SAFT

a Simple ASCII File Transfer system

User and Implementation Documentation

Version 1.2

June 1, 1981

Robert L. Fink, UC/LBL
Örjan Leringe, QZ
Hans Nilsson, KI

Addresses to the authors:

Robert L. Fink
University of California
Lawrence Berkeley Laboratory
Berkely, CA 94720 USA
(Guest researcher at QZ 1980/1981)

Örjan Leringe
Stockholm University Computing Centre, QZ
Fack
S-104 50 Stockholm, Sweden

Hans Nilsson
Systemgruppen
Karolinska institutet
Box 60400
S-104 01 Stockholm, Sweden

Table Of Contents

1. INTRODUCTION TO SAFT

1.1 Overview

SAFT is a system for the transfer of text and simple byte files between smart (i.e. computer controlled) terminals and shared multi-user computer systems. The word SAFT means Simple ASCII File Transfer. The entire design of the SAFT protocol and user interface is intended to allow the use of asynchronous ASCII serial communication lines for the transfer of simple text and byte binary files. These lines are often called V.24 or EIA RS232 lines, and are the lines used to interconnect many interactive terminals and computer ports.

The SAFT system includes both the formal SAFT protocol, for communicating with another SAFT system, and a recommended SAFT human user interactive dialogue, for the user to tell the SAFT system what to do. SAFT is the name given to both the techniques and the programs that implement these techniques.

Hopefully you, as a user of SAFT, will be able to find a SAFT program you might use so that any two smart terminals or computer systems can transfer files between themselves. The entire concept of SAFT is based on the desire to have this wide spread availability and use possible. SAFT accomplishes this through the use of terminal communication lines in the simplest way possible, and by the use of human interaction to invoke, specify and execute the SAFT file transfer desired.

In the remainder of this manual SAFT is described from two points of view, that of the user and that of the implementor. The implementor is the person(s) that will implement a SAFT system for a particular machine environment. The user is the actual person hoping to transfer a file between two computer systems.

The user information specified here must not be construed to be all that is needed to understand how to use SAFT to transfer files of data between two systems. Systems of different types (software or hardware) vary greatly and it is the flexibility of the SAFT system specification, in the area of different system environments, that is the greatest strength and merit of SAFT. Thus the methods of using SAFT may also vary quite widely.

It is assumed that the user is familiar with each of the two systems being used for the SAFT transfer. It is the user's responsibility to enter each system (i.e. login), invoke (i.e. cause to be executed) the SAFT program and specify, for that system environment, the SAFT operation to be performed. In this light the user documentation is oriented towards the description of sample typical environments.

The implementors information is primarily to describe the formal part of the SAFT system, the SAFT protocol. Much more than this is needed by an implementor in practice. In particular, the hosting system environment and a selected user interface for SAFT are obviously intimate parts of the SAFT im-

plementation also.

1.2 Why yet another protocol?

There is a need for simple transferring of character data between dissimilar machines that only have ASCII asynchronous serial communication interfaces to communicate with. Any protocol used for this purpose must have the following characteristics:

- Must be able to move lines of character text between machines that may have different character set and line format standards.
- Must operate as transparently as possible to avoid the possible sensitivity that computer front ends, data communication networks, terminal concentrators and exchanges etc. might have.
- Must reasonably guarantee reliable data delivery without high overhead or complexity.
- Must be as simple to implement as possible to allow its implementation on a minimal 'smart terminal'.
- Must not be proprietary to any given vendor, system or environment.
- Must provide flow control to prevent overloading the communicating systems.
- Must be symmetric in operation so that any two implementations can 'talk' with each other.

The above set of requirements seemed able to be best met by the creation of a new protocol. Ideas from many different protocols were integrated and much 'collective' experience went into the creation of SAFT. Also it was felt that if the requirements for simplicity were to be taken seriously that the proof would be in how easy the (or any) new protocol would be to implement.

In fact, the first implementations of the protocol were developed in BASIC for three different system environments within a few days. The second implementation was done by another person with no previous familiarity with communications protocols. It was completed in a few days. This version was in SIMULA and has been extended to work on two other systems of different type.

As to how well these first versions worked, the person who had implemented the first version in BASIC simply 'tried' the SIMULA version, without any help from its author, and the first transfer of a file was successful.

1.3 SAFT, as protocols go

It must be stated that it is not believed that SAFT is the answer to all communications problems. In fact it is a good deal less capable than many of the currently 'serious' protocols, such as HDLC, SDLC, DDCMP, etc. SAFT had

some rather specific goals to meet and it hopefully meets them, but it does have some drawbacks. At least a few of them are detailed below. To fully understand these drawbacks the user may need to read more information in this document first.

- SAFT does not use the communication line efficiently due to its requirements for transparency. In fact, a file transfer with SAFT may only use the communication line between 20 and 30 percent of its theoretic capacity. This 'high overhead cost' has been acceptable, in fact necessary, in order to keep the protocol simple and transparent.
- Only simple 'network *ISO code*' character mapping is supported due to the differences between systems. A simple byte binary mode is offerred to partially solve more complex requirements.
- Binary data movement is only supported in the most trivial 8-bit form. This is again due to wide differences between systems and the need to keep things simple. Requirements for more complex forms of transfer will need host specific utilities to package the data in the 'local' byte format if one must use SAFT to transfer them.
- There is not a perfect guarantee of reliable message delivery as the SAFT checksum approach only protects the data field. However, it is easy to implement on simple machines with little effort. Thus there is the theoretic possibility of having a bad message escape error detection. Most existing direct or dial-up ASCII interfacing is done in a totally non-checked or protected environment; this does not seem to have caused excessive problems with bad data. Most local direct lines, or those routed through X.25 PAD-s, are very reliable and not at all likely to suffer from 'reliability' problems.
- The recovery, i.e. retransmission, of bad data is restricted to data messages that have had their "=*" header correctly received. This greatly simplifies protocol logic as the correct recovery of all messages would entail very complex state and timing rules that SAFT wishes to avoid for the sake of simplicity. The assumption here is that the things most likely to fail are data messages as they are 42 characters longer than control messages (which are only 2).
- SAFT may appear somewhat clumsy or dirty to use in some system environments and is definitely not suited, in the general case, for use by 'automata' programs (i.e. non human operated). SAFT is intended for use in so many different systems that it relies on the human user to deal with differences and possible inconsistencies between systems. Some help in this area has been given by the specification of 'recommended' (i.e. not mandatory) menu formats for use by the user in starting up SAFT transfers. However, the use of these menu formats is not recommended for 'automata' use, in the general case, although it is quite possible to do so in specific cases.

