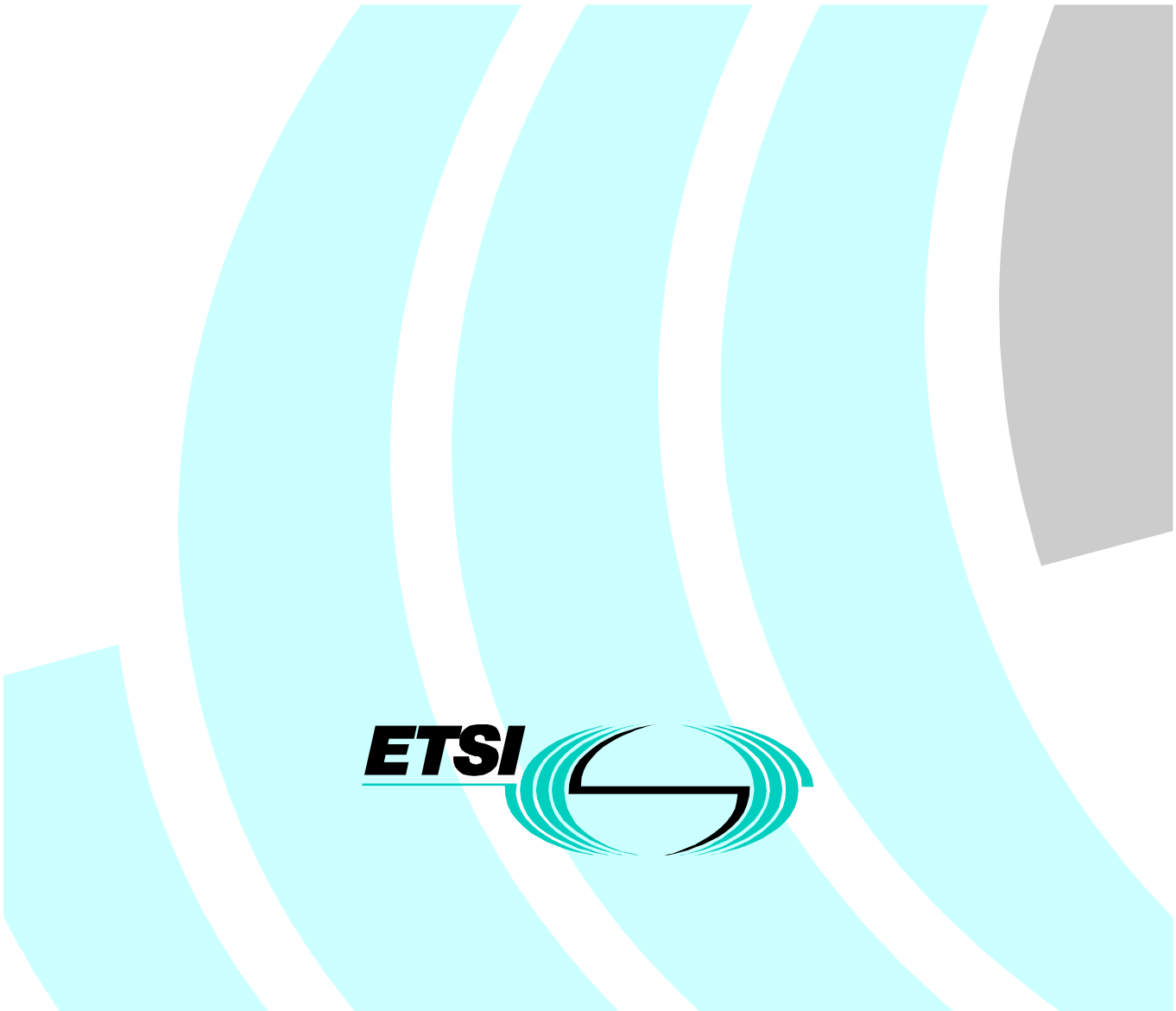
Draft ES 201 684 V1.1.1 (1999-05)

ETSI Standard

Integrated Services Digital Network (ISDN); File Transfer Profile; B-channel aggregation and synchronous compression



Reference
DES/DTA-005069 (flc00icp.PDF)

Keywords
B-channel, FT, ISDN, Terminal

ETSI

Postal address
F-06921 Sophia Antipolis Cedex - FRANCE

Office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16
Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Internet
secretariat@etsi.fr
Individual copies of this ETSI deliverable
can be downloaded from
http://www.etsi.org
If you find errors in the present document, send your
comment to: editor@etsi.fr

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available **free of charge** from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://www.etsi.org/ipr).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This ETSI Standard (ES) has been produced by ETSI Project Digital Terminals and Access (DTA), and is now submitted for the ETSI standards Membership Approval Procedure.

1 Scope

The present document specifies the negotiation mechanism and the algorithm used for file transfer compression and channel aggregation.

The present document completes the work related to the need for a standardized simple file transfer protocol over ISDN. The work is limited to ETS 300 383 [1] that defines File Transfer over ISDN, EUROFILE transfer profile (EFT), but is applicable for other file transfer protocols.

The present document establishes the following principles:

- Definition of the compression mechanism applicable for EFT;
- Definition of the channel aggregation mechanism applicable for EFT;
- Definition of the associated negotiation mechanisms.

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.
- A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] ETS 300 383: "Integrated Services Digital Network (ISDN); File transfer over the ISDN EUROFILE transfer profile".

[2] ETS 300 075 including Amendment 1: "Terminal Equipment (TE); Processable data; File transfer".

[3] ITU-T Recommendation X.25 (1996): "Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit".

[4] ITU-T Recommendation V.42 bis (1990): "Data compression procedures for data circuit terminating equipment (DCE) using error correcting procedures".

[5] IETF RFC 1951 (May 1996): "DEFLATE Compressed Data Format Specification version 1.3".

[6] IETF RFC 1952 (May 1996): "GZIP File Format Specification version 4.3".

[7] ITU-T Recommendation X.210 (1993): "Information technology; Open systems interconnection; Basic Reference Model: Conventions for the definition of OSI services".

[8] ISO/IEC 13871 (1995): "Information technology; Telecommunications and information exchange between systems; Private telecommunications networks; Digital channel aggregation".

[9] IETF RFC 1717 (November 1994): "The PPP Multilink Protocol (MP)".

[10] ISO/IEC 8208 (1995): "Information technology; Data communications; X.25 Packet Layer Protocol for Data Terminal Equipment".

3 Definitions and abbreviations

3.1 Definitions

Within the context of the present document the following definitions apply. In case the definition is drawn from another source, the reference to the source is cited accordingly.

(service-) primitive: abstract, implementation independent interaction between a (service-) user and a (service-) provider. [ITU-T Recommendation X.210 [7]]

buffer: piece of storage of whatever kind belonging to the terminal equipment, where data can be held temporarily.

called party: party which should wait for calls and operations. It has a passive role and acts as a slave.

caller: party which is the initiator of the call. It has an active role and is the master compared to the remote terminal. It inputs the requests for actions on files. [ETS 300 383 [1]]

compressed data set: whole compressed data produced/consumed by the DEFLATE compatible compression algorithm during one run.

linked channel: connection that is established subsequent to an existing main channel within the context of channel aggregation.

main channel: connection that is established first within the context of channel aggregation.

mass transfer phase: period of time during which transfer of data may occur.

mass transfer primitive: primitives that perform the data transfer during the mass transfer phase.

master: within the context of channel aggregation the master is presented by the caller that initiates establishment of the main channel.

physical file: physical unit of an arbitrary amount of continuous data that can be accessed within a terminal equipment. A physical file can be stored and retrieved sequentially within the terminal equipment through the use of specialized retrieval or storage functions provided by the terminal. Each physical file stored in the terminal equipment has a unique identifier to be used to access the file.

regime: set of protocol phases; a regime is a continuous period of time. A regime is established by using a confirmed or optionally confirmed service and it is orderly terminated by using a confirmed service, it may also be interrupted in an abnormal manner. A regime is fully defined by specifying the service(s) used to establish it and the service(s) used to terminate it. A regime is used in this description to limit the range of some services which may only be available during a particular regime. [ETS 300 075 [2]]

slave: within the context of channel aggregation the slave is presented by the called party towards which the main channel has been established.

transfer unit: data transferred by using one mass transfer primitive. [ETS 300 075 [2]]

transfer file: whole amount of data to be transferred during a mass transfer phase, i.e. the raw information plus additional attributes or header, trailer or other fields.

3.2 Abbreviations

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

ASCII	American Standard Code for Information Interchange
BONDING	Bandwidth ON Demand INteroperability Group
CAR-ID	Channel Aggregation Reference IDentifier
CRC	Cyclic Redundancy Check
CSN	Channel Sequence Number
DTE	Data Terminal Equipment
EFT	EUROFILE Transfer according to ETS 300 383 [1]
IPR	Intellectual Property Rights
ISDN	Integrated Services Digital Network
MLPPP	Multi Link PPP
NSAP	Network layer Service Access Point
OSI	Open Systems Interconnection
PCI	Programmable Communication Interface
PPP	Point to Point Protocol
RFC	Request For Comments
TE	Terminal Equipment
TDU	Teleservice Data Unit
TLV	Type - Length - Value

4 Overview and reader's guideline

The main purposes of the present document are as follows:

- to define a standard compression and channel aggregation service for EFT;
- to provide end-to-end compatibility between terminals supporting such services.

The general guidelines are given in clause 5.

Clause 6 describes the services provided, gives some background information and explains the algorithms used.

The coding and implementation of the structures and algorithms imposed by the new services are described within clause 7.

5 General model

This clause provides the general model and general principles that guide the specification of the new services for file transfer. The specification aims to:

- be independent of hardware platforms;
- provide for interoperability and compatibility;
- be extensible.

The major concern of the present document is however, to provide up-to-date services making best use of proven, reliable and well accepted algorithms in order to improve the acceptance and wide spread usage of the EUROFILE service in general.

6 Services description

The present document provides new, additional services for the EUROFILE transfer over ISDN, namely:

- an additional compression method and format;
- a new channel aggregation mechanism localized on layer 3 of the protocol pillar.

6.1 Compression

This subclause specifies an additional, optional compression method and data transfer format. For reference purposes, this compression method is called the *balanced compression* method.

