attribute set such as comma (,) and full stop (.) can be used in number formatting as in the case of the string "£10,000.25".

6.3.1 Initialization

Initialization of the punctuation processor will typically involve loading a table containing the combination of attributes defined for each character in the character set to be used for compression for the language defined by the CLC.

6.3.2 Compression

For compression, the punctuation processor operates as follows:

Table 14: compression punctuation processor

Input	The input stream of characters to be compressed, rendered in the appropriate character set.
Step 1	Set the current character position to the start of the input stream.
Step 2	Determine the attributes of the current character. If some previous character in the input stream has not had the PU-IWS attribute set goto Step 3. If the current character has the PU-IWS attribute set goto Step 8. Convert the current character to lower case and store the returned value as that of the "previous character". Store the attributes of the current character as those of the "previous character" after clearing any PU-UCW attribute. Goto Step 8.
Step 3	If the previous character has the PU-WSF attribute and the current character has the PU-IWS attribute goto Step 8. Otherwise clear the PU-WSF attribute for the "previous character".
Step 4	If the previous character has the PU-UCF attribute, convert the current character to lower case and clear the PU-UCF attribute for the "previous character".
Step 5	If the previous character has the PU-UCW attribute and the current character has the PU-IWS attribute, convert the previous character to lower case.
Step 6	If the previous character has the PU-IWS attribute and the current character has the PU-IWS attribute, goto Step 8. Otherwise add the previous character to the output stream and set the value of the previous character to that of the current character.
Step 7	If the current character has the PU-UCW attribute and the previous character attributes do not contain the PU-IWS attribute, clear the PU-UCW attribute for the current character. Set the attributes for the "previous character" to those of the current character.
Step 8	If the current character is the last character in the input stream *and* if some previous character in the input stream has not had the PU-IWS attribute set *and* if the previous character attributes contain neither the PU-IWS not the PU-LST attribute, add the previous character to the output stream.
Step 9	If the current character is *not* the last character in the input stream, read the next character from the input stream, set the current character to this value and goto Step 2.
Output	The de-punctuated data stream.

6.3.3 Decompression

For decompression, the punctuation processor operates as follows:

Table 15: decompression punctuation processor

Input	The de-punctuated stream of characters to be punctuated, rendered in the character set used for compression.
Step 1	Set the current character position to the start of the de-punctuated stream.
Step 2	Determine the attributes of the current character. If the current character is the first character in the stream, convert it to upper case and goto Step 8.
Step 3	If the current character has the PU-IWS attribute and the "previous character" attributes has the PU-UCW attribute, convert the stored value of the "previous character" to upper case.
Step 4	If the "previous character" attributes contain the PU-UCF attribute, and the current character was not generated by Step 10 below, convert the current character to upper case and clear the PU-UCF attribute for the "previous character" attributes.
Step 5	If the "previous character" was generated as a result of Step 10 and the current character contains the PU-NSI attribute goto Step 7.
Step 6	Add the "previous character" value to the output stream.
Step 7	If "previous character" attributes contain the PU-IWS attribute and the current character has the PU-UCW attribute, add the PU-UCW attribute to those of the "previous character". Otherwise clear any PU-UCW attribute stored for the "previous character".
Step 8	Set the value of the "previous character" to be that of the current character.
Step 9	If the attributes of the current character contain the PU-UCF attribute set this attribute for the "previous character".
Step 10	If the attributes of the current character contain the PU-WSF attribute and the current character is not the last character in the de-punctuated stream, insert the character containing the PU-IWS attribute at the position following the current character in the de-punctuated stream.
Step 11	If the current character is *not* the last character in the de-punctuated stream, read the next character from the stream, set the current character to this value and goto Step 2.
Step 12	Add the previous character to the output stream. If the current character attributes do not contain the PU-UCF attribute or the previous character value equals that of the character which has the PU-LST attribute set, add the character which has the PU-LST attribute set to the output stream.
Output	The punctuated data stream.

6.4 Keywords

The operation of the Keyword processor is controlled by the set of parameters defined by a Keyword Dictionary that is uniquely defined (within a CLC) by the value of the Keyword Dictionary ID (KD-ID) specified in the CH.

6.4.1 Dictionaries

A Keyword Dictionary specifies the following items:

1) Character Set ID

   This is the character set in which the dictionary is composed and shall therefore be equal to the character set to be used for compression as specified in the CH.

2) Match Options

   This is a collection of bit flags that control how text in the input stream is to be matched against key word dictionary entries. These are described in the table below in which Bit 0 is considered to be the lease significant bit of the Match Options value.

Table 16: Match options

Bit	Description
0	If set, input stream text shall exactly match the dictionary entry.
1	If set, input stream text may match the lower case conversion of a dictionary entry.
2	If set, input stream text may match the upper case conversion of a dictionary entry.
3	If set, input stream text may match the upper case conversion of the 1st character of a dictionary entry followed by the lower case conversion of the remaining characters of the dictionary entry.
4	If set, input stream text may match a dictionary entry prefixed by the keyword prefix characters (if any) described below.
5	If set, input stream text may match a dictionary entry suffixed by the keyword suffix characters (if any) described below.
6	If set, input stream text may match a part of a dictionary entry. A partial match occurs when, a dictionary entry contains n characters and a match is found with the first m characters where m is less than n.
7-	All other bits are reserved.

3) Keyword Prefix

   The 1st octet is the Keyword Prefix Length which specifies the number of characters that form the prefix string. The length octet is followed by the actual characters of the prefix string.

4) Keyword Suffix

   The 1st octet is the Keyword Suffix Length which specifies the number of characters that form the suffix string. The length octet is followed by the actual characters of the suffix string.

5) Keyword Threshold

   This value determines the minimum number of characters in the input stream that needs to be replaced by a full match with a keyword entry. For a partial match the value of the threshold needs to be incremented by 2.

   If a match occurs involving fewer characters than that specified by the threshold, keyword substitution does not take place.

6) Maximum Partial Match Length

   This value determines the maximum number of characters in the input stream that needs to be replaced by a partial match with a keyword entry.

   If a partial match occurs involving fewer characters than that specified by this value, keyword substitution does not take place.

7) Key Word Group List

   The actual key word dictionary entries are not directly specified within the Keyword Dictionary. Instead, a set of key word dictionary entries is explicitly identified by a Key Word Group ID - an octet value that is unique within the language specified by the CLC. This approach allows the same set of keyword dictionary entries to be used in conjunction with different values for the parameters specified within the Keyword Dictionary and for Keyword Dictionaries to be defined that combine multiple Key Word Groups.

   The 1st octet of the Key Word Group List specifies the number of Key Word Group IDs that follow, each of the following octets specifies a Key Word Group ID.

6.4.2 Groups

A Keyword Group specifies the following items:

1) Character Set ID

   This is the character set in which the keyword dictionary entries are composed and shall therefore be equal to the character set to be used for compression as specified in the CH.

2) Number of Entries

   The value specifies the number of keyword dictionary entries contained in the Keyword Group.

3) Keyword Entry

   The 1st octet is the Keyword Entry Length which specifies the number of characters that form the keyword entry string. The length octet is followed by the actual characters of the entry string.

The sequence of entries within a dictionary needs to be known by both coder and decoder. Thus keyword entries in a Keyword Group needs to be sorted in ascending sequence of the actual characters of the entry string. Furthermore if a dictionary defines multiple Keyword Groups, the combined set of entries needs to be resorted as part of initialization of the Keyword processor so that the ascending alphanumeric sequence of entries is achieved for all entries in the combined set.

A further requirement is that all entries in the combined set shall be unique.

6.4.3 Matches

A Keyword Match specifies how a sequence of characters in the input stream is represented by a keyword dictionary entry. A Keyword Match is a bit stream that is interpreted left to right as described on the table below wherein Bit 0 refers to the most significant, left most bit.