1.4 Advantages of SAFT

The protocol developed to meet all of the above requirements is called SAFT for Simple ASCII File Transfer. The use of the word ASCII is to represent the basic message structure as being encoded in ASCII on an ASCII line, not because ASCII characters are being transmitted as data in the messages.

Each byte or character of data transferred by SAFT is encoded as two hexadecimally represented (hex doublet) characters using ASCII characters 0 through 9 and A through F to represent each 4-bit hexadecimal value. This results in there being absolutely no data sensitivity of the message to any switches, networks, nodes, frontends or otherwise that the message may be passed through on its way from source to destination (sender to receiver). This means that almost any way of getting a terminal to computer path for timesharing use will provide an acceptable path for SAFT.

SAFT has been implemented in the required symmetric fashion that allows any two implementations to work together. Thus having two terminal ports for access to two different big machines will provide a method to send files between the machines with SAFT.

SAFT will detect almost all failures, partially due to the use of a checksum and partially due to the use of fixed format characters in front of each message and within the hexadecimal doublet characters. Though theoretically possible, it is extremely unlikely that bad data will be transferred undetected.

The interactive nature of the protocol to startup SAFT provides many possible parameters or features for the use of SAFT, but does not complicate the automatic part of SAFT. This greatly helps the simplicity requirements for SAFT.

1.5 ASCII and ISO character code

It should be noted that the use of the word ASCII to define characters in the SAFT message protocol, or to refer to serial communication ports, is entirely for historical reasons. Those characters used by SAFT in the basic message protocol (i.e. 0 through 9, A through F, Z and = + - ? *) are the same in ASCII or *ISO code*. *ISO code* refers to the International Reference 7-bit coded character set as defined in ISO 646-1973 (E). It is more important to refer to *ISO code* when defining character mapping in SAFT *text mode*, as described in sections 2.4.1 and 3.4.1. APPENDIX A includes the table for *ISO code*.

2. SAFT FOR THE USER

2.1 Overview of SAFT from the users point of view

2.1.1 SAFT as a program

SAFT is the name of the system as well as the definition of the actual line protocol (i.e. defined rules of communication). It is hoped that in all implementations of SAFT that the program to execute the SAFT system will be called SAFT as well. SAFT is fully intended for interactive use on interactive communication lines with computers capable of interactive processing. What this means in practice is that SAFT is an interactive program that you call into operation from your terminal. Then you will interactively specify to SAFT what operation it is to perform. Finally, SAFT will carry out the operation of transferring data by utilizing the interactive communication line between your terminal and the remote computing facility.

In actual practice there are two SAFT programs communicating with each other to effect the transfer of data between any two systems. It is the responsibility of the user to startup both of the communicating SAFT programs. One of the SAFT programs will typically be in your 'smart' (intelligent) terminal and the other in the remote computer facility you are trying to transfer files of data with. This 'inconvenience' of having to manually startup both SAFT programs is due to the use of interactive communications lines and the wide and diverse nature of all the different systems that may implement SAFT. In fact, the interactive nature of the entire SAFT system actually means that SAFT programs are operating as interactive, or timesharing programs in both machines (i.e. terminal and computer).

2.1.2 The ways SAFT may vary

As has been stated previously, SAFT is intended for use in a wide variety of systems (both hardware and software). This has created a need for great flexibility and simplicity in the SAFT system design. The result has been to place the least possible formal definitions within SAFT and to allow very flexible methods for the user interface to SAFT. This has been done by the use of interactive startup procedures.

SAFT is expected to be implemented in an interactive programming environment, in each system, to allow access to the ASCII asynchronous communications lines and to provide interactive procedures for the startup of SAFT. Thus it is

very difficult to say what SAFT will look like in every different case. One can only generalize.

Some systems may have just one communication port or line available for use by the interactive SAFT program. Other systems may provide two. This means that in the single line case human interaction will take place on the same line that is used for the SAFT formal communication with another SAFT. On systems having two ports for a SAFT program, the user dialogue with SAFT will take place on one of the lines and the formal communication on the other. It is also possible that the user will be offered the choice of one or two port mode of operation, thus allowing convenient local or remote use of SAFT.

Some versions of SAFT may wish to provide all possible SAFT services while other 'quick and dirty' implementations may only provide a single service (e.g. transfer in one mode and direction only). The quick and dirty SAFT may provide no human interaction at all, just the formal communication with another SAFT to transfer a file. While, on the other hand, the full service SAFT may have an elaborate menu dialogue with the user before SAFT transfer actually starts.

Some implementations of SAFT may wish to totally automate the entire process of selecting a remote system, invoking the remote SAFT and specifying, to the remote SAFT, the desired operation. This type of implementation will probably not be very general, thus allowing access only to specific remote systems, to allow the automatic SAFT program to know all the 'quirks and peculiarities' of the given remote system. Nonetheless, this type of SAFT, as others, may yet appear entirely different to a user even though the desired result is still a simple SAFT file transfer.

Hopefully by now the point is clear; there is little ability to specify 'the' way SAFT will appear to the user. However, typical recommended scenarios are included in the following sections that will go far in helping the user to understand the concepts of SAFT operation, and how it might appear in different systems.

2.1.3 Interactive startup of SAFT

The startup operation for SAFT, and examples of its use, are given in the following sections of this chapter. To understand these procedures it is important for the user to understand a typical operating environment. This would most typically be an interactive smart computer controlled terminal or word processor. The smart terminal usually has a terminal, meaning a keyboard with an output device like a printer or CRT, to provide interactive control of the entire smart terminal. Also, there is some local file storage such as floppy disc or cassette. Certainly, at least, the smart terminal will have a large internal memory if not external local storage. Finally, the typical smart terminal has an interactive communication line to allow it to connect elsewhere. This 'line' must be capable of asynchronous ASCII operation in a full duplex mode to be useable by SAFT.