6.1.1 General description

ETS 300 075 [2] already defines various data transfer formats and provides for optional compression methods. However, those compression methods have been experienced to:

- be efficient in terms of speed but poor in compression ratio;
- deliver excellent compression ratio on the cost of speed and continuous data flow;
- be good in terms of compression ratio and speed but difficult to implement in a way that guarantees conformance among different system platforms and implementations.

As available bandwidth increases due to channel aggregation, the demand grows for an algorithm that is fairly balanced between compression ratio, speed and continuous data flow. Furthermore, the necessity of conformance between different platforms and implementations highly recommends an algorithm that is easily available in the public domain.

The present document does not specify a compression method by itself. It rather proposes to use a compression method that is well balanced, widely used and available in the public domain. The compression method proposed is compatible with the so called DEFLATE compressed data format and is described in IETF RFC 1951 [5]. The compressed data are transferred using a format that is known as GZIP. A specification of GZIP can be found in IETF RFC 1952 [6]. Code libraries for the implementation of the GZIP/DEFLATE compression formats are available in the public domain.

For reference purposes copies of IETF RFC 1951 [5] and IETF RFC 1952 [6] are included as informative annexes.

6.1.2 Negotiation

The default transfer format is non-compressed. The optional compressed format shall only be exchanged after successful negotiation.

The basic negotiation procedure is defined in ETS 300 383 [1]. The communicating entities exchange capability lists inside the T-ACCESS primitive when establishing the Access regime [2]. This capability list is located in the User data parameter of the T-ACCESS primitive.

To negotiate the additional compression method the present document defines an specific value for the *Compression mode* to be used by the T-ACCESS primitive during negotiation and the T-WRITE or T-WRITE-END primitive inside the file attributes during the mass transfer phase. The encoding of the *Compression mode* is described in subclause 7.1.1.

6.1.3 Compression service description

During the mass transfer phase that occurs within an established Transfer regime, the T-WRITE (or T-WRITE-END) primitive repeatedly transfers the data chunk by chunk as single Transfer units [2].

The data to be transferred - called the Transfer file in [2] - consist of the following consecutive parts:

- the File header, containing the attributes as specified in [2];
- the raw information to be exchanged, also called local files in [1].

When using one of the (optional) compression methods the raw information - not including the File Header that holds the attributes - shall be compressed and formatted before transmission. After reception the transmitted information shall be uncompressed and unformatted accordingly.

The compress / uncompress and format / unformat procedure may:

- either be performed before / after the transmission starts, as depicted in figure 1;
- or (preferably) occur on the fly during the transmission.

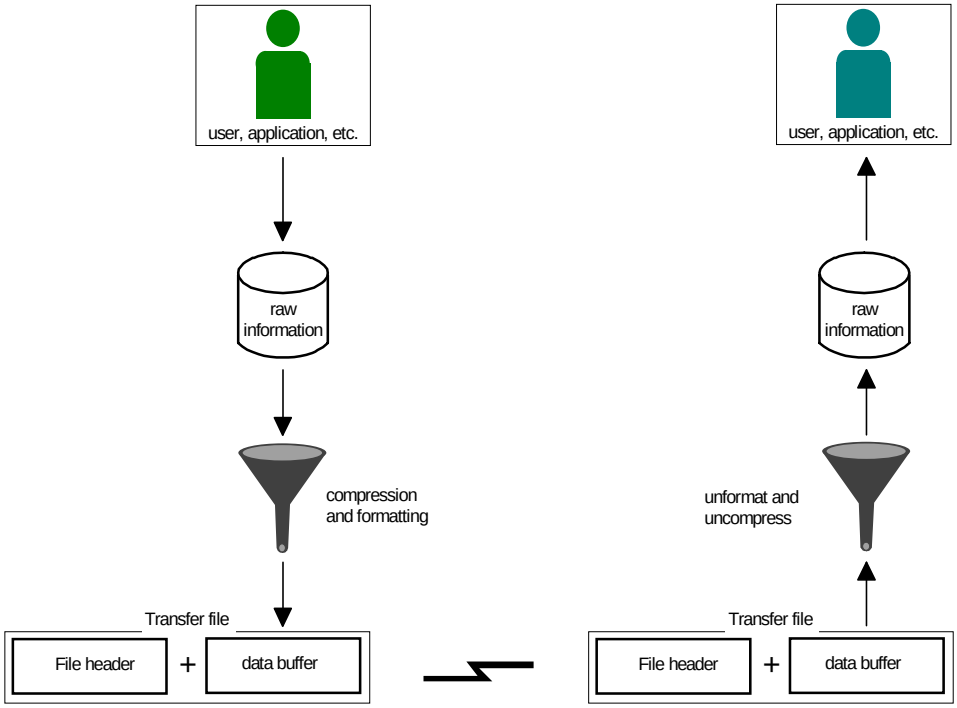


Figure 1: compression and formatting

6.2 Channel aggregation

For the transfer of a large amount of data more bandwidth is sometimes desired than offered by one narrowband ISDN user channel (B-channel). Channel aggregation allows to use several B-channels simultaneously to shorten the transmission time.

Various channel aggregation methods are specified (e.g. BONDING ISO [8] or MLPPP [9], but these methods cannot coexist with the ISDN lower layer protocols for telematic terminals that are used by the EUROFILE transfer profile [1].

Furthermore, the EUROFILE application may desire a means to demand for channel aggregation because it best knows, when the additional bandwidth is really needed. The aggregation mechanisms that are located inside the lower layers of the protocol pillar do lack this ability of control.

Therefore, the present document defines an channel aggregation mechanism based on layer 3 of the protocol pillar [3] and [10]. The channel aggregation mechanism is compatible with the existing profiles and allows for channel aggregation on demand.

6.2.1 General description

To perform properly, the described mechanism must cope with:

- negotiation of the channel aggregation mechanism;
- establishment of additional channels;
- synchronization of the data stream;
- tear down of the channels;
- error treatment.

To shelter the remote side from misbehaving, the EUROFILE application of the calling side - the side that desires to use channel aggregation - shall first negotiate the capability of channel aggregation with the remote side.

After successful negotiation of the capability to channel aggregate and with the knowledge that additional channels are available, the EUROFILE application shall be able to establish and to release additional user channels on demand.

When channel aggregation is in effect, the EUROFILE application is in charge of synchronizing the data streams transmitted through the channels. In case an error occurs in the lower layers, the EUROFILE application may tear down all connections and indicate an error.

Under normal - error free - circumstances the EUROFILE application should be able to increase or shrink the amount of channels aggregated on demand, even during a transmission phase, without any loss of data. However, it may be decided by the EUROFILE application itself, how far such dynamic behaviour it is willing to implement.

6.2.2 Negotiation

The optional feature of channel aggregation defined by the present document shall be negotiated using the user data field of the level 3 call packet [3] and [10] when establishing the layer 3 connection.

The caller - the master - shall initiate the negotiation process by inserting the negotiation request parameters into the user data field of the layer 3 call packet.

The called party - the slave - shall confirm positive or negative the capability to channel aggregate. If the called party is unaware of the possibility to negotiate this capability, it may simply ignore the request parameters of the user data field. If the caller does not receive a positive confirmation it shall refrain from using the channel aggregation feature during this communication with this called party.

During the negotiation process, channel aggregation reference identifiers (CAR-IDs) shall be exchanged. The caller shall compute his CAR-ID and place it into the user data of the call packet. In case of a positive confirmation, the response shall contain the CAR-ID of the called party.

The CAR-IDs computed shall be unique within the communication address, through which the DTEs are addressed. It is left to EUROFILE application to achieve this uniqueness.

The uniqueness of the CAR-IDs and the fact that a reference can be made by either communicating side, now allows both sides, the master and the slave, to initiate additional establishment of channels. However, the capability of the master to handle the establishment of additional channels initiated by the slave is also subject of negotiation, for which related parameters shall be provided in the user data of the layer 3 call packet.

The encoding of the user data of the layer 3 call packet is described in subclause 7.2.1.

6.2.3 Establishment of additional channels

Since (successful) negotiation enforces a layer 3 connection, the first channel of the channel aggregate is definitely established when the negotiation is positive confirmed. This channel is called the main channel for reference. Additional channels to be established are called linked channels.

NOTE 1: For the purpose of the present document the term master (vs. the term slave) always identifies the party that initiated the establishment of the main channel.

As a general rule, linked channels may be created or destroyed at any time without affecting the main channel. Disconnecting or clearing the main channel however, shall force all linked channels to be closed.

To establish an additional channel, i.e. a linked channel, the side demanding for it simply shall connect to the same remote and issue a layer 3 call packet containing user data very similar to that used during negotiation. The channel aggregation capable remote side shall recognize this channel as linked rather than independent and shall confirm the additional layer 3 connection accordingly.

NOTE 2: The capability of the master to accept the initiation of the slave to establish linked channels is subject of negotiation.

During the establishment of the linked channels the value of the so called channel sequence number (CSN) shall be agreed. It is the role of the master to justify the CSN. In case the master initiates the linked channel, the slave simply confirms the CSN value given. In case the slave initiates the linked channel, the master inserts the correct CSN value into the positive confirmation.

The master shall allocate the CSN according to the following rules:

- the CSN of the main channel is always 0 (zero);
- the CSN of a linked channel is a positive integer number between 1 and the maximum amount of established linked channels. Expressed in a formula:
  - $1 \leq CSN \leq n$;
  - $CSN \in N > 0$;
  - n = maximum amount of linked channels.
- a new CSN shall be allocated in sequence, i.e. the new CSN shall receive the CSN of the recent linked channel incremented by one. If it is the first linked channel it shall be set to 1 (one).

Application of this rules leads to a reservoir of consecutive CSNs with the most recently established linked channel holding the highest CSN and the main channel holding the CSN with a value of 0 (zero).

The encoding of the user data of the layer 3 call packet is described in subclause 7.2.2.

6.2.4 Release of additional channels

Linked channels shall be released by use of the layer 3 clearing packet. The information about the channel aggregation shall be passed in the user data field.

In order to be able to dynamically shrink and increase the amount of linked channels even during data transfer, the linked channel with the highest CSN shall be released (cleared) first.

Disconnection of the main channel shall enforce disconnection of all linked channels.

The encoding of the user data of the layer 3 clearing packet is described in subclause 7.2.3.

6.2.5 Data Transfer

TDU's shall be conveyed either through the main channel or through linked channels. In order to get a maximum of efficiency, the application shall use the linked channels for data transfer if the length of the data stream is larger than the packet size. The application choose for each data stream the number of linked channels which will be used. A header is inserted to each data stream conveyed through more than one linked channels.