Table 17:

Bits	Description
0 to N_1	Case conversion. If bit 0 of the Dictionary Match Options is set (i.e. Exact matching is enabled), the Case conversion bits are omitted and the Keyword Match starts with the Keyword Entry ID described below. Otherwise, if the match involves a lower case conversion, a single Case conversion bit with value 0 is used. Otherwise, 2 case conversion bits are used with the following value: 10 Upper Case. 11 1st character Upper case, remainder Lower case.
N_1+1 to N_2	Keyword Entry ID. This value represents the position in the list of keyword dictionary entries of the entry with which a match has been found. A value of 0 indicates the first entry. The number of bits used to express the Keyword Entry ID is minimum number of bits required to represent the total number of keyword dictionary entries defined for the Keyword Dictionary minus 1.
N_2+1 to N_3	Prefix Match. If bit 4 of the Dictionary Match Options is set (i.e. Prefix matching is enabled), a single bit is used to indicate whether a prefix match applies (1) or not (0). If prefix matching is not enabled, this bit is omitted from the Keyword Match.
N_3+1 to N_4	Partial Match. If bit 6 of the Dictionary Match Options is set (i.e. Partial matching is enabled), a single bit is used to indicate whether a partial match has occurred (1) or not (0). If partial matching is not enabled, this bit is omitted from the Keyword Match. If partial matching is enabled and a full match has occurred, no further bits are required to describe the match. If partial matching is enabled and a partial match has occurred, it is necessary to encode the length of the partial match as follows: The partial match length equals the total number of characters in the input stream represented by the Keyword Match (excluding any characters represented by any prefix and suffix matches) less the value of the partial match threshold (i.e. Keyword Threshold +2). If the partial match length is less than 8 a single bit (0) is added to the bit stream to indicate this fact followed by 3 bits containing the partial match length. Otherwise a single bit (1) is added to the bit stream to indicate that more than 3 bits follow containing the partial match length. In this case the number of bits used to represent the partial match length is the minimum number of bits required to represent the value (Maximum Partial Match Length - (Keyword Threshold +2))
N_4+1 to N_5	Suffix Match. If bit 5 of the Dictionary Match Options is set (i.e. Suffix matching is enabled), a single bit is used to indicate whether a suffix match applies (1) or not (0). If suffix matching is not enabled, this bit is omitted from the Keyword Match.

6.4.4 Initialization

Initialization of the Keyword processor involves loading the various parameters specified by the KD-ID contained in the CH.

As noted above, if the dictionary is composed on more than 1 Keyword Group, the combined set of keyword entries needs to be resorted so that the full set conforms to an ascending alphanumeric sequence.

Clearly,as it is the total combined and sorted set of keyword entries that is required, implementors may choose to construct this from the component keyword groups at run time or to produce such a combination and use it directly as indicated by the constituent keyword group ID's.

6.4.5 Compression

For compression, the Keyword processor operates as follows:

Table 18: compression Keyword processor

Input	A offset into the input stream of characters from which a matching keyword is to be found.
Step 1	Set the current character position to the input offset.
Step 2	If Prefix matching is not enabled goto Step 3. If the string starting at the current character position exactly matches Keyword Prefix, record this fact and increment the current character position by the length of the prefix string.
Step 3	Identify the Keyword Entry ID and if enabled Case Conversion and Partial Match details for the longest match (i.e. that what whereby the greatest number of characters in the input stream are represented) between a dictionary entry and the string starting at the current character position subject to the following rules: 1) An exact match shall be greater than or equal to the Keyword Threshold to be considered. 2) A partial match shall be greater than or equal to the Keyword Threshold +2 to be considered. 3) If more than 1 partial match of equal length is found, the one with the greater Keyword Entry ID is used. 4) If an exact match and a partial match are found, the length of the partial match shall be at least 2 greater than that of the exact match for it to be used. 5) Although the case of more than 1 exact match of equal length being found is not possible as entries are unique, should such a case arise, the one with the greater Keyword Entry ID is used. If the longest match is a partial match with length greater than the Maximum Partial Match Length, the match length is limited to the Maximum Partial Match Length. If no match has been found goto Step 5.
Step 4	If Suffix matching is not enabled goto Step 5. If the string starting at the current character position exactly matches Keyword Prefix, record this fact and increment the current character position by the length of the prefix string.
Step 5	If a matching keyword has been found, construct the Keyword Match bitstream.
Output	A Keyword Match bitstream or an indication that no suitable match is available.

6.4.6 Decompression

For decompression, the Keyword processor operates as follows:

Table 19: decompression Keyword processor

Input	A Keyword Match bitstream.
Step 1	Interpret the Keyword Match bitstream to determine if there is a Prefix match. If so add the Keyword Prefix string to the string to be output.
Step 2	Interpret the Keyword Match bitstream to identify the dictionary entry or part thereof as indicated by any Partial Match details. Perform any case conversion (indicated by the Keyword Match bitstream) on the dictionary entry string and add the resulting string to the string to be output.
Step 3	Interpret the Keyword Match bitstream to determine if there is a Suffix match. If so add the Keyword Suffix string to the string to be output.
Output	The character string represented by the input Keyword Match bitstream.

6.5 UCS2

6.5.1 Initialization

Initialization of the USC2 processor involves storing the default UCS2 row as specified by the CH.

6.5.2 Compression

For compression, the UCS2 processor operates as follows:

Table 20:

Input	A 16 bit UCS2 character value.
Step 1	If the row octet of the input character is different from the "current UCS2 row" store the row octet of the input character as the new "current UCS2 row".
Output	A Boolean value indicating whether the current UCS2 row has been changed.

6.5.3 Decompression

For decompression, the USC2 processor needs to set and sense the "current UCS2 row" as required by the higher level software described in subclause 6.1.2 above.

6.6 Character group processing

The operation of the Character Group processor is controlled by the set of parameters defined by a Character Group that is uniquely defined (within a CLC) by the value of the Character Group ID (CG-ID) specified in the CH.

Character grouping operates by defining 2 or more subsets (groups) of characters within the character set used for compression with the following properties:

- Each sub set contains the same number of characters.
- One subset (referred to as Group 0 or the "base group" contains the characters expected to have higher frequencies in a input stream than those of the characters in other subsets.
- Input stream are expected to contain contiguous sequences of characters belonging to a single group.

Compression is achieved by assigning a 1:1 mapping between the characters in the base group and those in the other groups and when appropriate signalling a transition between groups and then continuing to encode base group

characters. This has the effect of improving the performance of the Huffman encoder by reducing the need to add new characters to the tree and by maintaining a smaller overall tree with a more distinct frequency distribution.

For example, assume that we have a character set that comprises just the numeric digits 0 to 9 and the letters A to B and 3 groups containing the digits 1 to 3, 4 to 6 and 0 and 7 to 9. The digits 1 to 3 are considered to be the most frequent and are therefore the base group. The digit 0 is defined to exist in all the groups and the letters A and B do not occur in any group.

Encoding and decoding of characters is achieved using the various items in table 21.

Table 21: Encoding and decoding of characters

Item	Element												Comment
Value	0	1	2	3	4	5	6	7	8	9	10	11	Decimal character value
Character	0	1	2	3	4	5	6	7	8	9	A	B	Character symbol
Group 0	1	1	1	1	0	0	0	0	0	0	0	0	Bit flags for Group 0
Group 1	1	0	0	0	1	1	1	0	0	0	0	0	Bit flags for Group 1
Group 2	1	0	0	0	0	0	0	1	1	1	0	0	Bit flags for Group 2
Fold 0	0	1	2	3	1	2	3	1	2	3	A	B	Group 0 Conversions
Fold 1	0	4	5	6	4	5	6	7	8	9	A	B	Group 1 Conversions
Fold 2	0	7	8	9	4	5	6	7	8	9	A	B	Group 2 Conversions

The items Group 0, Group 1 and Group 2 simply enable the determination of whether a given character is a member of the given group by checking the value of the Group x element associated with the value of the character.

The elements of the Fold 0 item associated with the members of a given group represent the characters within Group 0 to which the characters of the given group are mapped. For example character 4 in Group 1 is mapped to character 1 in Group 0.

The elements of the Fold 1 and Fold 2 items provide the reverse mapping in that the elements associated with membership of Group 0 represent the characters in Groups 1 or 2 that are associated with the Group 0 characters.

Thus if the "current group" is Group x, a character with value c can be encoded as follows:

- If c is a member of Group x or not a member of any group, element c of Fold 0 is output.
- If c is not a member of Group x it can be output as a "literal" which is element c of Fold y where Group y has c as a member alternatively a change of group can be signalled.

Similarly, if the "current group" is Group x, a character with value c can be decoded as follows:

- If c is a member of Group x or x is not 0 then, element c of Fold x is output.
- Otherwise the value c is output unchanged.

The detailed operation of the Character Group processor (described below) primarily extends these simple rules to optimize the case where a choice between a "literal" or a group change arises.