The smart terminal user will invoke (i.e. cause to be executed) a SAFT program

on the smart terminal. The first mode of operation for the smart terminal SAFT is to pretend to be a remote 'dumb' interactive terminal to some remote computer. Thus the user may select the remote computer, login to it and invoke the SAFT program in the remote computer. At this point the smart terminal user is actually talking to the remote computer SAFT program, even though the smart terminal SAFT program is 'helping' in this by pretending to be a dumb terminal.

The user selects the desired option in the remote computer SAFT program, e.g. receive a text file, and typically specifies a file for the remote computer to use for the SAFT transfer. At this point the user tells the local smart terminal SAFT program to stop being dumb. Now the user will be interacting with the smart terminal SAFT to tell it the desired option, e.g. send a text file, and typically a file for the smart terminal to use for the SAFT transfer. From this point the smart terminal and remote computer SAFT programs will be communicating themselves, transferring the specified file of data in the selected direction. At the conclusion of the transfer, the typical remote computer SAFT will return control to its local time sharing system, and the typical smart terminal SAFT will once again pretend to be a dumb terminal to the remote computer. This will allow the user to perform what is necessary in the remote computer, such as checking the status of the file transferred, or at least logging off of the remote computer. When the user is done with his/her remote computer activities the smart terminal SAFT will be told, by the user, to return to a normal smart terminal mode.

It must be appreciated that the foregoing is only a very general description of a possible SAFT operation. Not all SAFT programs will emulate (i.e. pretend to be) dumb terminals, not all smart terminals will have a separate communication line from the local control terminal and communication will not always be between a local smart terminal and a remote large computer. Nonetheless, the conceptual operation of SAFT is always the same:

1. select and enter the first system
2. invoke the SAFT program on the first system
3. select operation in the first SAFT
4. select and enter the second system
5. invoke the SAFT program on the second system
6. select operation in the second SAFT
7. allow the transfer between the two SAFT programs to take place
8. return to normal operation in both systems

The necessarily vague parts of the above general conceptual model are related to specific hardware and system software features that differ widely between systems.

2.2 Startup menus

2.2.1 Basic menus

It is recommended to implementors that they follow the menu specifications given in this section, thus allowing the user to see as standard a SAFT user interface as possible. At this point no distinction is made between one and two port SAFT programs, or whether local or remote. How the user selects a system and invokes the SAFT program is not specified here, nor can it be, and is not important to the understanding of menus. However done, once the user has accessed the desired system with an interactive terminal and the SAFT program has been invoked, it is the intention that the user will see the following welcome message and primary menu:

```
This is system name SAFT
1.  Help
2.  Send text file from system name
3.  Receive text file into system name
4.  Other commands
5.  Exit from SAFT
Enter choice number:
```

The above menu is called the primary menu. The *system name* is a descriptive title identifying the system hardware and/or software, and possibly the given site. If "`Other commands`" is requested (i.e. "4"), the following secondary menu is given:

```
1.  Timing options
2.  Send byte file from system name
3.  Receive byte file into system name
4.  Other commands
5.  Exit from menu
Enter choice number:
```

Again, the above menu is called the secondary menu. If "`Other commands`" is requested from this menu (i.e. "4"), the first of several optional menus is given. If there is none the following message will be given:

```
No other commands
```

The optional menu format is as follows:

```
1.  optional command
2.  optional command
3.  optional command
4.  Other commands
5.  Exit from menu
Enter choice number:
```

Subsequent optional menus look the same as the one above. The same "`No other commands`" message will be given in response to "`Other commands`" requests that have no other menus following.

2.2.2 Terminal mode menu

The general description of an interactive startup of SAFT, given in section 2.1.3, describes an implementation for a smart terminal with two communication ports. One is for the local control console (terminal if you will) and the other for remote communication. In this environment it may be desirable for the SAFT program to pretend to be a normal interactive ('dumb') terminal to the remote system the user is going to invoke SAFT on. This mode is called terminal mode and may also be known as virtual or pseudo terminal mode. If terminal mode is to be supported the SAFT primary menu is expanded to provide the ability to select terminal mode whenever it may be convenient. This terminal mode form of the primary menu looks like:

```
This is system name SAFT
0.  Terminal mode
1.  Help
2.  Send text file from system name
3.  Receive text file into system name
4.  Other commands
5.  Exit from SAFT
Enter choice number:
```

If terminal mode is selected the following message will be given:

```
You are in terminal mode, enter chars to stop
```

Where *chars* is a totally implementation dependent choice of a character sequence that SAFT terminal mode will not pass on to the remote system, but will cause a return to the SAFT primary menu on the local system.

It is possible that the user's non choice of "`Terminal mode`", at startup, will cause a reversion to single port use. This is the implementor's choice, though the user must be clearly notified if this has happened.

2.2.3 Menu choices

Choices from menus are single number entries, from the menu, followed by a carriage return (or other local system line delimiter). Illegal entries should yield the response:

```
Bad choice, try again:
```

The user then simply tries again.

- "`Help`" will give simple operational information about the use of the given version of SAFT. If more information is available the help information should describe how to access it.
- "`Send...`" and "`Receive...`" choices will cause a request for filename to be output as follows:

  `Enter filename:`

  The user shall then enter a local filename, followed by a carriage return, specifying the file to SAFT send or receive operation is to take place on. There is no standard for these filenames as they are totally local system dependent.
- "`Other commands`" choices give the next menu as described in the basic menus section, 2.2.1 above.
- "`Exit from SAFT`" will cause a return to the local system that SAFT is operating under. A goodbye message will be given as follows:

  `Goodbye from` *system name* `SAFT`
- "`Exit from menu`" will return to the primary SAFT menu.
- "`Send text file from` *system name*" will cause a text file to be transmitted from the specified file on the local system. A text file is one that is in a simple line format that is described later in more detail.
- "`Receive text file into` *system name*" will cause a text file to be received into the specified file on the local system. A text file is one that is in a simple line format that is described later in more detail.
- "`Send byte file from` *system name*" will cause a byte file to be transmitted from the specified file on the local system. A byte file is one that is in a simple binary structure that is described later in more detail.
- "`Receive byte file into` *system name*" will cause a byte file to be received into the specified file on the local system. A byte file is one that is in a simple binary format that is described later in more detail.
- "`Timing options`" selects two slowdown options for specific use in ways described later in more detail.