A data stream shall be divided in n parts which will be transmitted over n linked channels. Each part will have a two octets header prior to the data's.

The sending side shall place the first part of the data stream into the channel holding the lowest CSN, which, in any case, is the main channel. The next part it shall place into the channel holding the CSN that computes according to the formula:

- CSN_{next} = (CSN + 1) modulo n.

Where n equals the total count of channels, i.e. the main channel plus the amount of linked channels.

EXAMPLE: There are 3 linked channels established. Then the total count of channels equals 4 (n = 4). According to the formula, the sender will assign the following sequence of CSNs: 0, 1, 2, 3, 0, 1, 2, 3, 0, 1 etc.

The receiving side will use the headers to assemble the arriving packets accordingly.

If during data transmission additional linked channels are initiated for establishment or indicated to be released, it is left to the EUROFILE application to cope with this situation. In general it should be possible for the application to handle this dynamic behaviour, provided the establishment and clearing performs orderly and not erroneous.

NOTE: The capability to cope with the dynamic behaviour during data transfer should be indicated during negotiation.

If during transmission of data a linked channel is indicated to be released, the sending side should ensure that no unacknowledged data packet is on its way through that channel before responding.

6.2.6 Error management

The EUROFILE implementation has many options to handle error situations. In any case, it has the option to disconnect completely or - preferably - to disconnect the linked channels and to re-synchronize at an appropriate procedural state.

7 Encoding

To specify parameter values, the present document uses the same conventions as ETS 300 383 [1] and ETS 300 075 [2]. The notation used to specify the value of an octet is shown in table 1.

Table 1: notation used for presentation of octet values

Notation	Example	Explanation
nibble notation	0/0 … 15/15	high nibble / low nibble = most significant 4 bits / least 4 significant bits, both presented as decimal number.
hex notation	'00'H … 'FF'H	the octet is presented as hexadecimal number.
decimal notation	0..255	the octet is presented as decimal number
bit notation	xxxx-xxxx	presented as 8 bits, the least significant bit is the one on the right hand side.
ASCII notation	'C'	the octet value is corresponds to the ASCII code that presents the character

7.1 Compression service

To implement the compression service, three functionalities shall be provided:

- the identification of the compression mode during negotiation and during the mass transfer phase;
- the DEFLATE compression algorithm;
- the formatting and encoding of the compressed data and proper feed into the transfer file.

7.1.1 Compression mode

The compression mode of the present document - the *balanced compression* mode - has a parameter value of 4/2 ('42'H) that is to be used during negotiation and during the mass transfer phase. Table 2 shows all values that are valid for the compression mode.

Table 2: Valid compression modes

Octet coding	Description	according to...
4/0	basic compression mode	ETS 300 075 [2]
4/1	"high efficiency" compression mode	ETS 300 075 [2]
4/2	*balanced compression* mode	the present document
4/15	"application defined" compression mode	ETS 300 075 [2]

7.1.1.1 Negotiation

The negotiation of the compression mode occurs through the T-ACCESS primitive when establishing the Access regime [2]. As defined in subclauses 8.1.2.1 and 8.1.2.2 of ETS 300 383 [1] the capability list needed for the negotiation is located at octet offset n° 4 in the User data parameter of the T-ACCESS primitive.

Table 3: User data of T-ACCESS primitive

Octet N°	Content	Value(s)	defined by…
1	Group compatibility (1st octet)		ETS 300 075 [2], § 7.4.1
2	Group compatibility (2nd octet)		ETS 300 075 [2], § 7.4.1
3	Navigation indicator support	4/15 or 0/0 (yes or no)	ETS 300 383 [1], § 8.1.2.1
4 …	Capability list	5/15 …	ETS 300 383 [1], § 8.1.2.1

As depicted by figure 2 the capability list may consist of several capability information elements, which are coded according to the type-length-value (TLV) scheme for the parameter field of the Teleservice Data Units (TDUs) [2].

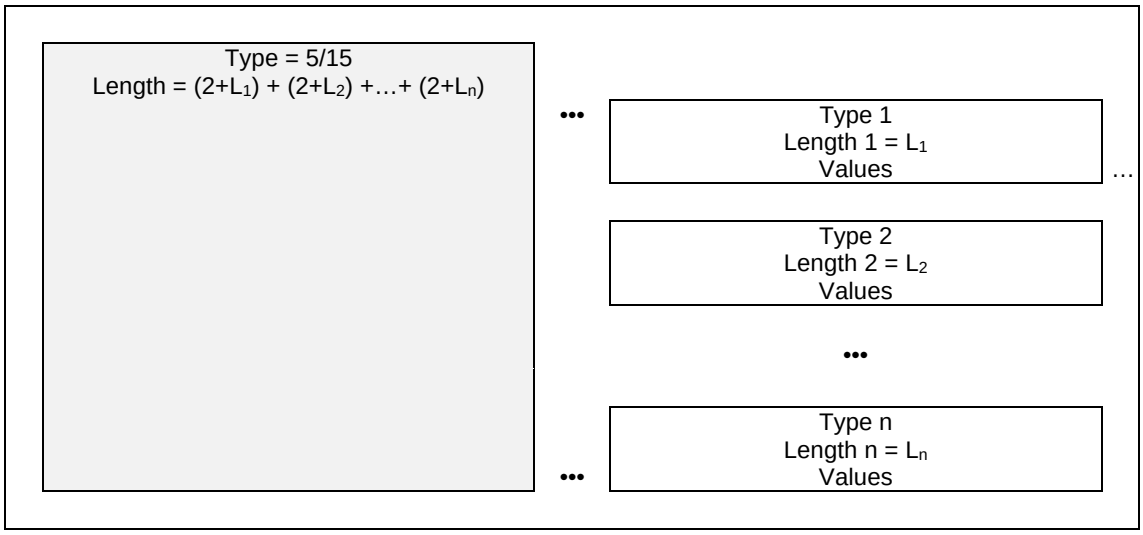


Figure 2: TLV coding of capability list

According to ETS 300 383 [1] two capability information elements are defined:

- the list of supported compression modes;
- parameters for the "high efficiency" compression mode.

Table 4 and table 5 show the coding of the related capability information elements.

Table 4: Capability information element: list of supported compression modes

	Octet coding	Explanation
Type	6/0	identifies list of supported compression modes
Length	0…4	amount of compression modes supported. If none, length shall be set to 0 (zero).
Value(s)	4/0	basic compression mode
and/or	4/1	"high efficiency" compression mode
and/or	4/2	*balanced compression* mode
and/or	4/15	"application defined" compression mode

Table 5: Capability information element: parameters for the "high efficiency" compression mode

	Octet coding	Explanation
Type	6/1	identifies "high efficiency" compression mode parameters
Length	2	fixed
Value(s)	N1	maximum supported value for N1 (see ETS 300 383 [1])
and	N7	maximum supported value for N7 (see ETS 300 383 [1])

7.1.1.2 Mass Transfer Phase

When a compression mode has been negotiated successfully, the use of the compression during the mass transfer shall be indicated in the Compression mode attribute of the related File Header [2]. The TLV coding of the valid compression modes is shown in table 6.

Table 6: Compression mode attribute

	Octet coding	Explanation
Type	2/14	Compression mode attribute identifier
Length	1	fixed
Value(s)	0/0	no compression - default.
or	4/0	basic compression mode
or	4/1	"high efficiency" compression mode
or	4/2	*balanced compression* mode
or	4/15	"application defined" compression mode

7.1.2 Compression Algorithm

The compression algorithm proposed to be used is compatible with the DEFLATE compression format.

The algorithm itself is not specified as normative part within the present document. It is rather contained as informative annex A, which holds a copy of IETF RFC 1951 [5] as of May 1996.

Even if - strongly spoken - an RFC does not represent a standard in the classic sense, the aforementioned RFC does represent a specification that constitutes a de-facto standard. Since it is possible to obtain software modules or libraries able to encode and decode the DEFLATE compression format without affecting any Individual Property Rights (IPRs) or patents, the format itself does not need to be specified in the usual, normative manner.

In any case, the compression algorithm shall be able to:

- consume an arbitrary amount of uncompressed data and produce data compatible to the DEFLATE compression format;

- consume an arbitrary amount of data compatible to the DEFLATE compression format and produce correctly the uncompressed data.

The whole compressed data produced (or consumed) by this algorithm during one run are called the compressed data set for further reference.

7.1.3 Formatting and Encoding of the Compressed Data

The compressed data sets shall be formatted in a way that is compatible with the so called GZIP file format. The GZIP compatible file format is described in IETF RFC 1952 [6], from where the following description has been extracted. Informative annex C holds a copy of IETF RFC 1952 [6] as of May 1996.

NOTE: Although the format the compressed data sets are presented in is called the GZIP file format, it presents merely a buffer of data rather than a (physical) file. In order not to confuse the reader, the term data buffer is used instead to identify this kind of data.

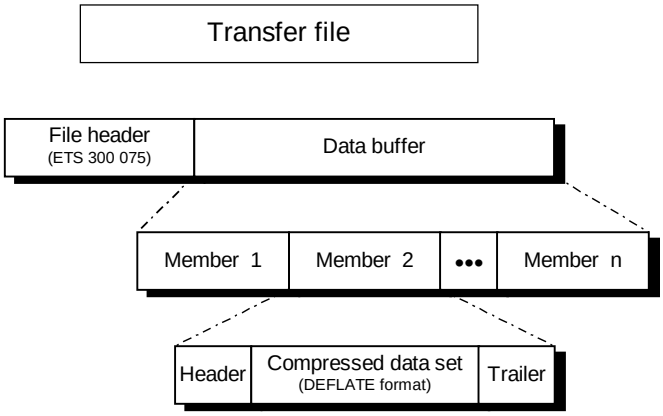
7.1.3.1 Data buffer format

The data buffer format - since compatible to the GZIP file format - allows for several compressed data sets - that is the DEFLATE compatible compressed data - to be concatenated in sequence.

A compressed data set when contained in the data buffer shall be embedded within header and trailer structures. A unit that consists of a header, the compressed data set and a trailer is called a member.

If the data buffer is to contain several members, the members shall appear in the data buffer one after the other, with no additional information before, between or after them. The data buffer shall consist of one or more members.