6.6.1 Character Groups

A Character Group specifies the following items:

1) Character Set ID

   This is the character set in which the character group is composed and shall therefore be equal to the character set to be used for compression as specified in the CH.

2) Number of Groups

   This value specifies the number of groups to be defined. The maximum value is 8.

3) Group Transition Controls

Group transitions are signalled through the use of the Character Group Transition symbols in the decimal range 259 to 265.

If the Number of Groups is N, (N-1) Character Group Transition symbols shall be specified such that if the "current group" is x one Character Group Transition symbol is allocated to signify a transition to each of the other (N-1) groups.

4) Fold Tables

These are the inter-group character conversion tables described above. One is required for each group defined.

5) Group Membership

This is an array of octets, one for each character in the character set. The 1st octet in the array contains bit flags indicating the group membership of the character value 0 and so on.

Within each octet, bit 0 (least significant) indicates membership of Group 0, bit 1 that of Group 1 and so on.

6.6.2 Initialization

Initialization of the Character Group processor involves loading the various parameters specified by the CG-ID contained in the CH.

Additionally on initialization, the "current group" is assumed to be Group 0.

6.6.3 Compression

For compression, the Character Group processor operates as follows:

Table 22: compression Character Group processor

Input	1) A single symbol to be encoded. 2) An indication that this is the last symbol to be encoded.
Step 1	Set the number of output symbols to zero.
Step 2	If the input symbol is not the Keyword symbol, goto Step 3. If a previous input symbol is being held, add this as a "literal" to the output sequence by calculating the value of the element indicated by the value of the previous symbol in the fold table associated with the group of the previous symbol and increment the number of output symbols and clear the previous symbol. Goto Step 9.
Step 3	If the input symbol is a member of no group or a member of the current group, set the group for the input symbol to be the current group. Otherwise, if a previous input symbol is being held and the input symbol is a member of the group of the previous symbol, set the group for the input symbol to be that of the previous symbol. Otherwise, test the input symbol for membership of each group in ascending order of groups starting with group 0 and set the group for the input symbol to be that for which membership is first detected.
Step 4	If a previous input symbol is not being held goto Step 5. If the input symbol group equals the previous symbol group: - Add the Character Group Transition symbol that indicates a transition from the current group to the previous symbol group to the output sequence and increment the number of output symbols. - Set the current group to the previous symbol group. - Encode the previous symbol by calculating the value of the element indicated by the value of the previous symbol in the fold table associated with the base group and add this value to the output sequence and increment the number of output symbols. - Encode the input symbol by calculating the value of the element indicated by the value of the input symbol in the fold table associated with the base group and add this value to the output sequence and increment the number of output symbols. - Clear the previous symbol. - Goto Step 9. Otherwise, encode the previous symbol as a "literal" by calculating the value of the element indicated by the value of the previous symbol in the fold table associated with the group of the previous symbol group and add this value to the output sequence and increment the number of output symbols and clear the previous symbol.
Step 5	If the input symbol group is the base group and the current group is not the base group, add the Character Group Transition symbol that indicates a transition from the current group to the base group to the output sequence and increment the number of output symbols. Set the current group to be the base group.
Step 6	If the input symbol group is the base group or the current group: - Encode the input symbol by calculating the value of the element indicated by the value of the input symbol in the fold table associated with the base group and add this value to the output sequence and increment the number of output symbols. - Goto Step 9.
Step 7	If the input symbol is the last symbol to be encoded: - Encode the input symbol as a "literal" by calculating the value of the element indicated by the value of the input symbol in the fold table associated with the group of the input symbol and add this value to the output sequence and increment the number of output symbols. - Goto Step 9.
Step 8	Set the previous symbol to be the value of the input symbol and set the group for the previous symbol to be that of the input symbol.
Step 9	Output the number of output symbols and the associated symbols.
Output	A count of the number of encoded symbols output and a sequence of encoded symbols.

6.6.4 Decompression

For decompression, the Character Group processor operates as follows:

Table 23: Decompression Character Group processor

Input	A single symbol to be decoded.
Step 1	If the symbol is a Character Group Transition symbol, update the "current group" to be that indicated by the Character Group Transition. Goto Step 3.
Step 2	If the input symbol is a member of the "current group" or the "current group" is not the base group, calculate the value of the decoded symbol as that given by the element indicated by the value of the input symbol in the fold table associated with the "current group". Otherwise set the value of the decoded symbol to that of the input symbol.
Step 3	If a decoded symbol has been generated indicate this fact.
Output	The decoded symbol or an indication that no symbol has been generated.

6.7 Huffman coding

As described in subclause 4.2, Huffman encoding requires the set of characters that may be encoded to be represented within a binary tree structure. The tree is constructed of "nodes" which have the following properties:

- A Parent node. A node that has no parent is the "root" node.
- Up to 2 Child nodes. A node that has no children is a "leaf" node.
- Character value. If the node is a leaf node it represents a character represented within the tree.
- Weight. If the node is a leaf node, the weight is the frequency with which the associated character has occurred in the input stream. Otherwise the weight is simply the sum of the weights of the nodes children.

Typically, a tree will be implemented as an array of node structures and parent / child details for a given node will be represented by the index of the appropriate node within the array.

Every node in the tree (except the root node or in the case where the tree contains just a single leaf node) has a "sibling" - the other node that shares the same parent node.

For the binary tree to be a Huffman tree its construction needs to display a further property. This is that the nodes can be listed in ascending order of weight and in so doing every node is adjacent to its sibling in the list. This property needs to be preserved at all times - when the tree is initially created, when a new leaf node is added to the tree to represent a new character and when the frequency of a leaf node is incremented as a new instance of that character is processed.

The ordering of nodes is also significant in that it will determine which of the siblings is the "left-hand" as opposed to "right-hand" of the sibling pair. Encoding a symbol involves navigating the tree from leaf to root and emitting a bit to the encoded stream the value of which depends on whether the current node is the left or right hand sibling. If the node is a left hand node, the bit value is 0 and if it is a right hand node, the bit value is 1. Assuming that the 1st element of the array of nodes has an index value of 0, this means that left hand nodes will have even numbered indices and right hand nodes will have odd numbered indices.

Node weights are assumed to be 16 bit unsigned values and this means that the potential exists for these values to overflow. To handle this case, the algorithm defines a maximum weight value for the root node. If this is to be exceeded, the weights of all leaf nodes are divided by 2 and the tree is rebuilt. The maximum value for the root weight is defined to be 8000 (hex).

Although the bit sequence representing the encoded symbol is discovered in the order of traversing the tree from leaf to root, for decoding the bit sequence needs to be processed in the order that describes the navigation of the tree from root to leaf. Thus the entire encoding bit sequence needs to be collected in some temporary variable and emitted to the output stream in reverse order. For example if the passage from leaf to root is described by the sequence 010011, the bits added to the output stream would be 110010. The need to collect the bits in a temporary variable also introduces the potential for this value to overflow. Given the maximum value for the root node weight described above, a 32bit variable is suitable of containing all possible bit sequences.

If a symbol that does not already exist in the tree is to be encoded, either the "New 7bit Character" or the "New 8bit Character" is encoded, the lower 7 bits of the new character value are then added literally to the out put stream and the new character needs to be added to the tree. This is done by splitting the "lightest" node (the first node in the list ordered by ascending weight) such that it becomes a parent node whose right hand child is the leaf node that was originally represented by the node being split and the left hand child is a new leaf node representing the new character. The new leaf is initially created with a weight of 0 but this is immediately updated as described below.

If a new symbol has been added to the tree or a new instance of an existing symbol processed, the weight for the associated leaf node needs to be incremented and the tree updated to preserve the "sibling" property.

The tree is updated in the following manner. If the node a position x in the ascending weight ordered list has had its weight incremented by 1, the list needs to be scanned from position x in ascending weight order to identify the node at position y such that the node at position (y+1) is the first node encountered that has a weight greater than or equal to the new weight of the node at position x. The nodes at x and y are then "swapped" in terms of their position in the list and their parents while maintaining all other attributes. This process of weight increment and swapping is then repeated for the parent of the node at position y until the root node is reached.

The operation of the Huffman processor is controlled by the set of parameters defined by a Huffman Initialization that is uniquely defined (within a CLC) by the value of the Huffman Initialization ID (HI-ID) specified in the CH.