2.3 Terminal mode

Terminal mode, as has been described earlier, simply allows the SAFT program to pretend to be an interactive terminal to the remote system. All this means is that SAFT will pass on all characters received from the local terminal's keyboard to the second communication line output port, and that all characters received on the second communication line input port will be output on the local terminal's output device (i.e. CRT or printer). This is desirable due to the human interactive startup procedures for SAFT programs. Thus, an ideal implementation of a smart terminal SAFT program will let the user switch into terminal mode to provide a method for the user to startup the remote computer SAFT program.

At the conclusion of a SAFT transfer it is also desirable to provide the user, once again, with terminal mode to allow any final interaction with the remote computer. For example a user might want to look at or use a file he/she has just transmitted to the remote computer (from the smart terminal). Also a logoff (i.e. telling the system that the user is going away) may be desired on the remote computer system. All these options are possible with the use of terminal mode and the SAFT program in the smart terminal will hopefully be able to supply this feature.

It will not always be possible to have a terminal mode for use with SAFT as not all smart terminals will have a communication line separate from that used for the local console. Also, it may be the case that the two systems needing to transfer a file between themselves are not a smart terminal and a large computer, but two large computer systems. In these cases (where there is no terminal mode available) another approach will be needed. One possible way is to utilize the single communication line in each computer system for different purposes by manually switching the equipment connected to the line.

An example of the above suggested manually switched line could be as follows:

1. user connects a regular interactive terminal to computer system 1 and proceeds to:
    a. login
    b. invoke SAFT
    c. interact with menu to specify SAFT options
2. user disconnects terminal from SAFT 1
3. user connects terminal to computer system 2 and performs steps a, b and c as in 1. above
4. user quickly interconnects SAFT 1 communication line to SAFT 2 communication line, with terminal remaining in parallel on line to observe transfer
5. user reconnects to each computer system at end of transfer to accomplish desired tasks in both systems 1 and 2 (i.e. using files, logging off, etc.)

This procedure above may be either very difficult or very simple depending on the knowledge and assistance a user has of the given systems and their communications peculiarities. Also it helps to have a manual switch box to simplify all the physical switching required.

The "`Timing options`", selectable by menu, have a role in this type of operation as an added delay is provided for the SAFT program to start the transfer, thus allowing the physical line switching to take place first.

It is planned to have a separate operational technique description to provide assistance for this mode of operation. A recommendation for the switch box would also be provided. However, this information is considered an adjunct to SAFT, not an integral part of it.

2.4 File structure

File structure for any computer system can be a very complex subject. SAFT cannot provide a general solution for transferring all possible file types between all possible computer types. Instead SAFT assumes two simple structures are able to satisfy most typical requirements. These two structures are text and byte mode.

2.4.1 Text mode

Text files are usually stored as printable lines in any given machine. There are no standards for the internal representations of the lines, or even of the character sets themselves. IBM can have blocked EBCDIC text files, CD CYBER systems use 6-bit Display Code with zero filled 60-bit words to represent lines while DEC machines use several different formats for storing both 6-bit and 7-bit forms of characters.

SAFT takes the simple view that each unique version of SAFT for a specific machine type must deal in a private, and hidden from the outside point of view, way with lines and character sets. SAFT has the concept of the 'network virtual line' with characters represented in the ISO International Reference 7-bit coded character set (*ISO code*).

The network virtual line is a line of up to any size terminated with a *CR LF* (carriage-return and line-feed). If a line is too long to be stored in one local line structure, then it is up to the receiving SAFT to break it up, though hopefully this will not be the typical case. The transmitting SAFT must take the local representation of a line and make a network virtual line, terminated by *CR LF*, of it. This includes mapping the local character set into network *ISO code*. Similarly the receiver must map the received characters to a locally accepted character standard.

Mapping of a local character set into *ISO code* should be done in accordance with ISO 646-1973 (E). National versions (e.g. Sweden's SIS 63 61 27) or application-oriented versions may be used when necessary. However, it should be noted that the success of a SAFT 'text mode' transfer will be determined by the agreements for character mapping between sender and receiver.

2.4.2 Byte mode

Byte files are stored in an unfortunately large number of ways, even on the same machine. SAFT makes no attempt to bring order to this chaos and takes the simple view that there is at least some simple format for 8-bit binary byte storage on each machine. SAFT byte transfers perform no special formatting for word values, or for sub file structures that may exist, such as records. Each implementation of SAFT must decide on the most likely to be useful local form of 8-bit binary byte storage and support that format only.

Formats beyond this must be handled by external to SAFT processes or programs that put unusual structures into the byte form supported by SAFT. The inverse of this is also true for the receiving end of SAFT.

Byte mode will be most useful when two systems of identical file structures want to exchange data. It is also possible for a 'third' system of different file structure to be utilized as a backup medium, with the transferred files structure being ignored, as the proper structure of the data will be preserved upon subsequent retrieval.

There is no internal structure for the byte data within byte binary mode messages other than the data itself. The final message of the file will be filled with a termination character, as described in the implementors chapter.

2.5 Timing options

There are two timing conventions that may be selected for use by SAFT. One is to provide a 100 milli-sec delay between receipt and start of transmission of characters. This allows very minimally equipped intelligent terminals to have time to turn around the use of the line interface internally to themselves. For example, the ABC-80 will lose characters received within 100 milli-secs of the last character the ABC-80 has output.

The second timing convention is for a 5 second delay in SAFT before it issues its first control message. This allows time for a user to quickly connect his/her intelligent terminal to the central computer port. This is useful in those cases when a SAFT implementation only has access to one ASCII communication port or line. This is typically the case with most big machines and is sometimes the case with intelligent terminals with no integral terminal. In these cases the user must connect an interactive ('dumb') terminal to the intelligent ('smart') terminal to startup SAFT, wait for its prompt to know that it is going, then disconnect the dumb terminal so it may be connected to the large machine port to startup its SAFT.

Just after the user has started up the large machine SAFT with the delayed timing options selected, he/she quickly disconnects the terminal yet again and quickly connects the smart terminal to the big machine terminal port. All this hopefully before the timer has exhausted.

This scenario may yet change as more experience is gained with use of SAFT in these type of environments.

2.6 Transfer operation

2.6.1 Transfer startup

Transfer is initiated between two SAFT systems by the selection of the ap-

propriate "`Send...`" or "`Receive...`" in each SAFT. One SAFT will always be started first but the formal SAFT protocol will cause it to wait for the startup of the other SAFT. Details of this should be noted in the sections on terminal mode and timing options above. Once initiated the transfer will go to completion, whether it succeeds or fails.