The layout of the whole transfer file, i.e. the information that is actually transferred during the mass transfer phase, is depicted in figure 3.



NOTE: An implementation is not forced to construct and store the transfer file physically prior to sending or after receiving it. The implementation may perform the build up and split of the transfer file dynamically, chunk by chunk on the fly during the transmission process.

Figure 3: Format of the transfer file using the GZIP file format

7.1.3.2 Member format

As depicted in figure 3, each member consists of a header, the compressed data set (DEFLATE format) and a trailer. The structures of the header and trailer are not further specified. The can be taken as indicated in informative annex B.

7.1.3.3 CRC Generator

The member format imposes generation of a Cyclic Redundancy Check (CRC) value. This CRC value is computed relative to a base of 32 bits. The computation shall follow the rules laid down in [4].

7.2 Channel aggregation

To implement the channel aggregation service, three functionalities shall be provided:

- negotiation of the channel aggregation and related capabilities;
- establishment of linked channels;
- release of linked channels.

All three functionalities shall be implemented through parameters placed inside the user data field of the level 3 call packet and clearing packet, respectively.

Because the EUROFILE profile [1] uses some of the first octets of the user data field, the parameters shall be placed at offset '10'H of the user data, i.e., they shall start occupying the 17th octet and above, freeing a gap of 16 octets for the purpose of the EUROFILE profile.

Since the size needed for the user data field exceeds 16 octets, the layer 3 packets shall make use of the *fast select* facility as described in ITU-T Recommendation X.25 [3]. The *fast select* facility then will provide a user data field size of 128 octets.

The parameter sets needed for the channel aggregation service shall be coded according to the type-length-value (TLV) scheme for the parameter field of the Teleservice Data Units (TDUs) [2].

The parameter sets are introduced by a type value of 2/10 followed by the length of the actual parameter set. Figure 4 illustrates the layout.

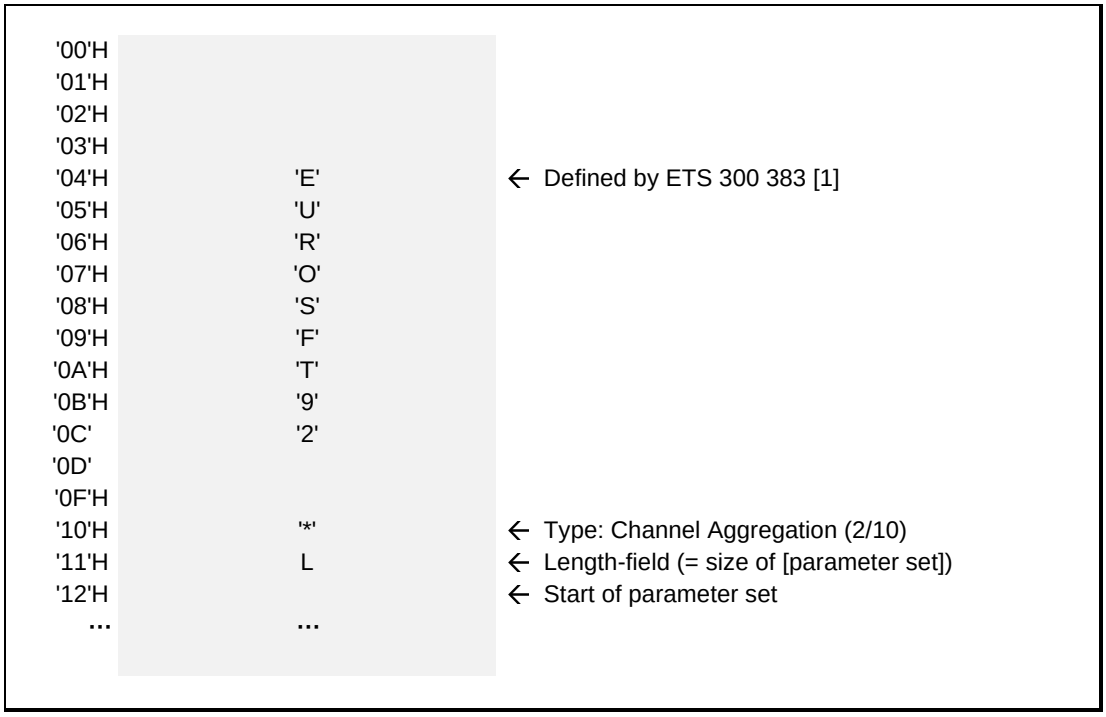


Figure 4: Layout of layer 3 packet user data used for channel aggregation

7.2.1 Negotiation

The negotiation parameter set is used in the layer 3 call packet on establishment of the main channel. The coding of the parameter set is shown in table 7.

Table 7: Negotiation parameter set

Octet	Parameter	Value(s)	Explanation
1	command	'N'	fixed; indicates a negotiation
2	channels	octet =	
3	dynamic	0/1 or 0/0 (yes or no)	negotiates the "dynamic behaviour"
4	both-sides	0/1 or 0/0 (yes or no)	negotiates the "bi-directional" initiation of establishment of linked channels
5	CAR-ID length	octet =	gives the length of the CAR-ID that follows
6…	CAR-ID	string of octets	the CAR-ID

The master shall set the parameters on request as follows:

- the command parameter shall be set to 4/14 ('N');
- the channels parameter shall be set to the number of additional channels desired to negotiate;
- the dynamic parameter shall be set to 0/1 if dynamic allocation or shrinking of linked channels is desired, otherwise shall be set to 0/0;
- the both-sides parameter shall be set to 0/1 if the slave shall be allowed to initiate establishment of linked channels;
- the CAR-ID length parameter shall be set to the size of the CAR-ID of the master and the CAR-ID itself shall follow in the CAR-ID parameter.

The slave shall confirm the parameters as follows:

- the command parameter shall be set to 4/14 ('N');
- the channels parameter shall be set to the number of additional channels the slave confirms. A setting of 0/0 indicates that channel aggregation is not supported;
- the dynamic parameter shall be reset to 0/0 if dynamic behaviour is not desired by the slave, otherwise the setting of the request shall stay unchanged;
- the both-sides parameter shall be reset to 0/0 if the slave refrains from initiating establishment of linked channels, otherwise the setting of the request shall stay unchanged;
- the CAR-ID length parameter shall be set to the size of the CAR-ID of the slave and the CAR-ID itself shall follow in the CAR-ID parameter.

7.2.2 Establishment of Linked Channels

The establish parameter set is used in the layer 3 call packet on establishment of a linked channel. The coding of the parameter set is shown in table 8.

When initiating establishment of a linked channel, the CAR-ID shall be copied into the OSI NSAP address prior to issue the call packet.

Table 8: Linked Channel Establishment parameter set

Octet	Parameter	Value(s)	Explanation
1	command	'E'	fixed; indicates linked channel establishment
2	CSN	octet =	channel sequence number
3	spare1	0/0	fixed
4	spare1	0/0	fixed
5	CAR-ID length	octet =	gives the length of the CAR-ID that follows
6…	CAR-ID	string of octets	the CAR-ID

The caller (master or slave) shall set the parameters on request as follows:

- the command parameter shall be set to 4/5 ('E');
- if the master is the caller then the CSN parameter shall be set to the value computed by the master, otherwise it shall be set to 0/0;
- the spare1 and spare2 parameters shall be set to 0/0;
- the CAR-ID length parameter shall be set to the size of the CAR-ID of the caller and the CAR-ID itself shall follow in the CAR-ID parameter.

The called party (master or slave) shall confirm the parameters as follows:

- the command parameter shall be set to 4/5 ('E');
- if the called party is the master it shall set the CSN parameter to the value computed. If the called party is the slave it shall let the parameter unchanged;
- the spare1 and spare2 parameters shall be let unchanged;
- the CAR-ID length parameter shall be set to the size of the CAR-ID of the called party and the CAR-ID itself shall follow in the CAR-ID parameter.

7.2.3 Release of Linked Channels

The release parameter set is used in the layer 3 clearing packet on release of a linked channel. The coding of the parameter set is shown in table 9.

Table 9: Linked Channel Release parameter set

Octet	Parameter	Value(s)	Explanation
1	command	'R'	fixed; indicates linked channel release
2	CSN	octet =	channel sequence number
3	spare1	0/0	fixed
4	spare1	0/0	fixed
5	CAR-ID length	octet =	gives the length of the CAR-ID that follows
6…	CAR-ID	string of octets	the CAR-ID

The caller (master or slave) shall set the parameters on request as follows:

- the command parameter shall be set to 5/2 ('R');
- the CSN parameter shall be set to the value of the most recently obtained linked channel (highest available CSN);
- the spare1 and spare2 parameters shall be set to 0/0;
- the CAR-ID length parameter shall be set to the size of the CAR-ID of the caller and the CAR-ID itself shall follow in the CAR-ID parameter.

The called party (master or slave) shall confirm the parameters as follows:

- the command parameter shall be set to 5/2 ('R');
- the CSN parameter shall be left unchanged;
- the spare1 and spare2 parameters shall be let unchanged;
- the CAR-ID length parameter shall be set to the size of the CAR-ID of the called party and the CAR-ID itself shall follow in the CAR-ID parameter.

7.2.4 Data transfer through linked channels

The data stream may be conveyed through linked channels or through the main channel depending of the size of the stream. If the data stream is conveyed through n linked channels , the stream is divided in n parts. For each part a header is inserted prior to the data's. The structure of this header is the following:

Table 10: Data Header

Octet	Parameter	Value(s)	Explanation
1	Marker	1/6	Marker for a data stream conveyed through linked channels
2 high	Channels number	B3,b2,b1,b0	Number of linked channels
2 low	Channel sequence number	B7,b6,b5,b4	Channel sequence number

Annex A (informative): DEFLATE Compressed Data Format

This annex holds a copy of IETF RFC 1951 [5] as of May 1996.

Network Working Group P. Deutsch
Request for Comments: 1951 Aladdin Enterprises
Category: Informational May 1996

DEFLATE Compressed Data Format Specification version 1.3

Status of This Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

IESG Note:

The IESG takes no position on the validity of any Intellectual Property Rights statements contained in the present document.

Notices

A pointer to the latest version of this and related documentation in HTML format can be found at the URL <ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html>.