6.7.1 Initialization Overview

A Huffman Initialization specifies the following items:

1) Character Set ID

   This is the character set in which the Huffman Initialization is composed and shall therefore be equal to the character set to be used for compression as specified in the CH.

2) Options

   This is a collection of bit flags that control how the processor is to operate. These are described in table 24 in which Bit 0 is considered to be the lease significant bit of the Match Options value.

Table 24: collection of bit flags

Bit	Description
0	If set, weights for leaf nodes representing control symbols (other than New 7 bit character and New 8 bit character symbols) are to be updated.
1	If set, weights for leaf nodes representing control symbols are to be updated.
2	All other bits are reserved.

3) The Character Group ID with which these initializations may operate.

4) Number of initial symbol frequencies

   2 values representing the cases where the Character Group processor is enabled or disabled.

   These are counts of the number of characters or control symbols for which there are following initial frequencies defined.

   As this initializations will vary significantly depending on whether the Character Group processor is enabled 2 sets of initializations are provided to cover both cases.

5) Initial frequencies

Two sets of initialization values are supplied as described above.

Any control symbol that may occur when processing an input stream needs to be represented within the tree, prior to the first character of the input stream being processed. These symbols shall therefore be handled by the initialization process. This is achieved by :

- The frequency initialization *data* will always include all control symbols that *might* occur for any stream. Thus the New 7bit character, New 8bit character, New UCS2 Row and Keyword symbols will always be included and if the initialization set is that for the case where the specified Character Group ID is enabled, the associated Character Group Transition symbols will also be included.

- For a given input stream, the frequency initialization *process* (described in subclause 6.7.2 below) will determine whether a control symbol contained in the frequency initialization *data* can occur in the input stream based on the information contained in the CH. If it is determined that a control symbol contained in the frequency initialization *data* can NOT occur in the input stream, this symbol will not be added to the Huffman tree.

Frequency initialization data comprises the value of the character or symbol and the initial frequency for that symbol.

- The order in which character or symbol values and their associated initial frequencies are stated is significant and this order must be preserved when these items are loaded as part of the Huffman Initialisation process. Frequency Initialisation data must be stated in ascending order of character or symbol initial frequency.

6.7.2 Initialization

Initialization of the Huffman processor involves loading the various parameters specified by the HI-ID contained in the CH.

The appropriate set of frequency initialization data is selected depending on whether the Character Group processor is enabled.

Leaf nodes are created for each symbol for which a frequency initialization is specified, subject to the following rules:

- Leaf nodes must be created within the array of Huffman tree nodes in exactly the same ascending order in which they are stated in the Huffman Initialisation data.

- If the character set specified for compression is the GSM 7 bit default alphabet, leaf nodes are not created for the New 8bit Character and the New UCS2 Row symbols.

- If the character set specified for compression is not UCS2 a leaf node is not created for the New UCS2 Row symbol.

- If the Keyword processor is disabled, no leaf node is created for the Keyword symbol.

The initial tree is then built as described below - rescaling is not indicated.

6.7.3 Build Tree

To build the tree, the Huffman processor operates as follows:

Table 25: Build Tree, Huffman processor operation

Input	1) The array of Huffman tree nodes. 2) A Boolean value indicating whether frequencies need to be rescaled as a result of the root node weight becoming the maximum value.
Step 1	Assemble all leaf nodes, preserving their ascending weight order at the start of the node array. This is achieved by setting the “current node” and “assembled leaf” node position to the base of the array. If the current node is a leaf node, set the symbol and frequency associated with assembled leaf node to those of the current node and increment the assembled leaf node position. Increment the current node position and repeat this process until the current node becomes the root node. If rescaling is requested recalculate each leaf node weight as (current weight+1)/2. Set the current node to the start of the array.
Step 2	Create a parent node for the current node and the next node and insert it into the array at position x where the node at position (x+1) is the first node with a weight greater than that of the newly created node. If the newly created node is not the root node, increment the current node by 2 and goto Step 2.
Output	A completed Huffman tree.

6.7.4 Update Tree

To update the tree, the Huffman processor operates as follows:

Table 26: Update Tree, Huffman processor operation

Input	The symbol whose frequency is to be incremented by 1.
Step 1	If the weight of the root node +1 is greater than 0 x 8000 build the tree indicating that resealing is required.
Step 2	Increment the weight of the leaf node associated with the input symbol by 1 and "swap" it with the node at position y such that the node at position (y+1) is the first node encountered in the order list that has a weight greater than or equal to the new weight of the incremented leaf node. Repeat this process of weight increment and "swap" for the parent of the node at position y until the node at position y becomes the root node.
Output	An updated Huffman tree.

6.7.5 Add New Node

To add a new node, the Huffman processor operates as follows:

Table 27: Add New Node, Huffman processor operation

Input	The symbol to be added to the tree.
Step 1	Splitting the "lightest" node (the first node in the list ordered by ascending weight) such that it becomes a parent node whose right hand child is the leaf node that was originally represented by the node being split and the left hand child is a new leaf node representing the new input symbol. The new leaf node is initially created with a weight of 0.
Step 2	Update the tree (as above) passing the new symbol as the input parameter.
Output	An updated Huffman tree.

6.7.6 Compression

For compression, the Huffman processor operates as follows:

Table 28: Compression, Huffman processor operation

Input	A character from the input stream or control symbol.
Step 1	If there is no existing leaf node for the input symbol set the "source" symbol to be either the New 7bit or New 8bit symbol depending on the value of the input symbol. Otherwise set the source symbol to be the input symbol.
Step 2	Traverse the tree from the leaf node associated with the source symbol to the root node while generating the Huffman bit sequence.
Step 3	Reverse the generated Huffman bit sequence and add it to the output bitstream.
Step 4	If the source symbol equals the input symbol goto Step 5. Add the lower 7 bits of the input symbol to the output bitstream. Add a new node for the input symbol. Update the tree for the input symbol. Goto Output.
Step 5	If the input symbol value is less than 256 and bit 0 of the Huffman Initialization Options value is set, update the tree for the input symbol and goto Output.
Step 6	If the input symbol value is greater than or equal 256 and bit 1 of the Huffman Initialization Options value is set, update the tree for the input symbol.
Output	A Huffman bitstream.

6.7.7 Decompression

For decompression, the Huffman processor operates as follows:

Table 29: Decompression, Huffman processor operation

Input	A bit stream.
Step 1	Traverse the tree from the root node to a leaf node as indicated by the value of the bits read from the front of the input bitstream.
Step 2	If the symbol associated with the leaf node identified in step 1 is neither the New 7bit nor New 8bit symbol, goto Step 3. Set the lower 7 bits of the output symbol to be next 7 bits read from the input bitstream and set bit 7 as indicated. Add a new node for the output symbol. Update the tree for the output symbol. Goto Output.
Step 3	Set the output symbol to the symbol associated with the leaf node from Step 1.
Step 4	If the output symbol value is less than 256 and bit 0 of the Huffman Initialization Options value is set, update the tree for the output symbol and goto Output.
Step 5	If the input symbol value is greater than or equal 256 and bit 1 of the Huffman Initialization Options value is set update the tree for the output symbol.
Output	A decoded symbol.

7 Test Vectors

In order to assist implementors of the compression algorithm described in this specification, a suite of test vectors and 'help' information are available in electronic format. The test vectors are supplied on a single diskette attached to this specification.

These test vectors provide checks for most of the commonly expected parameter value variants in this specification and may be updated as the need arises.

Annex A (normative): German Language parameters

A.1 Compression Language Context

CLC Value: 0 (decimal)

This specifies the following items as defaults:

1) Language	German
2) Character set	Character Set ID 3 (decimal) = Code Page 850
3) Punctuator ID	0 (decimal)
4) Keyword Dictionary ID	0 (decimal)
5) Character Group ID	1 (decimal)
6) Huffman Initialization ID	1 (decimal)

A.2 Punctuators

Punctuator ID 0 (decimal)

This punctuator ID has the special meaning that no punctuator is defined (or therefore enabled) and the value of bit 2 of octet 1 of the CH is always to be interpreted as zero.

Punctuator ID 1 (decimal)

The punctuator is rendered in Character Set ID 3 (decimal) = Code Page 850.