2.6.2 Display of transfer status

During the transfer the user will be given status of the transfer in one of two ways. If the local initiating SAFT has terminal mode, or at least a second communication port, SAFT will present information regarding the transfer (i.e. status) while the transfer is taking place. This information is unspecified and could be the clear text or byte information being sent or received, or it could be just an occasional message indicating the transfer was progressing. On the other hand, if there is no local console (i.e. second port with a control terminal), then all status of the transfer must be gotten from the flow of data between the two SAFT programs, i.e. the formal message communication. In this case the user's terminal may be connected to either direction of flow, it doesn't matter, as messages will be seen indicating the continuing operation of the transfer.

2.6.3 Diagnostic termination messages

As an aid to the user, SAFT shall output diagnostic messages at the successful conclusion of transfers and upon aborting in the case of failures. These diagnostic message will be output by each respective SAFT program on its communications line or on the control console of a smart terminal if one exists independently of the SAFT communications line. Typically this would be in smart terminal implementations where terminal mode was supported as well. Either way, however, the user will see a diagnostic message as it is necessary for the user to monitor the communication lines SAFT is using if a local console is not available for observing the transfer. These diagnostic messages should greatly aid the user in understanding the status of the SAFT operation.

Examples of these messages are given below, but are more fully presented in the examples of transfers section of this document. Comments in parenthesis are descriptions of the message, not part of them.

`Send done,` *n* `chars,` *m* `lines` *(text file completed by sender)*
`Send done,` *n* `bytes` *(byte file completed by sender)*
`Receive done,` *n* `chars,` *m* `lines` *(text file fully received)*
`Receive done,` *n* `bytes` *(byte file completely received)*
`Receiver aborted sender` *possible reason*
`Sender aborted by receiver` *(in response to above situation)*
`Sender aborted receiver` *possible reason*

`Receiver aborted by sender` *(in response to above situation)*

The underlying concept for the format of these messages is that it will always be clear to the user, no matter which message is seen, what has happened and who initiated the action in case of an error.

The send and receive done messages must only be issued when the entire send or receive is complete. For send, this means that the final acknowledging SAFT control message has been received. For receive, it means that the received file has been completely stored away and the final SAFT control message has been sent.

The character, and line counts for text files, and the byte count for byte files, are important information and should be given if at all possible. Line count is the number of text lines sent and received, not the number of SAFT data lines. Character and byte counts shall not include "Z" fill or *CR LF* line boundaries in the case of text files.

2.6.4 Termination of transfers

Upon completion of SAFT transfers, each SAFT shall either exit back to the host local system or to the primary menu if it supports terminal mode. In the case of a full exit from SAFT, the goodbye message will be given as described in the menu choice section, 2.2.3.

2.7 Example of SAFT usage

The following example will be useful in understanding how a user interacts with both SAFT programs to effect a transfer. The example given is for a smart terminal SAFT that supports terminal mode and a remote computer SAFT that does not. This is a quite ideal case of SAFT use in the human driven style. Manually switched cases of use will be more complex operationally, and some possible 'automata' machine driven forms of use hopefully less complex.

For the example operational notes are enclosed in parenthesis. The view is that of a user interacting with a console of a smart terminal with output from the computer in upper case and human input in lower case.

```
(user starts up the smart terminal)
WELCOME TO ST-SYS
(the smart terminal system will prompt with *)
*run program:saft
THIS IS ST-SYS SAFT
0.  TERMINAL MODE
1.  HELP
2.  SEND TEXT FILE FROM ST-SYS
3.  RECEIVE TEXT FILE INTO ST-SYS
4.  OTHER COMMANDS
```

```
5.  EXIT FROM SAFT
ENTER CHOICE NUMBER:
0 (user selects terminal mode)
YOU ARE IN TERMINAL MODE, TYPE CTRL-S TO STOP
(at this point the user may have to contend with
a terminal switch, frontend or other networking
function to reach the remote computer)
WELCOME TO DEC-10 SYSTEM
PLEASE LOGIN:username
ENTER PASSWORD:password
YOU ARE NOW LOGGED IN
(the remote computer will prompt with .)
.r saft
THIS IS DEC-10 SAFT
1.  HELP
2.  SEND TEXT FILE FROM DEC-10
3.  RECEIVE TEXT FILE INTO DEC-10
4.  OTHER COMMANDS
5.  EXIT FROM SAFT
ENTER CHOICE NUMBER:
3 (user wants to transfer a file to the DEC-10)
ENTER FILENAME:rcvfil.txt
=+ (this is part of the internal SAFT protocol)
ctrl-s (remember, this escapes from terminal mode)
THIS IS ST-SYS SAFT
0.  TERMINAL MODE
1.  HELP
2.  SEND TEXT FILE FROM ST-SYS
3.  RECEIVE TEXT FILE INTO ST-SYS
4.  OTHER COMMANDS
5.  EXIT FROM SAFT
ENTER CHOICE NUMBER:
2 (user wants to transfer a file from the st-sys)
ENTER FILENAME:file:text-data
TRANSFER STARTING
SEND DONE, 5280 CHARS, 1000 LINES
THIS IS ST-SYS SAFT
0.  TERMINAL MODE
1.  HELP
2.  SEND TEXT FILE FROM ST-SYS
3.  RECEIVE TEXT FILE INTO ST-SYS
4.  OTHER COMMANDS
5.  EXIT FROM SAFT
ENTER CHOICE NUMBER:
```

```
0 (user selects terminal mode again)
(user now wants to look at file on DEC-10)
YOU ARE IN TERMINAL MODE, TYPE CTRL-S TO STOP
ctrl-c ctrl-c (user is just seeing if DEC-10 is there)
..sos rcvfil.txt
EDIT RCVFIL.TXT
(user looks at file received)
eq (user leaves editor)
.kjob (user logs off DEC-10)
YOU ARE LOGGED OFF DEC-10
ctrl-s (user leave terminal mode for the last time)
THIS IS ST-SYS SAFT
0.  TERMINAL MODE
1.  HELP
2.  SEND TEXT FILE FROM ST-SYS
3.  RECEIVE TEXT FILE INTO ST-SYS
4.  OTHER COMMANDS
5.  EXIT FROM SAFT
ENTER CHOICE NUMBER:
5 (user chooses to exit saft)
GOODBYE FROM ST-SYS-X SAFT
* (now user is back in original state in smart terminal)
```

3. SAFT FOR THE IMPLEMENTOR

3.1 Overview of SAFT from the implementors point of view

This chapter specifies those parts of SAFT which are mandatory for an implementation to be considered a SAFT system. But first, several of the points from chapter two are more fully explained before the details of the formal protocol are presented. As mentioned in the chapter on SAFT for the user, SAFT is the name of a system, which includes a formal protocol and a user interface. The program written to implement SAFT shall also be known as SAFT. The user interface of SAFT is entirely system dependent, taking into account the goals of the specific design.