Abstract

This specification defines a lossless compressed data format that compresses data using a combination of the LZ77 algorithm and Huffman coding, with efficiency comparable to the best currently available general-purpose compression methods. The data can be produced or consumed, even for an arbitrarily long sequentially presented input data stream, using only an a priori bounded amount of intermediate storage. The format can be implemented readily in a manner not covered by patents.

Table of Contents

1. Introduction

1.1. Purpose

The purpose of this specification is to define a lossless compressed data format that:

* Is independent of CPU type, operating system, file system, and character set, and hence can be used for interchange;
* Can be produced or consumed, even for an arbitrarily long sequentially presented input data stream, using only an a priori bounded amount of intermediate storage, and hence can be used in data communications or similar structures such as Unix filters;
* Compresses data with efficiency comparable to the best currently available general-purpose compression methods, and in particular considerably better than the "compress" program;
* Can be implemented readily in a manner not covered by patents, and hence can be practiced freely;

* Is compatible with the file format produced by the current widely used gzip utility, in that conforming decompressors will be able to read data produced by the existing gzip compressor.

The data format defined by this specification does not attempt to:

* Allow random access to compressed data;
* Compress specialized data (e.g., raster graphics) as well as the best currently available specialized algorithms.

A simple counting argument shows that no lossless compression algorithm can compress every possible input data set. For the format defined here, the worst case expansion is 5 bytes per 32K-byte block, i.e., a size increase of 0.015% for large data sets. English text usually compresses by a factor of 2.5 to 3; executable files usually compress somewhat less; graphical data such as raster images may compress much more.

1.2. Intended audience

This specification is intended for use by implementors of software to compress data into "deflate" format and/or decompress data from "deflate" format.

The text of the specification assumes a basic background in programming at the level of bits and other primitive data representations. Familiarity with the technique of Huffman coding is helpful but not required.

1.3. Scope

The specification specifies a method for representing a sequence of bytes as a (usually shorter) sequence of bits, and a method for packing the latter bit sequence into bytes.

1.4. Compliance

Unless otherwise indicated below, a compliant decompressor must be able to accept and decompress any data set that conforms to all the specifications presented here; a compliant compressor must produce data sets that conform to all the specifications presented here.

1.5. Definitions of terms and conventions used

Byte: 8 bits stored or transmitted as a unit (same as an octet). For this specification, a byte is exactly 8 bits, even on machines

which store a character on a number of bits different from eight. See below, for the numbering of bits within a byte.

String: a sequence of arbitrary bytes.

1.6. Changes from previous versions

There have been no technical changes to the deflate format since version 1.1 of this specification. In version 1.2, some terminology was changed. Version 1.3 is a conversion of the specification to RFC style.

2. Compressed representation overview

A compressed data set consists of a series of blocks, corresponding to successive blocks of input data. The block sizes are arbitrary, except that non-compressible blocks are limited to 65,535 bytes.

Each block is compressed using a combination of the LZ77 algorithm and Huffman coding. The Huffman trees for each block are independent of those for previous or subsequent blocks; the LZ77 algorithm may use a reference to a duplicated string occurring in a previous block, up to 32K input bytes before.

Each block consists of two parts: a pair of Huffman code trees that describe the representation of the compressed data part, and a compressed data part. (The Huffman trees themselves are compressed using Huffman encoding.) The compressed data consists of a series of elements of two types: literal bytes (of strings that have not been detected as duplicated within the previous 32K input bytes), and pointers to duplicated strings, where a pointer is represented as a pair <length, backward distance>. The representation used in the "deflate" format limits distances to 32K bytes and lengths to 258 bytes, but does not limit the size of a block, except for uncompressible blocks, which are limited as noted above.

Each type of value (literals, distances, and lengths) in the compressed data is represented using a Huffman code, using one code tree for literals and lengths and a separate code tree for distances. The code trees for each block appear in a compact form just before the compressed data for that block.

3. Detailed specification

3.1. Overall conventions In the diagrams below, a box like this:

```
+---+
|   | <-- the vertical bars might be missing
+---+
```

represents one byte; a box like this:

```
+==============+
|              |
+==============+
```

represents a variable number of bytes.

Bytes stored within a computer do not have a "bit order", since they are always treated as a unit. However, a byte considered as an integer between 0 and 255 does have a most- and least-significant bit, and since we write numbers with the most-significant digit on the left, we also write bytes with the most-significant bit on the left. In the diagrams below, we number the bits of a byte so that bit 0 is the least-significant bit, i.e., the bits are numbered:

```
+--------+
|76543210|
+--------+
```

Within a computer, a number may occupy multiple bytes. All multi-byte numbers in the format described here are stored with the least-significant byte first (at the lower memory address). For example, the decimal number 520 is stored as:

```
      0        1
+--------+--------+
|00001000|00000010|
+--------+--------+
 ^        ^
 |        |
 |        + more significant byte = 2 x 256
 + less significant byte = 8
```

3.1.1. Packing into bytes

The present document does not address the issue of the order in which

bits of a byte are transmitted on a bit-sequential medium, since the final data format described here is byte- rather than

bit-oriented. However, we describe the compressed block format in below, as a sequence of data elements of various bit lengths, not a sequence of bytes. We must therefore specify how to pack these data elements into bytes to form the final compressed byte sequence:

* Data elements are packed into bytes in order of increasing bit number within the byte, i.e., starting with the least-significant bit of the byte.
* Data elements other than Huffman codes are packed starting with the least-significant bit of the data element.
* Huffman codes are packed starting with the most-significant bit of the code.

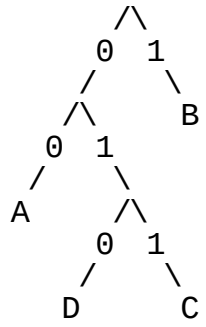
In other words, if one were to print out the compressed data as a sequence of bytes, starting with the first byte at the *right* margin and proceeding to the *left*, with the most-significant bit of each byte on the left as usual, one would be able to parse the result from right to left, with fixed-width elements in the correct MSB-to-LSB order and Huffman codes in bit-reversed order (i.e., with the first bit of the code in the relative LSB position).

3.2. Compressed block format

3.2.1. Synopsis of prefix and Huffman coding

Prefix coding represents symbols from an a priori known alphabet by bit sequences (codes), one code for each symbol, in a manner such that different symbols may be represented by bit sequences of different lengths, but a parser can always parse an encoded string unambiguously symbol-by-symbol.

We define a prefix code in terms of a binary tree in which the two edges descending from each non-leaf node are labeled 0 and 1 and in which the leaf nodes correspond one-for-one with (are labeled with) the symbols of the alphabet; then the code for a symbol is the sequence of 0's and 1's on the edges leading from the root to the leaf labeled with that symbol. For example:



Symbol	Code
A	00
B	1
C	011
D	010

A parser can decode the next symbol from an encoded input stream by walking down the tree from the root, at each step choosing the edge corresponding to the next input bit.

Given an alphabet with known symbol frequencies, the Huffman algorithm allows the construction of an optimal prefix code (one which represents strings with those symbol frequencies using the fewest bits of any possible prefix codes for that alphabet). Such a code is called a Huffman code. (See reference [1] in Chapter 5, references for additional information on Huffman codes.)

Note that in the "deflate" format, the Huffman codes for the various alphabets must not exceed certain maximum code lengths. This constraint complicates the algorithm for computing code lengths from symbol frequencies. Again, see Chapter 5, references for details.

3.2.2. Use of Huffman coding in the "deflate" format

The Huffman codes used for each alphabet in the "deflate" format have two additional rules:

* All codes of a given bit length have lexicographically consecutive values, in the same order as the symbols they represent;

* Shorter codes lexicographically precede longer codes.

We could recode the example above to follow this rule as follows, assuming that the order of the alphabet is ABCD:

Symbol	Code
A	10
B	0
C	110
D	111

I.e., 0 precedes 10 which precedes 11x, and 110 and 111 are lexicographically consecutive.

Given this rule, we can define the Huffman code for an alphabet just by giving the bit lengths of the codes for each symbol of the alphabet in order; this is sufficient to determine the actual codes. In our example, the code is completely defined by the sequence of bit lengths (2, 1, 3, 3). The following algorithm generates the codes as integers, intended to be read from most- to least-significant bit. The code lengths are initially in tree[I].Len; the codes are produced in tree[I].Code.

1) Count the number of codes for each code length. Let bl_count[N] be the number of codes of length N, N >= 1.

2) Find the numerical value of the smallest code for each code length:

```
code = 0;
bl_count[0] = 0;
for (bits = 1; bits <= MAX_BITS; bits++) {
    code = (code + bl_count[bits-1]) << 1;
    next_code[bits] = code;
}
```

3) Assign numerical values to all codes, using consecutive values for all codes of the same length with the base values determined at step 2. Codes that are never used (which have a bit length of zero) must not be assigned a value.

```
for (n = 0; n <= max_code; n++) {
    len = tree[n].Len;
    if (len != 0) {
        tree[n].Code = next_code[len];
        next_code[len]++;
    }
```

```
      }
```

Example:

Consider the alphabet ABCDEFGH, with bit lengths (3, 3, 3, 3, 3, 2, 4, 4). After step 1, we have:

N	bl_count[N]
2	1
3	5
4	2

Step 2 computes the following next_code values:

N	next_code[N]
1	0
2	0
3	2
4	14

Step 3 produces the following code values:

Symbol	Length	Code
A	3	010
B	3	011
C	3	100
D	3	101
E	3	110
F	2	00
G	4	1110
H	4	1111

3.2.3. Details of block format

Each block of compressed data begins with 3 header bits containing the following data:

first bit	BFINAL
next 2 bits	BTYPE

Note that the header bits do not necessarily begin on a byte boundary, since a block does not necessarily occupy an integral number of bytes.

BFINAL is set if and only if this is the last block of the data set.

BTYPE specifies how the data are compressed, as follows:

```
00 - no compression
01 - compressed with fixed Huffman codes
10 - compressed with dynamic Huffman codes
11 - reserved (error)
```

The only difference between the two compressed cases is how the Huffman codes for the literal/length and distance alphabets are defined.