The following characters have punctuator attributes set:

Table A.1: punctuator attributes set:

Char	Value	PU-IWS	PU-LST	PU-WSF	PU-UCF	PU-UCW	PU-NSI
<LF>	010	0	0	0	1	0	0
<CR>	013	0	0	0	1	0	0
<SP>	032	1	0	0	0	0	0
!	033	0	0	1	1	0	0
,	044	0	0	1	0	0	0
.	046	0	1	1	1	0	0
0	048	0	0	0	0	0	1
1	049	0	0	0	0	0	1
2	050	0	0	0	0	0	1
3	051	0	0	0	0	0	1
4	052	0	0	0	0	0	1
5	053	0	0	0	0	0	1
6	054	0	0	0	0	0	1
7	055	0	0	0	0	0	1
8	056	0	0	0	0	0	1
9	057	0	0	0	0	0	1
:	058	0	0	1	0	0	0
;	059	0	0	1	0	0	0
?	063	0	0	1	1	0	0
NOTE: The characters "<SP>" are used to represent the "space" character, the characters "<LF>" the "line feed" character and "<CR>" the "carriage return" character.							

Punctuator ID >1 (decimal)

No other punctuators are defined and all other values are reserved.

A.3 Keyword Dictionaries

Keyword Dictionary ID 0 (decimal)

This Keyword Dictionary ID has the special meaning that no Keyword Dictionary is defined (or therefore enabled) and the value of bit 1 of octet 1 of the CH is always to be interpreted as zero.

Keyword Dictionary ID 1 (decimal)

The Keyword Dictionary is rendered in Character Set ID 3 (decimal) = Code Page 850.

The Match Options value is 94 (decimal) indicating the following:

- Partial matching is enabled.
- Suffix matching is not enabled.
- Prefix matching is enabled.
- 1st char upper case, remainder lower case matching is enabled.

- Upper case matching is enabled.
- Lower case matching is enabled.
- Exact matching is not enabled.

The Keyword Prefix Length is 1 and the prefix string contains a single character with value 32 decimal (a space).

The Keyword Suffix Length is 0.

The Keyword Threshold value is 4.

The Maximum Partial Match Length value is 20 (decimal).

The Key Word Group List contains only 1 Key Word Group ID. The value of this Key Word Group ID is 0.

Keyword Dictionary ID >1 (decimal)

No other Keyword Dictionaries are defined and all other values are reserved.

Key Word Group ID 0 (decimal)

The entries within this Key Word Group are rendered in Character Set ID 3 (decimal) = Code Page 850.

The Number of Entries value is 128 (decimal).

The entries are defined in table A.2 wherein the characters "<SP>" are used to represent the "space" character of decimal value 32.

Table A.2: Key Word Group ID 0 (decimal)

Entry ID	Entry Length	Entry String
1	5	Abend
2	7	Abholen
3	6	Alles<SP>
4	10	Angekommen
5	9	Angerufen
6	7	Anrufen
7	7	Antwort
8	6	Anzahl
9	6	Arbeit
10	5	Auch<SP>
11	8	Bekommen
12	8	Bescheid
13	6	Besser
14	5	Bitte
15	7	Brauche
16	5	Dabei
17	6	Damit<SP>
18	5	Danke
19	5	Dann<SP>
20	8	Dienstag
21	5	Doch<SP>
22	10	Donnerstag
23	8	Dringend
24	10	Eigentlich
25	7	Einfach
26	6	Einmal
27	7	Empfang
28	7	Endlich
29	11	Erfolgreich
30	9	Eröffnung
31	8	Erhalten
32	10	Erreichbar
33	5	Essen
34	6	Etwas<SP>
35	6	Fahren
36	10	Feierabend
37	6	Fertig
38	7	Freitag
39	6	Freund
40	5	Gegen
41	5	Gehen
42	5	Geht_
43	6	Gerade
44	8	Gespräch
45	7	Gestern
46	6	Glaube
47	6	Gleich
48	6	Grüsse
49	5	Guten
50	5	Haben
51	6	Hallo<SP>
52	6	Heute<SP>
53	12	Hoffentlich<SP>
54	6	Immer<SP>
55	6	Jetzt<SP>
56	6	Kaufen
57	6	Können
58	5	Komme
59	6	Konnte
60	5	Konto
61	5	Lange

Entry ID	Entry Length	Entry String
62	7	Langsam
63	6	Lassen
64	6	Laufen
65	7	Leider<SP>
66	6	Letzte
67	5	Liebe
68	6	Machen
69	5	Macht
70	6	Melden
71	6	Mittag
72	8	Mittwoch
73	6	Montag
74	6	Morgen
75	7	Nachher
76	10	Nachmittag
77	9	Nachricht
78	5	Nacht
79	9	Natürlich
80	5	Nicht
81	6	Nummer
82	7	Nutzung
83	5	Pause
84	7	Problem
85	7	Rückruf
86	8	Rechnung
87	5	Reden
88	7	Richtig
89	5	Sagen
90	7	Samstag
91	8	Schlafen
92	8	Schlecht
93	7	Schnell
94	6	Schon<SP>
95	5	Schön
96	7	Schreib
97	6	Schule
98	5	Sehen
99	6	Sicher
100	6	Sofort
101	7	Sonntag
102	5	Sonst
103	6	Später
104	6	Stunde
105	7	Telefon
106	6	Termin
107	5	Total
108	7	Treffen
109	7	Trinken
110	10	Unterwegs<SP>
111	6	urlaub
112	9	Vergessen
113	7	Versuch
114	11	Vielleicht<SP>
115	14	Wahrscheinlich
116	5	Wann<SP>
117	6	Warum<SP>
118	6	Wegen<SP>
119	5	Wenn<SP>
120	6	Werden
121	7	Wichtig
122	6	Wieder
123	8	Wirklich

Entry ID	Entry Length	Entry String
124	6	Wissen
125	5	Woche
126	10	Wochenende
127	6	Zurück
128	8	Zusammen

Key Word Group ID >0 (decimal)

No other Key Word Groups are defined and all other values are reserved.

A.4 Character Groups

Character Group ID 0 (decimal)

This Character Group ID has the special meaning that no Character Group is defined (or therefore enabled) and the value of bit 0 of octet 1 of the CH is always to be interpreted as zero.

Character Group ID 1 (decimal)

The Character Group is rendered in Character Set ID 3 (decimal) = Code Page 850.

The Number of Groups value is 3.

There are 2 Group Transition symbols used these have the decimal values 259 and 260. Their use in signalling transitions between the 3 groups are shown in the table A.3.

Table A.3: Character Group ID 1 (decimal)

	New Group		
Current Group	0	1	2
0		260	259
1	260		259
2	260	259	

The fold tables and Group Membership bit flags are set out in the following table A.4.

Table A.4: fold tables and Group Membership bit flags

Char	Value	Group 0 Fold Table	Group 1 Fold Table	Group 2 Fold Table	Group 2 Member	Group 1 Member	Group 0 Member
<SP>	032	032	032	032	1	1	1
!	033	033	033	033	1	1	1
"	034	034	034	012	1	1	1
#	035	107	035	035	1	0	0
$	036	113	035	035	1	0	0
%	037	118	037	037	1	0	0
&	038	112	038	038	1	0	0
'	039	111	039	039	1	0	0
(	040	119	040	040	1	0	0
)	041	098	041	041	1	0	0
*	042	109	042	042	1	0	0
+	043	103	043	043	1	0	0
,	044	044	044	044	1	1	1
-	045	097	045	045	1	0	0
.	046	046	046	046	1	1	1
/	047	102	047	047	1	0	0
0	048	110	048	048	1	0	0
1	049	101	049	049	1	0	0
2	050	105	050	050	1	0	0
3	051	104	051	051	1	0	0
4	052	114	052	052	1	0	0
5	053	100	053	053	1	0	0
6	054	108	054	054	1	0	0
7	055	115	055	055	1	0	0
8	056	117	056	056	1	0	0
9	057	099	057	057	1	0	0
:	058	116	058	058	1	0	0
;	059	121	059	059	1	0	0
<	060	106	060	060	1	0	0
=	061	120	061	061	1	0	0
>	062	122	062	062	1	0	0
?	063	063	063	063	1	1	1
A	065	097	065	065	0	1	0
B	066	098	066	066	0	1	0
C	067	099	067	067	0	1	0
D	068	100	068	068	0	1	0
E	069	101	069	069	0	1	0
F	070	102	070	070	0	1	0
G	071	103	071	071	0	1	0
H	072	104	072	072	0	1	0
I	073	105	073	073	0	1	0
J	074	106	074	074	0	1	0
K	075	107	075	075	0	1	0
L	076	108	076	076	0	1	0
M	077	109	077	077	0	1	0
N	078	110	078	078	0	1	0
O	079	111	079	079	0	1	0
P	080	112	080	080	0	1	0
Q	081	113	081	081	0	1	0
R	082	114	082	082	0	1	0
S	083	115	083	083	0	1	0
T	084	116	084	084	0	1	0
U	085	117	085	085	0	1	0
V	086	118	086	086	0	1	0
W	087	119	087	087	0	1	0
X	088	120	088	088	0	1	0
Y	089	121	089	089	0	1	0
Z	090	122	090	090	0	1	0
a	097	097	065	045	0	0	1