3.1.1 Ways to implement SAFT

There are several ways to approach an implementation of SAFT. All of them require the use of the SAFT formal protocol described in this chapter. However, there are different approaches possible based on the goals desired. One approach is the 'quick and dirty' one, to simply get a file of one type transferred in just one direction. For example, to receive a text file. At the other extreme is an automata SAFT that can select a remote system, login to it, invoke a remote copy of SAFT in that system and interactively converse with it all automatically. And, of course, all possible options of SAFT would be available. Between these two rather extreme examples, both of which are possible, is the normal case. There are two typical examples of this normal case. The first is a central multi-user shared use computer system with only one communication port for the use of its SAFT implementation. The other is a typical single user smart (e.g. microprocessor controlled) terminal which has two communication ports for the use of the SAFT program.

The typical multi-user system (call it a remote computer) will implement the typical basic set of SAFT services, i.e. send and receive in text and byte mode, and the timing options. This type of SAFT will typically be the remote end of a conversation with a smart terminal system where the user is. All of the SAFT communications, both formal protocol and user interface, are carried out on one communication port. Thus the user will be interacting with the remote computer in a fairly normal way while logging in, invoking SAFT and interactively specifying the desired SAFT options. Then, for the transfer, the user relinquishes the communication line to the use of SAFT.

The typical smart terminal environment will implement the basic set of SAFT services as does the typical remote computer, i.e. send and receive text and byte

files, and the timing options. This type of SAFT will typically be the local end of a conversation with a remote computer system. The user will be at the local smart terminal. The user will interact with the smart terminal via a console that in itself is very much like a typical interactive terminal, i.e. it has a keyboard and an output device (CRT or printer). The interactive setup of the remote computer SAFT will take place through this console and the local interactive setup of the local SAFT as well. For the actual SAFT file transfer the smart terminal will use another communication line than the console is attached to.

In describing the two different, and typical, environments above it is not meant to imply they are all inclusive of possible uses that SAFT will be put to. Each implementor will have to decide the intended environment for the desired SAFT.

One example, of deciding the intended use environment, is where two modes of SAFT operation are desired in a computer system. One for local use with two terminal ports, and the other for remote use with only one terminal port (i. e. the menu interaction and file transmission occur on the same terminal port). An implementor could choose to provide these two modes of operation by the use of either an *optional command*, or the user's implicit refusal of "Terminal mode" at startup, to allow single port use instead of two port use.

3.1.2 Use of menu format

Though the menu formats described in the user chapter are not at all a mandatory part of the SAFT specification, it is the hope that if the use of an interactive setup is needed that the menus as described will be used. This could greatly simplify the life of a user.

The implementor should refer to the user chapter for details about the menus. Note that if menus are used for a single port SAFT design that the menu is used over the single port. If a two port SAFT is implemented, then the menus are used over the console port, not the communication port. As *optional command* could be used to allow the selection of one or two port mode of use.

3.1.3 Use of diagnostic termination messages

The diagnostic termination messages are to be used in the same fashion as the menus, optionally, but to be used if at all possible. Again, refer to the user chapter for specification of these messages.

3.1.4 Display of transfer status

There is no recommendation for the specific format of displaying transfer status while the transfer is in progress. Of course, the single port SAFT implementation need do nothing for this as the protocol messages themselves must suffice for

informing the user of the progress of the transfer. In two port SAFT implementations, it is desirable to keep the user informed by the use of some display on the console.

3.2 Design philosophy

3.2.1 Transparency

SAFT is intended to operate transparently across any or all communication paths typically used for terminal access to a computer system. Thus a smart or intelligent terminal, such as a microprocessor based word processing system, could transfer files with a traditional multi-user timesharing machine. As SAFT is also symmetric two large machines can also use it to transfer files between themselves. More important here, however, is the transparency. All messages are encoded with a minimal set of ASCII characters.

3.2.2 Symmetry

SAFT is a symmetric protocol to allow any two implementations to transfer files between themselves. The only real requirement that this imposes on the implementation is the startup. First of all most protocols that are not symmetric come about due to the difference between who gives the commands and who responds to them. In the case of SAFT this is not a problem due to the requirement for simplicity which led to all commands and responses for initiating the transfer being done by the human user of SAFT.

In fact, typical implementations of SAFT take commands from the human user as interactive setup of the SAFT transfer before the actual SAFT protocol for control and data message passing starts. This is possible as all SAFT communication must be done on an ASCII terminal line and thus this line can be used for the interactive setup as well. This becomes a great advantage in keeping the protocol simple. Thus, since all parameter control is now separate from the SAFT protocol itself, symmetry is more easily achieved.

The other problem in attaining symmetry is the issue of which SAFT program starts up first, and thus issues an "=+" prompt which will not be seen yet by the other SAFT as it has yet to be started by the user. Thus all SAFT startups, whether for sending or receiving, will issue the "=+" prompt, and all SAFT programs receiving a prompt will either start sending data or send another prompt. Thus symmetry is achieved, either SAFT may be started up first and the process will work.

3.3 Message structure

3.3.1 Basic format

When two SAFT programs are in communication with each other, one SAFT will send a protocol message causing the other SAFT to respond with an appropriate protocol message. During this part of the SAFT transfer any echoing of characters must be suppressed in both direction. See the section on echo suppress later in this chapter.

All SAFT protocol messages begin with an equal sign ("=") character. All messages end with a carriage return. All characters encountered (i.e. received) between the carriage return and equal sign are totally ignored by SAFT. These characters may be discarded by the receiving SAFT with no affect on the SAFT operation in progress. There are only two basic types of messages, control and data.

3.3.2 Control messages

There are four (4) different control messages and they are created by one character following the equal sign header character.