In all cases, the decoding algorithm for the actual data is as follows:

```
do
   read block header from input stream.
   if stored with no compression
      skip any remaining bits in current partially
         processed byte
      read LEN and NLEN (see next section)
      copy LEN bytes of data to output
   otherwise
      if compressed with dynamic Huffman codes
         read representation of code trees (see
            subsection below)
      loop (until end of block code recognized)
         decode literal/length value from input stream
         if value < 256
            copy value (literal byte) to output stream
         otherwise
            if value = end of block (256)
               break from loop
            otherwise (value = 257..285)
               decode distance from input stream

               move backwards distance bytes in the output
               stream, and copy length bytes from this
               position to the output stream.
      end loop
while not last block
```

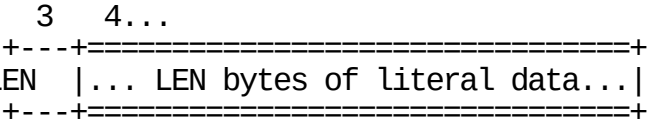
Note that a duplicated string reference may refer to a string in a previous block; i.e., the backward distance may cross one or more block boundaries. However a distance cannot refer past the beginning of the output stream. (An application using a

preset dictionary might discard part of the output stream; a distance can refer to that part of the output stream anyway) Note also that the referenced string may overlap the current position; for example, if the last 2 bytes decoded have values X and Y, a string reference with <length = 5, distance = 2> adds X,Y,X,Y,X to the output stream.

We now specify each compression method in turn.

3.2.4. Non-compressed blocks (BTYPE=00)

Any bits of input up to the next byte boundary are ignored. The rest of the block consists of the following information:



LEN is the number of data bytes in the block. NLEN is the one's complement of LEN.

3.2.5. Compressed blocks (length and distance codes)

As noted above, encoded data blocks in the "deflate" format consist of sequences of symbols drawn from three conceptually distinct alphabets: either literal bytes, from the alphabet of byte values (0..255), or <length, backward distance> pairs, where the length is drawn from (3..258) and the distance is drawn from (1..32,768). In fact, the literal and length alphabets are merged into a single alphabet (0..285), where values 0..255 represent literal bytes, the value 256 indicates end-of-block, and values 257..285 represent length codes (possibly in conjunction with extra bits following the symbol code) as follows:

Code	Extra Bits	Length(s)	Code	Extra Bits	Lengths	Code	Extra Bits	Length(s)
257	0	3	267	1	15,16	277	4	67-82
258	0	4	268	1	17,18	278	4	83-98
259	0	5	269	2	19-22	279	4	99-114
260	0	6	270	2	23-26	280	4	115-130
261	0	7	271	2	27-30	281	5	131-162
262	0	8	272	2	31-34	282	5	163-194
263	0	9	273	3	35-42	283	5	195-226
264	0	10	274	3	43-50	284	5	227-257
265	1	11,12	275	3	51-58	285	0	258
266	1	13,14	276	3	59-66			

The extra bits should be interpreted as a machine integer stored with the most-significant bit first, e.g., bits 1110 represent the value 14.

Code	Extra Bits	Dist	Code	Extra Bits	Dist	Code	Extra Bits	Distance
0	0	1	10	4	33-48	20	9	1025-1536
1	0	2	11	4	49-64	21	9	1537-2048
2	0	3	12	5	65-96	22	10	2049-3072
3	0	4	13	5	97-128	23	10	3073-4096
4	1	5,6	14	6	129-192	24	11	4097-6144
5	1	7,8	15	6	193-256	25	11	6145-8192
6	2	9-12	16	7	257-384	26	12	8193-12288
7	2	13-16	17	7	385-512	27	12	12289-16384
8	3	17-24	18	8	513-768	28	13	16385-24576
9	3	25-32	19	8	769-1024	29	13	24577-32768

3.2.6. Compression with fixed Huffman codes (BTYPE=01)

The Huffman codes for the two alphabets are fixed, and are not represented explicitly in the data. The Huffman code lengths for the literal/length alphabet are:

Lit Value	Bits	Codes
0 - 143	8	00110000 through 10111111
144 - 255	9	110010000 through 111111111
256 - 279	7	0000000 through 0010111
280 - 287	8	11000000 through 11000111

The code lengths are sufficient to generate the actual codes, as described above; we show the codes in the table for added clarity. Literal/length values 286-287 will never actually occur in the compressed data, but participate in the code construction.

Distance codes 0-31 are represented by (fixed-length) 5-bit codes, with possible additional bits as shown in the table shown in Paragraph 3.2.5, above. Note that distance codes 30-31 will never actually occur in the compressed data.

3.2.7. Compression with dynamic Huffman codes (BTYPE=10)

The Huffman codes for the two alphabets appear in the block immediately after the header bits and before the actual compressed data, first the literal/length code and then the distance code. Each code is defined by a sequence of code lengths, as discussed in Paragraph 3.2.2, above. For even greater compactness, the code length sequences themselves are compressed using a Huffman code. The alphabet for code lengths is as follows:

```
0 - 15: Represent code lengths of 0 - 15
    16: Copy the previous code length 3 - 6 times.
        The next 2 bits indicate repeat length
              (0 = 3, ... , 3 = 6)
           Example:  Codes 8, 16 (+2 bits 11),
                     16 (+2 bits 10) will expand to
                     12 code lengths of 8 (1 + 6 + 5)
    17: Repeat a code length of 0 for 3 - 10 times.
        (3 bits of length)
    18: Repeat a code length of 0 for 11 - 138 times
        (7 bits of length)
```

A code length of 0 indicates that the corresponding symbol in the literal/length or distance alphabet will not occur in the block, and should not participate in the Huffman code construction algorithm given earlier. If only one distance code is used, it is encoded using one bit, not zero bits; in this case there is a single code length of one, with one unused code. One distance code of zero bits means that there are no distance codes used at all (the data is all literals).

We can now define the format of the block:

```
5 Bits: HLIT, # of Literal/Length codes - 257 (257 - 286)
5 Bits: HDIST, # of Distance codes - 1        (1 - 32)
4 Bits: HCLEN, # of Code Length codes - 4     (4 - 19)
```

(HCLEN + 4) x 3 bits: code lengths for the code length alphabet given just above, in the order: 16, 17, 18, 0, 8, 7, 9, 6, 10, 5, 11, 4, 12, 3, 13, 2, 14, 1, 15

These code lengths are interpreted as 3-bit integers (0-7); as above, a code length of 0 means the corresponding symbol (literal/length or distance code length) is not used.

HLIT + 257 code lengths for the literal/length alphabet, encoded using the code length Huffman code

HDIST + 1 code lengths for the distance alphabet, encoded using the code length Huffman code

The actual compressed data of the block, encoded using the literal/length and distance Huffman codes

The literal/length symbol 256 (end of data), encoded using the literal/length Huffman code

The code length repeat codes can cross from HLIT + 257 to the HDIST + 1 code lengths. In other words, all code lengths form a single sequence of HLIT + HDIST + 258 values.

3.3. Compliance

A compressor may limit further the ranges of values specified in the previous section and still be compliant; for example, it may limit the range of backward pointers to some value smaller than 32K. Similarly, a compressor may limit the size of blocks so that a compressible block fits in memory.

A compliant decompressor must accept the full range of possible values defined in the previous section, and must accept blocks of arbitrary size.

4. Compression algorithm details

While it is the intent of the present document to define the "deflate" compressed data format without reference to any particular compression algorithm, the format is related to the compressed formats produced by LZ77 (Lempel-Ziv 1977, see reference [2] below); since many variations of LZ77 are patented, it is strongly recommended that the implementor of a compressor follow the general algorithm presented here, which is known not to be patented per se. The material in this section is not part of the definition of the

specification per se, and a compressor need not follow it in order to be compliant.

The compressor terminates a block when it determines that starting a new block with fresh trees would be useful, or when the block size fills up the compressor's block buffer.

The compressor uses a chained hash table to find duplicated strings, using a hash function that operates on 3-byte sequences. At any given point during compression, let XYZ be the next 3 input bytes to be examined (not necessarily all different, of course). First, the compressor examines the hash chain for XYZ. If the chain is empty, the compressor simply writes out X as a literal byte and advances one byte in the input. If the hash chain is not empty, indicating that the sequence XYZ (or, if we are unlucky, some other 3 bytes with the same hash function value) has occurred recently, the compressor compares all strings on the XYZ hash chain with the actual input data sequence starting at the current point, and selects the longest match.

The compressor searches the hash chains starting with the most recent strings, to favor small distances and thus take advantage of the Huffman encoding. The hash chains are singly linked. There are no deletions from the hash chains; the algorithm simply discards matches that are too old. To avoid a worst-case situation, very long hash chains are arbitrarily truncated at a certain length, determined by a run-time parameter.

To improve overall compression, the compressor optionally defers the selection of matches ("lazy matching"): after a match of length N has been found, the compressor searches for a longer match starting at the next input byte. If it finds a longer match, it truncates the previous match to a length of one (thus producing a single literal byte) and then emits the longer match. Otherwise, it emits the original match, and, as described above, advances N bytes before continuing.

Run-time parameters also control this "lazy match" procedure. If compression ratio is most important, the compressor attempts a complete second search regardless of the length of the first match. In the normal case, if the current match is "long enough", the compressor reduces the search for a longer match, thus speeding up the process. If speed is most important, the compressor inserts new strings in the hash table only when no match was found, or when the match is not "too long". This degrades the compression ratio but saves time since there are both fewer insertions and fewer searches.

5. References

[1] Huffman, D. A., "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the Institute of Radio Engineers, September 1952, Volume 40, Number 9, pp. 1098-1101.

[2] Ziv J., Lempel A., "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343.

[3] Gailly, J.-L., and Adler, M., ZLIB documentation and sources, available in ftp://ftp.uu.net/pub/archiving/zip/doc/

[4] Gailly, J.-L., and Adler, M., GZIP documentation and sources, available as gzip-*.tar in ftp://prep.ai.mit.edu/pub/gnu/

[5] Schwartz, E. S., and Kallick, B. "Generating a canonical prefix encoding." Comm. ACM, 7,3 (Mar. 1964), pp. 166-169.

[6] Hirschberg and Lelewer, "Efficient decoding of prefix codes," Comm. ACM, 33,4, April 1990, pp. 449-459.

6. Security Considerations

Any data compression method involves the reduction of redundancy in the data. Consequently, any corruption of the data is likely to have severe effects and be difficult to correct. Uncompressed text, on the other hand, will probably still be readable despite the presence of some corrupted bytes.

It is recommended that systems using this data format provide some means of validating the integrity of the compressed data. See reference [3], for example.

7. Source code

Source code for a C language implementation of a "deflate" compliant compressor and decompressor is available within the zlib package at ftp://ftp.uu.net/pub/archiving/zip/zlib/.