Char	Value	Group 0 Fold Table	Group 1 Fold Table	Group 2 Fold Table	Group 2 Member	Group 1 Member	Group 0 Member
b	098	098	066	041	0	0	1
c	099	099	067	057	0	0	1
d	100	100	068	053	0	0	1
e	101	101	069	049	0	0	1
f	102	102	070	047	0	0	1
g	103	103	071	043	0	0	1
h	104	104	072	051	0	0	1
i	105	105	073	050	0	0	1
j	106	106	074	060	0	0	1
k	107	107	075	035	0	0	1
l	108	108	076	054	0	0	1
m	109	109	077	042	0	0	1
n	110	110	078	048	0	0	1
o	111	111	079	039	0	0	1
p	112	112	080	038	0	0	1
q	113	113	081	036	0	0	1
r	114	114	082	052	0	0	1
s	115	115	083	055	0	0	1
t	116	116	084	058	0	0	1
u	117	117	085	056	0	0	1
v	118	118	086	037	0	0	1
w	119	119	087	040	0	0	1
x	120	120	088	061	0	0	1
y	121	121	089	059	0	0	1
z	122	122	090	062	0	0	1

NOTE: The characters "<SP>" are used to represent the "space" character.

Characters with any other value in the range 0 to 255 are not a member of any group and therefore the fold table values will be equal to the character value in all cases.

Character Group ID >1 (decimal)

No other Character Groups are defined and all other values are reserved.

A.5 Huffman Initializations

Huffman Initialization ID 0 (decimal)

The Huffman Initialization is rendered in Character Set ID 3(decimal) = Code Page 850.

The Options value indicates that both character and control symbol updating are enabled.

As described is subclause 6.7.1, the tables below include initialization values for *all* control symbols that *might* occur in conjunction with the use of this Huffman Initialization. However, initialization values for control symbols that *cannot* occur for a *particular* use of this Huffman Initialization are identified as part of the Huffman initialization process and are *not* added to the Huffman tree as described in subclause 6.7.2.

The Character Group ID value is 1.

Character Group Processing is disabled:

The number of frequency initializations is 4.

The initial frequencies are:

Table A.5: Character Group Processing is disabled: initial frequencies

Symbol	Value	Frequency
New UCS2 Row	266	1
Keyword	258	1
New 8bit	257	1
New 7bit	256	1

Character Group Processing is enabled:

The number of frequency initializations is 6.

The initial frequencies are:

Table A.6: Character Group Processing is enabled: initial frequencies

Symbol	Value	Frequency
New UCS2 Row	266	1
Change CG1	260	1
Change CG0	259	1
Keyword	258	1
New 8bit	257	1
New 7bit	256	1

Huffman Initialization ID 1 (decimal)

The Huffman Initialization is rendered in Character Set ID 3 (decimal) = Code Page 850.

The Options value indicates that both character and control symbol updating are enabled.

As described is subclause 6.7.1, the tables below include initialization values for *all* control symbols that *might* occur in conjunction with the use of this Huffman Initialization. However, initialization values for control symbols that *cannot* occur for a *particular* use of this Huffman Initialization are identified as part of the Huffman initialization process and are *not* added to the Huffman tree as described in subclause 6.7.2.

The Character Group ID value is 1.

Character Group Processing is disabled:

The number of frequency initializations is 32.

The initial frequencies are:

Table A.7: Character Group Processing is disabled: initial frequencies

Symbol	Value	Frequency
New UCS2 Row	266	00001
q	113	00001
x	120	00001
y	121	00001
j	106	00001
v	118	00001
p	112	00001
New 8bit	257	00002
z	122	00002
.	46	00003
k	107	00003
f	102	00003
w	119	00003
Keyword	258	00004
b	98	00004
g	103	00004
o	111	00005
m	109	00006
l	108	00006
u	117	00007
c	99	00007
d	100	00007
New 7bit	256	00009
r	114	00009
t	116	00009
s	115	00010
h	104	00010
a	97	00012
i	105	00013
n	110	00014
e	101	00021
<SP>	32	00032

NOTE: In the above table, the characters "<SP>" are used to represent the "space" character.

Character Group Processing is enabled:

The number of frequency initializations is 34.

The initial frequencies are:

Table A.8: Character Group Processing is enabled: initial frequencies

Symbol	Value	Frequency
New UCS2 Row	266	00001
q	113	00001
x	120	00001
y	121	00001
j	106	00001
v	118	00001
p	112	00001
New 8bit	257	00002
z	122	00002
Change CG0	259	00002
.	46	00003
k	107	00003
f	102	00003
w	119	00003
Keyword	258	00004
Change CG1	260	00004
b	98	00004
g	103	00004
o	111	00005
m	109	00006
l	108	00006
u	117	00007
c	99	00007
d	100	00007
New 7bit	256	00009
r	114	00009
t	116	00009
s	115	00010
h	104	00010
a	97	00012
i	105	00013
n	110	00014
e	101	00021
<SP>	32	00032

Note in the above table, the characters "<SP>" are used to represent the "space" character.

Huffman Initialization ID >1 (decimal)

No other Huffman Initializations are defined and all other values are reserved.

Annex B (normative): English language parameters

B.1 Compression Language Context

CLC Value: 1 (decimal)

This specifies the following items as defaults:

1) Language	English
2) Character set	Character Set ID 2 (decimal) = Code page 437
3) Punctuator ID	1 (decimal)
4) Keyword Dictionary ID	0 (decimal)
5) Character Group ID	1 (decimal)
6) Huffman Initialization ID	1 (decimal)

B.2 Punctuators

Punctuator ID 0 (decimal)

This punctuator ID has the special meaning that no punctuator is defined (or therefore enabled) and the value of bit 2 of octet 1 of the CH is always to be interpreted as zero.

Punctuator ID 1 (decimal)

The punctuator is rendered in Character Set ID 2 (decimal) = Code Page 437.

The following characters have punctuator attributes set:

Table B.1: punctuator attributes set:

Char	Value	PU-IWS	PU-LST	PU-WSF	PU-UCF	PU-UCW	PU-NSI
<LF>	010	0	0	0	1	0	0
<CR>	013	0	0	0	1	0	0
<SP>	032	1	0	0	0	0	0
!	033	0	0	1	1	0	0
,	044	0	0	1	0	0	0
.	046	0	1	1	1	0	0
0	048	0	0	0	0	0	1
1	049	0	0	0	0	0	1
2	050	0	0	0	0	0	1
3	051	0	0	0	0	0	1
4	052	0	0	0	0	0	1
5	053	0	0	0	0	0	1
6	054	0	0	0	0	0	1
7	055	0	0	0	0	0	1
8	056	0	0	0	0	0	1
9	057	0	0	0	0	0	1
:	058	0	0	1	0	0	0
;	059	0	0	1	0	0	0
?	063	0	0	1	1	0	0
I	073	0	0	0	0	1	0
NOTE: The characters "<SP>" are used to represent the "space" character, the characters "<LF>" the "line feed" character and "<CR>" the "carriage return" character.							

Punctuator ID >1 (decimal)

No other punctuators are defined and all other values are reserved.

B.3 Keyword Dictionaries

Keyword Dictionary ID 0 (decimal)

This Keyword Dictionary ID has the special meaning that no Keyword Dictionary is defined (or therefore enabled) and the value of bit 1 of octet 1 of the CH is always to be interpreted as zero.

Keyword Dictionary ID 1 (decimal)

The Keyword Dictionary is rendered in Character Set ID 2 (decimal) = Code Page 437.

The Match Options value is 94 (decimal) indicating the following:

- Partial matching is enabled.
- Suffix matching is not enabled.
- Prefix matching is enabled.