=+ prompt at startup, no-op at any time, positive acknowledgement to data messages

=- negative acknowledgement to data messages, abort message

=? request for retransmission of last data message

=/ end of data file transfer

Any character following the legal control characters =+ =- =? =/ is discarded. This allows languages like FORTRAN to input fixed sized messages without concern for how long the message really is.

The use of the control messages is defined by the following table

DEFINITION OF CONTROL MESSAGES		
Command	**Received by sender**	**Received by receiver**
=+	start sending data, send next data	send an =+
=-	abort	abort
=?	retransmit last data message	*not defined*
=/	*not defined*	end of data file

The '*not defined*' messages may cause aborts if received.

3.3.3 Data messages

Data messages are fortyfour (44) characters in length.

=hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhcc

Any character after the 44th character in the message is discarded. The *h* field is forty (40) characters long and represents twenty (20) 8-bit bytes of information, whether character or binary. The encoding is in hexadecimal doublets with two *h* characters representing an 8-bit value. Each *h* character can have the value:

0 through 9 and A through F or Z

Note that the "A" through "F" and "Z" are upper case only. These values for *h* allow it to represent a 4-bit hexadecimal value. The "Z" is to provide fill when terminating the file transfer, as all data messages are always 44 characters (excluding the *CR*) long.

The *cc* field is two (2) characters long and represents a single 8-bit checksum value encoded in the same hexadecimal doublet format that the data is formatted in.

3.3.4 Checksum

The checksum (*cc*) is only present, and thus used for, data messages. When the checksum is computed it is the sum of the 20 8-bit values comprising the data field, before conversion to hexadecimal doublets, modulo 256.

Note that messages to be filled with "ZZ" characters should either be uncounted in the checksum, or added as binary zero values.

As an example of the format described above, the following string:

tttttttttttttttttttt (20 lower case t-s in a row)

will be formatted into the 44 character data message string:

=*74747474747474747474747474747474747474747410*

The ASCII value of "t" is 74 hex, or 116 decimal. The last two hex characters, the checksum, are 10 hex as the sum of 20 8-bit values of 74 hex is 910 hex, the rightmost 8 bits of which are 10 hex. To lead into the rest of the SAFT description, note that the proper receival of this message will result in the receiver sending an "=+" control message back to the sender.

3.3.5 Data message retransmission

Simple error recovery is done in SAFT by the use of the "=?" control message. The "=?" response is only for use in errors detected in the data messages by the receiving end of a SAFT file transfer. Checksum errors are cause for a request to

the sender (i.e. "=?") to retransmit the last data message. Errors in the character format (i.e. hex doublet) are also cause for retransmission.

There is no requirement for SAFT file receivers to keep a count of the number of times a given message is retransmitted (e.g. to eventually abort on too many retransmits). Thus it is possible that infinite loops can occur if a SAFT transmitter is generating a bad checksum or the line is continually causing data message faults. This is not a practical consideration, however, as any line that is faulting so badly that this might occur is even more likely to cause an error in a header (i.e. "=") or control message, which will entirely stop data flow. These cases are left for detection by the user on the monitoring terminal.

3.3.6 Echo suppression during the formal protocol

During the formal protocol phase of the SAFT transfer, all echoing must be suppressed in the communications interfacing system for each SAFT. This should be done (i.e. the echo suppress) automatically by the SAFT program, if at all possible. If it is not able to be done automatically the user must be informed how to accomplish this by the documentation of the SAFT implementation.

At the conclusion of the formal protocol (i.e. the actual file transfer) the SAFT program must enable the echoing again to provide for the subsequent return to interactive mode by the user.

3.4 File structure

3.4.1 Text mode

Text files, on any given system that has a SAFT implementation, must have a well defined mapping between the local system character set and the SAFT character set (i.e. *ISO code*, see sections 1.5 and 2.4.1). In particular, all local system characters must have a mapping into some character sequence in the SAFT character set. Thus it is illegal for any SAFT to generate a text mode message with a character greater than 127 decimal. The presence of an illegal text character in a received data message will cause an abort (i.e. "=-") by the SAFT file receiver. This means that an abort will occur for characters greater than 127, although a correct text format and a correct checksum was received. This is required as the most likely reason for such an error is that text mode was chosen in the receiving SAFT when byte mode was chosen in the sending SAFT. A reproduction of the ISO International Reference 7-bit coded character set is given in appendix A.

There must also be a well defined mapping of the local file line structure into SAFT text mode line structure (i.e. *CR LF*). If there is more than one line structure convention in a given system, it is the responsibility of either the local file

system, or the user's file specification to SAFT, to tell SAFT which file structure is in use. However it is done, the result must be that text mode files clearly delineate line structure by the presence of the *CR LF*. If the file needs to represent a *CR* followed by a *LF*, then the sending SAFT shall format a *CR NUL LF* to avoid the misinterpretation of it as a line boundary.

3.4.2 Byte mode

Byte files are intended to be simple octet (i.e. 8-bit) binary files of data with no structure implied other than that of a serial stream of 8-bit values.

The SAFT implementation of byte mode transfer must have an appropriate mapping of the machine binary word and file structure with the SAFT byte binary transfer structure. This means two different things. One, that a meaningful mapping must be chosen, and two, that a file sent via SAFT and then received back, via SAFT, will result in the same local file.

A meaningful mapping is one that provides the most useful file from the user point of view. For example, a given machine type may have several ways of storing binary information. One way might be words of the local system, another might be 8-bit value storage. This is most critical to distinguish on machines with word sizes that are not a multiple of 8-bits.

Reproducible files, the second point mentioned above, is quite important as it prevents a file from being transmitted via SAFT, and later retrieved in a different form or value. This is not always an easy thing to accomplish, as machines with word sizes that are not multiples of 8-bits may not be able to resolve word values in the SAFT byte format.

An issue related to reproducible files is that of word versus byte reproducibility. This means that a design choice must be made about whether to transmit the local 'convenient' form of byte storage or whether to transmit all binary information within a word. An example of this would be on a 36 bit word size machine where the choice is between sending 9 bytes constructed from two consecutive 36 bit words and sending 4 bytes per 36 bit word. The latter is only relevant if there is a convention that stores 8 bit values in 9 bit quarter word values, with the ninth bit being zero or unused. There is no absolute recommendation made here, though it must be noted that the full word transmission method is most flexible, though possibly less efficient at times.