8. Acknowledgements

Trademarks cited in the present document are the property of their respective owners.

Phil Katz designed the deflate format. Jean-Loup Gailly and Mark Adler wrote the related software described in this specification. Glenn Randers-Pehrson converted the present document to RFC and HTML format.

9. Author's Address

L. Peter Deutsch
Aladdin Enterprises
203 Santa Margarita Ave.
Menlo Park, CA 94025

Phone: (415) 322-0103 (AM only)
FAX: (415) 322-1734
EMail: <ghost@aladdin.com>

Questions about the technical content of this specification can be sent by email to:

Jean-Loup Gailly <gzip@prep.ai.mit.edu> and
Mark Adler <madler@alumni.caltech.edu>

Editorial comments on this specification can be sent by email to:

L. Peter Deutsch <ghost@aladdin.com> and
Glenn Randers-Pehrson <randeg@alumni.rpi.edu>

Annex B (informative): GZIP File Format

This annex holds a copy of IETF RFC 1952 [6] as of May 1996.

Network Working Group P. Deutsch
Request for Comments: 1952 Aladdin Enterprises
Category: Informational May 1996

GZIP file format specification version 4.3

Status of This Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

IESG Note:

The IESG takes no position on the validity of any Intellectual Property Rights statements contained in the present document.

Notices

A pointer to the latest version of this and related documentation in HTML format can be found at the URL <ftp://ftp.uu.net/graphics/png/documents/zlib/zdoc-index.html>.

Abstract

This specification defines a lossless compressed data format that is compatible with the widely used GZIP utility. The format includes a cyclic redundancy check value for detecting data corruption. The format presently uses the DEFLATE method of compression but can be easily extended to use other compression methods. The format can be implemented readily in a manner not covered by patents.

Table of Contents

1. Introduction

1.1. Purpose

The purpose of this specification is to define a lossless compressed data format that:

* Is independent of CPU type, operating system, file system, and character set, and hence can be used for interchange;
* Can compress or decompress a data stream (as opposed to a randomly accessible file) to produce another data stream, using only an a priori bounded amount of intermediate storage, and hence can be used in data communications or similar structures such as Unix filters;
* Compresses data with efficiency comparable to the best currently available general-purpose compression methods, and in particular considerably better than the "compress" program;
* Can be implemented readily in a manner not covered by patents, and hence can be practiced freely;
* Is compatible with the file format produced by the current widely used gzip utility, in that conforming decompressors will be able to read data produced by the existing gzip compressor.

The data format defined by this specification does not attempt to:

* Provide random access to compressed data;
* Compress specialized data (e.g., raster graphics) as well as the best currently available specialized algorithms.

1.2. Intended audience

This specification is intended for use by implementors of software to compress data into gzip format and/or decompress data from gzip format.

The text of the specification assumes a basic background in programming at the level of bits and other primitive data representations.

1.3. Scope

The specification specifies a compression method and a file format (the latter assuming only that a file can store a sequence of arbitrary bytes). It does not specify any particular interface to a file system or anything about character sets or encodings (except for file names and comments, which are optional).

1.4. Compliance

Unless otherwise indicated below, a compliant decompressor must be able to accept and decompress any file that conforms to all the specifications presented here; a compliant compressor must produce files that conform to all the specifications presented here. The material in the appendices is not part of the specification per se and is not relevant to compliance.

1.5. Definitions of terms and conventions used

byte: 8 bits stored or transmitted as a unit (same as an octet). (For this specification, a byte is exactly 8 bits, even on machines which store a character on a number of bits different from 8.) See below for the numbering of bits within a byte.

1.6. Changes from previous versions

There have been no technical changes to the gzip format since version 4.1 of this specification. In version 4.2, some terminology was changed, and the sample CRC code was rewritten for clarity and to eliminate the requirement for the caller to do pre- and post-conditioning. Version 4.3 is a conversion of the specification to RFC style.

2. Detailed specification

2.1. Overall conventions

In the diagrams below, a box like this:

```
+---+
|   | <-- the vertical bars might be missing
+---+
```

represents one byte; a box like this:

```
+==============+
|              |
+==============+
```

represents a variable number of bytes.

Bytes stored within a computer do not have a "bit order", since they are always treated as a unit. However, a byte considered as an integer between 0 and 255 does have a most- and least-significant bit, and since we write numbers with the most-significant digit on the left, we also write bytes with the most-significant bit on the left. In the diagrams below, we number the bits of a byte so that bit 0 is the least-significant bit, i.e., the bits are numbered:

```
+--------+
|76543210|
+--------+
```

The present document does not address the issue of the order in which bits of a byte are transmitted on a bit-sequential medium, since the data format described here is byte- rather than bit-oriented.

Within a computer, a number may occupy multiple bytes. All multi-byte numbers in the format described here are stored with the least-significant byte first (at the lower memory address). For example, the decimal number 520 is stored as:

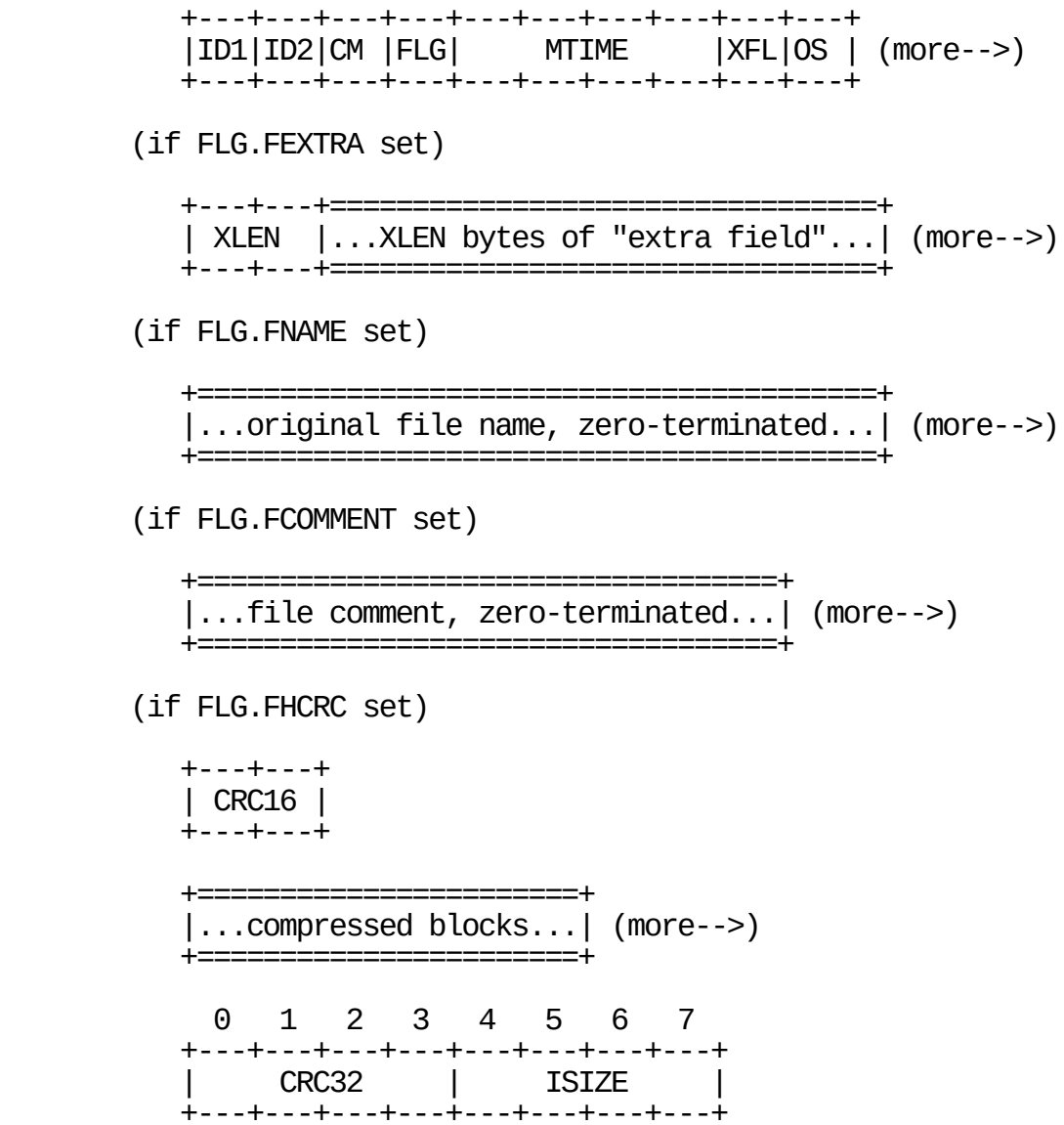
```
      0       1
+--------+--------+
|00001000|00000010|
+--------+--------+
 ^        ^
 |        |
 |        + more significant byte = 2 x 256
 + less significant byte = 8
```

2.2. File format

A gzip file consists of a series of "members" (compressed data sets). The format of each member is specified in the following section. The members simply appear one after another in the file, with no additional information before, between, or after them.

2.3. Member format

Each member has the following structure:



2.3.1. Member header and trailer

ID1 (IDentification 1)
ID2 (IDentification 2)
These have the fixed values ID1 = 31 (0x1f, \037), ID2 = 139 (0x8b, \213), to identify the file as being in gzip format.

CM (Compression Method)
This identifies the compression method used in the file. CM = 0-7 are reserved. CM = 8 denotes the "deflate" compression method, which is the one customarily used by gzip and which is documented elsewhere.

FLG (FLaGs)
This flag byte is divided into individual bits as follows:

bit 0	FTEXT
bit 1	FHCRC
bit 2	FEXTRA
bit 3	FNAME
bit 4	FCOMMENT
bit 5	reserved
bit 6	reserved
bit 7	reserved

If FTEXT is set, the file is probably ASCII text. This is an optional indication, which the compressor may set by checking a small amount of the input data to see whether any non-ASCII characters are present. In case of doubt, FTEXT is cleared, indicating binary data. For systems which have different file formats for ascii text and binary data, the decompressor can use FTEXT to choose the appropriate format. We deliberately do not specify the algorithm used to set this bit, since a compressor always has the option of leaving it cleared and a decompressor always has the option of ignoring it and letting some other program handle issues of data conversion.