- 1st char upper case, remainder lower case matching is enabled.
- Upper case matching is enabled.
- Lower case matching is enabled.
- Exact matching is not enabled.

The Keyword Prefix Length is 1 and the prefix string contains a single character with value 32 decimal (a space).

The Keyword Suffix Length is 0.

The Keyword Threshold value is 4.

The Maximum Partial Match Length value is 46 (decimal).

The Key Word Group List contains only 1 Key Word Group ID. The value of this Key Word Group ID is 0.

Keyword Dictionary ID >1 (decimal)

No other Keyword Dictionaries are defined and all other values are reserved.

Key Word Group ID 0 (decimal)

The entries within this Key Word Group are rendered in Character Set ID 2 (decimal) = Code Page 437.

The Number of Entries value is 128 (decimal).

The entries are defined in table B.2wherein the characters "<SP>" are used to represent the "space" character of decimal value 32.

Table B.2: Key Word Group ID 0 (decimal)

Entry ID	Entry Length	Entry String
1	5	About
2	9	Afternoon
3	5	Again
4	6	Agenda
5	6	Agreed
6	4	And<SP>
7	11	Appointment
8	4	Are<SP>
9	7	Arrange
10	6	Arrive
11	6	Attend
12	9	Available
13	4	Away
14	7	Because
15	6	Before
16	7	Benefit
17	8	Business
18	4	But<SP>
19	4	Call
20	6	Can't<SP>
21	6	Cancel
22	6	Commit
23	7	Company
24	8	Complete
25	7	Confirm
26	7	Contact
27	10	Convenient
28	5	Could
29	7	Deliver
30	6	Demand
31	10	Department
32	6	Dinner
33	7	Discuss
34	6	Don't<SP>
35	5	Exist
36	6	Flight
37	4	For<SP>
38	7	Forward
39	6	Friday
40	5	From<SP>
41	5	Going
42	7	Goodbye
43	8	Hardware
44	5	Have<SP>
45	4	Hear
46	5	Hello
47	4	Help
48	4	Home
49	5	Hotel
50	4	How<SP>
51	9	Immediate
52	9	Important
53	11	Information
54	4	Its<SP>
55	5	Later
56	6	Letter
57	7	Machine
58	5	Make<SP>
59	6	Manage
60	7	Meeting
61	7	Message

Entry ID	Entry Length	Entry String
62	6	Mobile
63	6	Monday
64	7	Morning
65	5	Need<SP>
66	6	Office
67	5	Other
68	6	Passed
69	8	Personal
70	5	Phone
71	6	Please
72	8	Possible
73	4	Post
74	8	Postpone
75	5	Price
76	8	Priority
77	7	Product
78	7	Project
79	5	Quick
80	7	Receive
81	9	Reference
82	7	Regards
83	8	Remember
84	6	Return
85	4	Ring
86	8	Saturday
87	4	Send
88	7	Service
89	6	Should
90	5	Since
91	8	Software
92	4	Soon
93	5	Speak
94	5	Still
95	7	Subject
96	7	Success
97	6	Sunday
98	4	Talk
99	9	Telephone
100	5	Thank
101	4	That
102	4	The<SP>
103	5	Them<SP>
104	5	There
105	5	They<SP>
106	5	Think
107	4	This
108	8	Thursday
109	5	Today
110	8	Tomorrow
111	7	Tonight
112	5	Total
113	6	Travel
114	7	Tuesday
115	6	Until<SP>
116	6	Update
117	6	Urgent
118	5	Using
119	4	Want
120	9	Wednesday
121	7	Weekend
122	7	Welcome
123	5	When<SP>

Entry ID	Entry Length	Entry String
124	6	Where<SP>
125	4	Will
126	5	Would
127	9	Yesterday
128	4	You<SP>

Key Word Group ID >0 (decimal)

No other Key Word Groups are defined and all other values are reserved.

B.4 Character Groups

Character Group ID 0 (decimal)

This Character Group ID has the special meaning that no Character Group is defined (or therefore enabled) and the value of bit 0 of octet 1 of the CH is always to be interpreted as zero.

Character Group ID 1 (decimal)

The Character Group is rendered in Character Set ID 2 (decimal) = Code Page 437.

The Number of Groups value is 3.

There are 2 Group Transition symbols used these have the decimal values 259 and 260. Their use in signalling transitions between the 3 groups are shown in the table B.3.

Table B.3: Character Group ID 1 (decimal)

	New Group		
Current Group	0	1	2
0		260	259
1	260		259
2	260	259	

The fold tables and Group Membership bit flags are set out in the following table B.4.

Table B.4: fold tables and Group Membership bit flags

Char	Value	Group 0 Fold Table	Group 1 Fold Table	Group 2 Fold Table	Group 2 Member	Group 1 Member	Group 0 Member
	012	034	012	012	1	0	0
<SP>	032	032	032	032	1	1	1
!	033	118	033	033	1	0	0
"	034	034	034	012	0	1	1
#	035	102	035	035	1	0	0
%	037	113	037	037	1	0	0
&	038	111	038	038	1	0	0
'	039	039	039	039	1	1	1
(	040	116	040	040	1	0	0
)	041	117	041	041	1	0	0
*	042	110	042	042	1	0	0
+	043	119	043	043	1	0	0
,	044	044	044	062	0	1	1
-	045	120	045	045	1	0	0
.	046	046	046	046	1	1	1
/	047	114	047	047	1	0	0
0	048	101	048	048	1	0	0
1	049	097	049	049	1	0	0
2	050	105	050	050	1	0	0
3	051	099	051	051	1	0	0
4	052	112	052	052	1	0	0
5	053	100	053	053	1	0	0
6	054	107	054	054	1	0	0
7	055	104	055	055	1	0	0
8	056	103	056	056	1	0	0
9	057	109	057	057	1	0	0
:	058	098	058	058	1	0	0
;	059	106	059	059	1	0	0
<	060	122	060	060	1	0	0
=	061	121	061	061	1	0	0
>	062	044	062	062	1	0	0
?	063	063	063	093	0	1	1
A	065	097	065	065	0	1	0
B	066	098	066	066	0	1	0
C	067	099	067	067	0	1	0
D	068	100	068	068	0	1	0
E	069	101	069	069	0	1	0
F	070	102	070	070	0	1	0
G	071	103	071	071	0	1	0
H	072	104	072	072	0	1	0
I	073	105	073	073	0	1	0
J	074	106	074	074	0	1	0
K	075	107	075	075	0	1	0
L	076	108	076	076	0	1	0
M	077	109	077	077	0	1	0
N	078	110	078	078	0	1	0
O	079	111	079	079	0	1	0
P	080	112	080	080	0	1	0
Q	081	113	081	081	0	1	0
R	082	114	082	082	0	1	0
S	083	115	083	083	0	1	0
T	084	116	084	084	0	1	0
U	085	117	085	085	0	1	0
V	086	118	086	086	0	1	0
W	087	119	087	087	0	1	0
X	088	120	088	088	0	1	0
Y	089	121	089	089	0	1	0
Z	090	122	090	090	0	1	0
[	091	108	091	091	1	0	0

Char	Value	Group 0 Fold Table	Group 1 Fold Table	Group 2 Fold Table	Group 2 Member	Group 1 Member	Group 0 Member
]	093	063	093	093	1	0	0
a	097	097	065	049	0	0	1
b	098	098	066	058	0	0	1
c	099	099	067	051	0	0	1
d	100	100	068	053	0	0	1
e	101	101	069	048	0	0	1
f	102	102	070	035	0	0	1
g	103	103	071	056	0	0	1
h	104	104	072	055	0	0	1
i	105	105	073	050	0	0	1
j	106	106	074	059	0	0	1
k	107	107	075	054	0	0	1
l	108	108	076	091	0	0	1
m	109	109	077	057	0	0	1
n	110	110	078	042	0	0	1
o	111	111	079	038	0	0	1
p	112	112	080	052	0	0	1
q	113	113	081	037	0	0	1
r	114	114	082	047	0	0	1
s	115	115	083	156	0	0	1
t	116	116	084	040	0	0	1
u	117	117	085	041	0	0	1
v	118	118	086	033	0	0	1
w	119	119	087	043	0	0	1
x	120	120	088	045	0	0	1
y	121	121	089	061	0	0	1
z	122	122	090	060	0	0	1
£	156	115	156	156	1	0	0

NOTE: The characters "<SP>" are used to represent the "space" character.