3.5 Examples of protocol

3.5.1 Message flow example

What follows is an example of the message flow between two copies of SAFT.

=+CR ⇒

⇐ =+CR

=*hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhcc CR ⇒

⇐ =+CR

=*hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhcc CR ⇒

⇐ =+CR

=*hhhhhhhhhhhhhhhhZZZZZZZZZZZZZZZZZZZZZZZZcc CR ⇒

⇐ =+CR

=/CR ⇒

⇐ =+CR

Note the use of the capital "Z" to denote the unused portion of the last messag in the transfer. Also note that the transmitting side issued a prompt when it wa started up. As it was started first, it is likely that the receiving side never sav it. It was the prompt sent by the receiver, when it started up, that triggered th transmitter to start sending data messages.

There were no errors in this transmission. It ended normally with the sende (transmitter) sending an "=/" and the receiver sending a prompt (i.e. "=+") bac to acknowledge it. At that point the user can safely usurp both machine processe as the "=+" from the receiver denotes full receipt, and safe storage, of the file.

3.5.2 Examples of transfers

▶ Sender starts first

=+CR ⇒

⇐ =+CR

=*hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhcc CR ⇒

⇐ =+CR

=*hhhhhhhhhhhhhhhhZZZZZZZZZZZZZZZZZZZZZZZZcc CR ⇒

⇐ =+CR

=/CR ⇒

⇐ =+CR

`send done, n chars, m lines ⇒`

`⇐ receive done, n chars, m lines`

The completion messages shall be output at the completion of the successfu transfer. In the case of the sender the message may be output upon receiving th "=+" for the "=/" . However, the receiver shall not send the "=+", in respons to the "=/" , until all the received data has been properly disposed of. Thus th sender may be totally assured of proper receipt of the message sent.

▶ Receiver starts first

⇐ =+*CR*

=+*CR* ⇒

⇐ =+*CR*

=*hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhcc*CR* ⇒

⇐ =+*CR*

=*hhhhhhhhhhhhhhhhZZZZZZZZZZZZZZZZZZZZZZZZZcc*CR* ⇒

⇐ =+*CR*

=/*CR* ⇒

⇐ =+*CR*

`send done, n chars, m lines` ⇒

⇐ `receive done, n chars, m lines`

▸ Receiver aborts transfer

Ignoring who starts first, this example presumes an error is detected by the receiver. This could be for a checksum or control character failure.

=*hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhcc*CR* ⇒

⇐ =+*CR*

=*hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhcc*CR* ⇒

⇐ =-*CR*

⇐ `receiver aborted sender` *possible reason*

`sender aborted by receiver` ⇒

In the example above it is the responsibility of both the aborting processes (copies of SAFT) to output error messages. The side aborting the process must try to specify the type of failure (*possible reason*) while the side that has been aborted need only state the fact that it has happened.

▸ Sender aborts transfer

As with the previous example, it does not matter who starts first. It is presumed that the sender detects a failure, for example a bad response character after the "=" has been received.

=*hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhcc*CR* ⇒

⇐ =+*CR*

=*hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhcc*CR* ⇒

⇐ =+*CR*

=-*CR* ⇒

`sender aborted receiver` *possible reason* ⇒

⇐ `receiver aborted by sender`

As in the previous example the side receiving the abort need not respond other that the mandatory error message, which the aborting side must output as well, but with a possible reason.

APPENDIX A — ISO International Reference 7-bit coded character set

This character code set is from the International Reference Version 7-bit coded charac ter set, ISO 646-1973 (E), section 6 (Table 2). We cite the paragraph 6.2 Internationa reference version in that document:

"This version is available for use when there is no requirement to use a national or an application oriented version. In international information processing interchange the international referenc version (Table 2) is assumed unless a particular agreement exists between sender and recipient c data."

	0	1	2	3	4	5	6	7
000_8	*NUL*	*SOH*	*STX*	*ETX*	*EOT*	*ENQ*	*ACK*	*BEL*
010_8	*BS*	*HT*	*LF*	*VT*	*FF*	*CR*	*SO*	*SI*
020_8	*DLE*	DC_1	DC_2	DC_3	DC_4	*NAK*	*SYN*	*ETB*
030_8	*CAN*	*EM*	*SUB*	*ESC*	*FS*	*GS*	*RS*	*US*
040_8	*SP*	!	"	#	$	%	&	'
050_8	(	)	*	+	,	-	.	/
060_8	0	1	2	3	4	5	6	7
070_8	8	9	:	;	<	=	>	?
100_8	@	A	B	C	D	E	F	G
110_8	H	I	J	K	L	M	N	O
120_8	P	Q	R	S	T	U	V	W
130_8	X	Y	Z	[	\	]	^	_
140_8	`	a	b	c	d	e	f	g
150_8	h	i	j	k	l	m	n	o
160_8	p	q	r	s	t	u	v	w
170_8	x	y	z	{	|	}	~	*DEL*

This is the Swedish version of ISO 646 7-bit coded character set intended for general use. It is called **SIS 63 61 27**.

	0	1	2	3	4	5	6	7
000_8	*NUL*	*SOH*	*STX*	*ETX*	*EOT*	*ENQ*	*ACK*	*BEL*
010_8	*BS*	*HT*	*LF*	*VT*	*FF*	*CR*	*SO*	*SI*
020_8	*DLE*	DC_1	DC_2	DC_3	DC_4	*NAK*	*SYN*	*ETB*
030_8	*CAN*	*EM*	*SUB*	*ESC*	*FS*	*GS*	*RS*	*US*
040_8	*SP*	!	"	#	¤	%	&	´
050_8	(	)	*	+	,	-	.	/
060_8	0	1	2	3	4	5	6	7
070_8	8	9	:	;	<	=	>	?
100_8	@	A	B	C	D	E	F	G
110_8	H	I	J	K	L	M	N	O
120_8	P	Q	R	S	T	U	V	W
130_8	X	Y	Z	Ä	Ö	Å	^	_
140_8	`	a	b	c	d	e	f	g
150_8	h	i	j	k	l	m	n	o
160_8	p	q	r	s	t	u	v	w
170_8	x	y	z	ä	ö	å	~	*DEL*

050	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
060	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
070	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Messkorsband

Tjänste
Stockholms Datorcentral
Fack
104 50 Stockholm