If FHCRC is set, a CRC16 for the gzip header is present, immediately before the compressed data. The CRC16 consists of the two least significant bytes of the CRC32 for all bytes of the gzip header up to and not including the CRC16. [The FHCRC bit was never set by versions of gzip up to 1.2.4, even though it was documented with a different meaning in gzip 1.2.4.]

If FEXTRA is set, optional extra fields are present, as described in a following section.

If FNAME is set, an original file name is present, terminated by a zero byte. The name must consist of ISO 8859-1 (LATIN-1) characters; on operating systems using EBCDIC or any other character set for file names, the name must be translated to the ISO LATIN-1 character set. This is the original name of the file being compressed, with any directory components removed, and, if the file being compressed is on a file system with case insensitive names, forced to lower case. There is no original file name if the data was compressed from a source other than a named file; for example, if the source was stdin on a Unix system, there is no file name.

If FCOMMENT is set, a zero-terminated file comment is present. This comment is not interpreted; it is only intended for human consumption. The comment must consist of ISO 8859-1 (LATIN-1) characters. Line breaks should be denoted by a single line feed character (10 decimal).

Reserved FLG bits must be zero.

MTIME (Modification TIME)
This gives the most recent modification time of the original file being compressed. The time is in Unix format, i.e., seconds since 00:00:00 GMT, Jan. 1, 1970. (Note that this may cause problems for MS-DOS and other systems that use local rather than Universal time.) If the compressed data did not come from a file, MTIME is set to the time at which compression started. MTIME = 0 means no time stamp is available.

XFL (eXtra FLags)
These flags are available for use by specific compression methods. The "deflate" method (CM = 8) sets these flags as follows:

```
XFL = 2 - compressor used maximum compression,
          slowest algorithm
XFL = 4 - compressor used fastest algorithm
```

OS (Operating System)
This identifies the type of file system on which compression took place. This may be useful in determining end-of-line convention for text files. The currently defined values are as follows:

```
   0 - FAT filesystem (MS-DOS, OS/2, NT/Win32)
   1 - Amiga
   2 - VMS (or OpenVMS)
   3 - Unix
   4 - VM/CMS
   5 - Atari TOS
   6 - HPFS filesystem (OS/2, NT)
   7 - Macintosh
   8 - Z-System
   9 - CP/M
  10 - TOPS-20
  11 - NTFS filesystem (NT)
  12 - QDOS
  13 - Acorn RISCOS
 255 - unknown
```

XLEN (eXtra LENgth)
If FLG.FEXTRA is set, this gives the length of the optional extra field. See below for details.

CRC32 (CRC-32)
This contains a Cyclic Redundancy Check value of the uncompressed data computed according to CRC-32 algorithm used in the ISO 3309 standard and in section 8.1.1.6.2 of ITU-T recommendation V.42 [4](See http://www.iso.ch for ordering ISO documents. See gopher://info.itu.ch for an online version of ITU-T Recommendation V.42 [4]

ISIZE (Input SIZE)
This contains the size of the original (uncompressed) input data modulo 2^32.

2.3.1.1. Extra field

If the FLG.FEXTRA bit is set, an "extra field" is present in the header, with total length XLEN bytes. It consists of a series of subfields, each of the form:

```
+---+---+---+---+==================================+
|SI1|SI2|  LEN  |... LEN bytes of subfield data ...|
+---+---+---+---+==================================+
```

SI1 and SI2 provide a subfield ID, typically two ASCII letters with some mnemonic value. Jean-Loup Gailly <gzip@prep.ai.mit.edu> is maintaining a registry of subfield IDs; please send him any subfield ID you wish to use. Subfield IDs with SI2 = 0 are reserved for future use. The following IDs are currently defined:

SI1	SI2	Data
0x41 ('A')	0x70 ('P')	Apollo file type information

LEN gives the length of the subfield data, excluding the 4 initial bytes.

2.3.1.2. Compliance

A compliant compressor must produce files with correct ID1, ID2, CM, CRC32, and ISIZE, but may set all the other fields in the fixed-length part of the header to default values (255 for OS, 0 for all others). The compressor must set all reserved bits to zero.

A compliant decompressor must check ID1, ID2, and CM, and provide an error indication if any of these have incorrect values. It must examine FEXTRA/XLEN, FNAME, FCOMMENT and FHCRC at least so it can skip over the optional fields if they are present. It need not examine any other part of the header or trailer; in particular, a decompressor may ignore FTEXT and OS and always produce binary output, and still be compliant. A compliant decompressor must give an error indication if any reserved bit is non-zero, since such a bit could indicate the presence of a new field that would cause subsequent data to be interpreted incorrectly.

3. References

[1] "Information Processing - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No.1" (ISO 8859-1:1987). The ISO 8859-1 (Latin-1) character set is a superset of 7-bit ASCII. Files defining this character set are available as iso_8859-1.* in ftp://ftp.uu.net/graphics/png/documents/

[2] ISO 3309

[3] ITU-T recommendation V.42 [4]

[4] Deutsch, L.P.,"DEFLATE Compressed Data Format Specification", available in ftp://ftp.uu.net/pub/archiving/zip/doc/

[5] Gailly, J.-L., GZIP documentation, available as gzip-*.tar in ftp://prep.ai.mit.edu/pub/gnu/

[6] Sarwate, D.V., "Computation of Cyclic Redundancy Checks via Table Look-Up", Communications of the ACM, 31(8), pp.1008-1013.

[7] Schwaderer, W.D., "CRC Calculation", April 85 PC Tech Journal, pp.118-133.

[8] ftp://ftp.adelaide.edu.au/pub/rocksoft/papers/crc_v3.txt, describing the CRC concept.

4. Security Considerations

Any data compression method involves the reduction of redundancy in the data. Consequently, any corruption of the data is likely to have severe effects and be difficult to correct. Uncompressed text, on the other hand, will probably still be readable despite the presence of some corrupted bytes.

It is recommended that systems using this data format provide some means of validating the integrity of the compressed data, such as by setting and checking the CRC-32 check value.

5. Acknowledgements

Jean-Loup Gailly designed the gzip format and wrote, with Mark Adler, the related software described in this specification. Glenn Randers-Pehrson converted the present document to RFC and HTML format.

6. Author's Address

L. Peter Deutsch
Aladdin Enterprises
203 Santa Margarita Ave.
Menlo Park, CA 94025

Phone: (415) 322-0103 (AM only)
FAX: (415) 322-1734
EMail: <ghost@aladdin.com>

Questions about the technical content of this specification can be sent by email to:

Jean-Loup Gailly <gzip@prep.ai.mit.edu> and
Mark Adler <madler@alumni.caltech.edu>

Editorial comments on this specification can be sent by email to:

L. Peter Deutsch <ghost@aladdin.com> and
Glenn Randers-Pehrson <randeg@alumni.rpi.edu>

7. Appendix: Jean-Loup Gailly's gzip utility

The most widely used implementation of gzip compression, and the original documentation on which this specification is based, were created by Jean-Loup Gailly <gzip@prep.ai.mit.edu>. Since this implementation is a de facto standard, we mention some more of its features here. Again, the material in this section is not part of the specification per se, and implementations need not follow it to be compliant.

When compressing or decompressing a file, gzip preserves the protection, ownership, and modification time attributes on the local file system, since there is no provision for representing protection attributes in the gzip file format itself. Since the file format includes a modification time, the gzip decompressor provides a command line switch that assigns the modification time from the file, rather than the local modification time of the compressed input, to the decompressed output.

8. Appendix: Sample CRC Code

The following sample code represents a practical implementation of the CRC (Cyclic Redundancy Check). (See also ISO 3309 and ITU-T Recommendation V.42 [4]
for a formal specification.)

The sample code is in the ANSI C programming language. Non C users may find it easier to read with these hints:

```
&      Bitwise AND operator.
^      Bitwise exclusive-OR operator.
>>     Bitwise right shift operator. When applied to an
       unsigned quantity, as here, right shift inserts zero
       bit(s) at the left.
!      Logical NOT operator.
++     "n++" increments the variable n.
0xNNN  0x introduces a hexadecimal (base 16) constant.
       Suffix L indicates a long value (at least 32 bits).

/* Table of CRCs of all 8-bit messages. */
unsigned long crc_table[256];

/* Flag: has the table been computed? Initially false. */
int crc_table_computed = 0;

/* Make the table for a fast CRC. */
void make_crc_table(void)
{
  unsigned long c;
```

```
  int n, k;
  for (n = 0; n < 256; n++) {
    c = (unsigned long) n;
    for (k = 0; k < 8; k++) {
      if (c & 1) {
        c = 0xedb88320L ^ (c >> 1);
      } else {
        c = c >> 1;
      }
    }
    crc_table[n] = c;
  }
  crc_table_computed = 1;
}

/*
   Update a running crc with the bytes buf[0..len-1] and return
 the updated crc. The crc should be initialized to zero. Pre- and
 post-conditioning (one's complement) is performed within this
 function so it shouldn't be done by the caller. Usage example:

   unsigned long crc = 0L;

   while (read_buffer(buffer, length) != EOF) {
     crc = update_crc(crc, buffer, length);
   }
   if (crc != original_crc) error();
*/
unsigned long update_crc(unsigned long crc,
                 unsigned char *buf, int len)
{
  unsigned long c = crc ^ 0xffffffffL;
  int n;

  if (!crc_table_computed)
    make_crc_table();
  for (n = 0; n < len; n++) {
    c = crc_table[(c ^ buf[n]) & 0xff] ^ (c >> 8);
  }
  return c ^ 0xffffffffL;
}

/* Return the CRC of the bytes buf[0..len-1]. */
unsigned long crc(unsigned char *buf, int len)
{
  return update_crc(0L, buf, len);
}
```

Bibliography

The following material, though not specifically referenced in the body of the present document (or not publicly available), gives supporting information.

- ETS 300 409: "Integrated Services Digital Network (ISDN); Eurofile transfer teleservice; Service definition".
- ETS 300 325: "Integrated Services Digital Network (ISDN); Programming Communication Interface (PCI) for Euro-ISDN".
- IETF RFC 1618 (May 1994): "PPP over ISDN".
- ETS 300 079: " Integrated Services Digital Network (ISDN); Syntax-based videotex end-to-end protocols; Circuit mode DTE-DTE".

History

Document history				
V1.1.1	May 1999	Membership Approval Procedure	MV 9930:	1999-05-25 to 1999-07-23