Characters with any other value in the range 0 to 255 are not a member of any group and therefore the fold table values will be equal to the character value in all cases.

Character Group ID >1 (decimal)

No other Character Groups are defined and all other values are reserved.

B.5 Huffman Initializations

Huffman Initialization ID 0 (decimal)

The Huffman Initialization is rendered in Character Set ID 2 (decimal) = Code Page 437.

The Options value indicates that both character and control symbol updating are enabled.

As described is subclause 6.7.1, the tables below include initialization values for *all* control symbols that *might* occur in conjunction with the use of this Huffman Initialization. However, initialization values for control symbols that *cannot* occur for a *particular* use of this Huffman Initialization are identified as part of the Huffman initialization process and are *not* added to the Huffman tree as described in subclause 6.7.2.

The Character Group ID value is 1.

Character Group Processing is disabled:

The number of frequency initializations is 4.

The initial frequencies are:

Table B.5: Character Group Processing is disabled: initial frequencies

Symbol	Value	Frequency
New UCS2 Row	266	1
Keyword	258	1
New 8bit	257	1
New 7bit	256	1

Character Group Processing is enabled:

The number of frequency initializations is 6.

The initial frequencies are:

Table B.6: Character Group Processing is enabled: initial frequencies

Symbol	Value	Frequency
New UCS2 Row	266	1
Change CG1	260	1
Change CG0	259	1
Keyword	258	1
New 8bit	257	1
New 7bit	256	1

Huffman Initialization ID 1 (decimal)

The Huffman Initialization is rendered in Character Set ID 2 (decimal) = Code Page 437.

The Options value indicates that both character and control symbol updating are enabled.

As described is subclause 6.7.1, the tables below include initialization values for *all* control symbols that *might* occur in conjunction with the use of this Huffman Initialization. However, initialization values for control symbols that *cannot* occur for a *particular* use of this Huffman Initialization are identified as part of the Huffman initialization process and are *not* added to the Huffman tree as described in subclause 6.7.2.

The Character Group ID value is 1.

Character Group Processing is disabled:

The number of frequency initializations is 32.

The initial frequencies are:

Table B.7: Character Group Processing is disabled: initial frequencies

Symbol	Value	Frequency
New UCS2 Row	266	00001
z	122	00001
Keyword	258	00001
q	113	00001
j	106	00003
x	120	00003
New 7bit	256	00003
New 8bit	257	00003
v	118	00008
w	119	00010
b	098	00010
y	121	00011
f	102	00011
u	117	00012
.	046	00014
m	109	00016
g	103	00017
k	107	00017
h	104	00018
d	100	00024
p	112	00029
c	099	00029
i	105	00030
r	114	00038
l	108	00038
s	115	00040
n	110	00048
t	116	00050
o	111	00055
<SP>	032	00060
a	097	00066
e	101	00079

NOTE: In the above table, the characters "<SP>" are used to represent the "space" character.

Character Group Processing is enabled:

The number of frequency initializations is 34.

The initial frequencies are:

Table B.8: Character Group Processing is enabled: initial frequencies

Symbol	Value	Frequency
New UCS2 Row	266	00001
Change CG1	260	00001
z	122	00001
Keyword	258	00001
q	113	00002
j	106	00003
x	120	00003
New 7bit	256	00003
New 8bit	257	00003
v	118	00008
w	119	00010
b	098	00010
Change CG0	259	00010
y	121	00011
f	102	00013
u	117	00013
.	046	00015
m	109	00017
g	103	00017
k	107	00019
h	104	00020
d	100	00026
p	112	00030
c	099	00030
i	105	00031
r	114	00040
l	108	00040
s	115	00045
n	110	00050
t	116	00053
o	111	00054
<SP>	032	00058
a	097	00064
e	101	00077

Note in the above table, the characters "<SP>" are used to represent the "space" character.

Huffman Initialization ID >1 (decimal)

No other Huffman Initializations are defined and all other values are reserved.

Annex C (normative): Italian Language parameters

Annex under development

Annex D (normative): French Language parameters

Annex under development

Annex E (normative): Spanish Language parameters

Annex under development

Annex F (normative): Dutch Language parameters

Annex under development

Annex G (normative): Swedish Language parameters

Annex under development

Annex H (normative): Danish Language parameters

Annex under development

Annex J (normative): Portuguese Language parameters

Annex under development

Annex K (normative): Finnish Language parameters

Annex under development

Annex L (normative): Norwegian Language parameters

Annex under development

Annex M (normative): Greek Language parameters

Annex under development

Annex N (normative): Turkish Language parameters

Annex under development

Annex P (normative): Reserved

Annex under development

Annex Q (normative): Reserved

Annex under development

Annex R (normative): Default Parameters for Unspecified Language

R.1 Compression Language Context

CLC Value: 15 (decimal)

This specifies the following items as defaults:

1) Language	Unspecified
2) Character set bitdefault alphabet	Character Set ID 1 (decimal) = 3GPP TS 23.038 [1] GSM 7
3) Punctuator ID	0 (decimal)
4) Keyword Dictionary ID	0 (decimal)
5) Character Group ID	0 (decimal)
6) Huffman Initialization ID	0 (decimal)

R.2 Punctuators

Punctuator ID 0 (decimal)

This punctuator ID has the special meaning that no punctuator is defined (or therefore enabled) and the value of bit 2 of octet 1 of the CH is always to be interpreted as zero.

Punctuator ID >0 (decimal)

No other punctuators are defined and all other values are reserved.

R.3 Keyword Dictionaries

Keyword Dictionary ID 0 (decimal)

This Keyword Dictionary ID has the special meaning that no Keyword Dictionary is defined (or therefore enabled) and the value of bit 1 of octet 1 of the CH is always to be interpreted as zero.

Keyword Dictionary ID >0 (decimal)

No other Keyword Dictionaries are defined and all other values are reserved.

R.4 Character Groups

Character Group ID 0 (decimal)

This Character Group ID has the special meaning that no Character Group is defined (or therefore enabled) and the value of bit 0 of octet 1 of the CH is always to be interpreted as zero.

Character Group ID >0 (decimal)

No other Character Groups are defined and all other values are reserved.

R.5 Huffman Initializations

Huffman Initialization ID 0 (decimal)

Only control symbols are included in this initialization. It's rendition is therefore independent of character set.

The Options value indicates that both character and control symbol updating are enabled.

As described is subclause 6.7.1, the tables below include initialization values for *all* control symbols that *might* occur in conjunction with the use of this Huffman Initialization. However, initialization values for control symbols that *cannot* occur for a *particular* use of this Huffman Initialization are identified as part of the Huffman initialization process and are *not* added to the Huffman tree as described in subclause 6.7.2.

Character Group Processing is always disabled:

The number of frequency initializations is 4.

The initial frequencies are:

Table R.1: Character Group Processing is always disabled: initial frequencies

Symbol	Value	Frequency
New UCS2 Row	266	1
Keyword	258	1
New 8bit	257	1
New 7bit	256	1

Character Group Processing can not be enabled therefore:

The number of frequency initializations is 0.

Huffman Initialization ID >0 (decimal)

No other Huffman Initializations are defined and all other values are reserved.

Annex S (informative): Change history

TSG#	TDoc	VERS	NEW_ VERS	CR	REV	REL	CAT	WORKITEM	SUBJECT
T#4			3.0.0	New					Creation of 3GPP 23.042 out of GSM 03.42 v.7.1.0
T#6	TP-99237	3.0.0	3.1.0	001		R99	F	TEI	Adaptations for UMTS
T#11	-	3.1.0	4.0.0	-		Rel-4			Upgrade to Rel-4
		4.0.0	4.0.1			Rel-4			2001-10: Replacement of erroneous R99 test vector file by correct Rel-4 one.
T#16	-	4.0.1	5.0.0	-		Rel-5			Upgrade to Rel-5
T#26	-	5.0.0	6.0.0	-		Rel-6			Upgrade to Rel-6
		6.0.0	7.0.0			Rel-7			Upgrade to Rel-7
		7.0.0	8.0.0			Rel-8			Upgrade to Rel-8
		8.0.0	9.0.0			Rel-9			Upgrade to Rel-9
		9.0.0	10.0.0			Rel-10			Upgrade to Rel-10
		10.0.0	11.0.0			Rel-11			Upgrade to Rel-11