INDRA Note 1185

INDRA

Feb. 1982

Working Paper

RFC 809

UCL FACSIMILE SYSTEM

Tawei Chang

ABSTRACT: This note describes the features of the computerised facsimile system developed in the Department of Computer Science at UCL. First its functions are considered and the related experimental work are reported. Then the disciplines for system design are discussed. Finally, the implementation of the system are described, while detailed description are given as appendices.

Department of Computer Science

University College, London

NOTE: Figures 5 and 6 may be obtained by sending a request to Ann Westine at USC-Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, California, 90291 (or WESTINE@ISIF) including your name and postal mailing address. Please mention that you are requesting figures 5 and 6 from RFC 809.

OR: You can obtain these two figures online from the files

<NETINFO>RFC809a.FAX and <NETINFO>RFC809b.FAX

from the SRI-NIC online library. These files are in the format described in RFC 769.

Contents

1. INTRODUCTION

The object of a facsimile system is to reproduce faithfully a document or image from one piece of paper onto another piece of paper sited remotely from the first one. Up to now, the main method of facsimile communication has been via the telephone network. Most facsimile machines permit neither the storage of image page nor their modification before transmission. With such machines, it is almost impossible to communicate between different makes of facsimile machines. In this respect, facsimile machines fall behind other electronic communication services.

Integration of a facsimile service with computer communication techniques can bring great improvements in service. Not only is the reliability and efficiency improved but, more important, the system can be integrated with other forms of data communication. Moreover, the computer enables the facsimile machine to fit into a complete message and information processing environment. The storage facilities provided by the computer system make it possible to store large amounts of facsimile data and retrieve them rapidly. Data conversion allows facsimile machines of different types to communicate with each other. Furthermore, the facsimile image is edited and/or combined with other forms of data, such as text, voice and graphics, to construct a multi-media message, which can be widely distributed over computer networks.

In the Department of Computer Science at UCL, a computerised facsimile system has been developed in order to fully apply computer technology, especially communication, to the facsimile field. Some work has been done to improve the facsimile service in several areas.

(1) Adaptation of the facsimile machine for use with computer networks. This permits more reliable and accurate document transmission, as well as improving the normal point-to-point transfers.

(2) Storage of facsimile pages. This permits the queueing of pages, so saving operator time. Also, standard documents can be kept permanently and transmitted at any time.

(3) Interworking with other facsimile machines. This permits different makes of facsimile machines to

exchange images.

(4) Compression of the facsimile images. This allows more efficient transmission to be achieved. Different compression schemes are investigated.

(5) Display of images on other devices. A colour display is used so that the result of image processing can be shown very vividly.

(6) Improvement of the images. The ability to 'clean' the facsimile images not only allows for even higher compression ratio, but also provide a better result at the destination.

(7) Editing of facsimile pages. This includes the ability to change pictures, alter the size of images and merge two or more images, all electronically.

(8) Integration of the facsimile service with other data types. For the time being, coded character text can be converted into facsimile format and mixed pages containing pictures and text can be manipulated.

This note first considers the functions of the facsimile system, the related experimental work being reported. Then the discipline for the system design is discussed. Finally, the implementation of the UCL facsimile system is described. As appendices, detailed description of the system are given, namely

I. Devices
II. Task controller and task processes
III. Utility routines and Data format

2. SYSTEM FUNCTIONS

The computerised facsimile system we have developed is composed of an LSI-11 micro-computer running the MOS operating system [14] with two AED62 floppy disk drives [17], a Grinnell colour display [18], a DACOM facsimile machine [16], and a VDU as the system console. This LSI-11 is also attached to several networks, including the ARPANET/SATNET [21], [22] and the UCL Cambridge Ring. A schematic of the system is shown in Fig. 1.

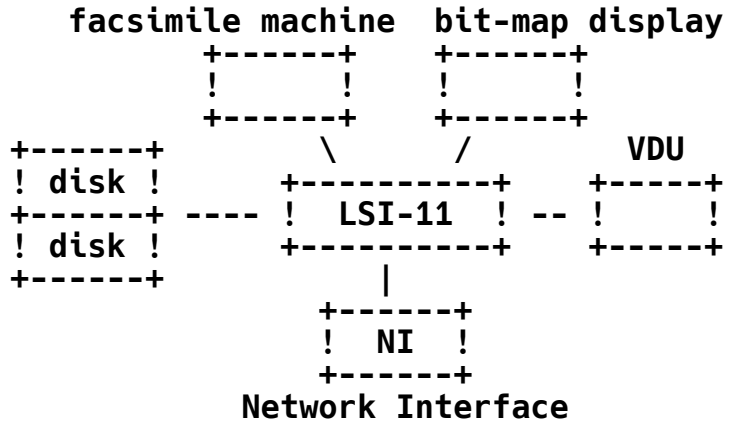


Fig. 1 Schematic of UCL facsimile system

In this system, a page is read on the facsimile machine and the image data produced is stored on the floppy disk. This data can be processed locally in the micro-computer and then sent to a file store of a remote computer across the computer network. At the remote site, the image data may be processed and printed on a facsimile machine.

On the other hand, we can receive image data which is sent by a remote host on the network. This data can be manipulated in the same way, including being printed on the local machine.

Section 2.1 dicusses the problems concerned with transmission of facsimile image data over a network, while the following sections deal with those of local manipulation of image data.

In order to interwork with other facsimile machine, we have to convert the image data from one representation format to another. Interworking with other output devices requires that the image be scaled to fit the dimension of the destination device. These are described in section 2.2.

Being able to process the image by computer opens the door to many possibilities. First, as considered in section 2.3, an image can be enhanced, so that the quality of the image may be improved and more efficient storage and transmission can be achieved. Secondly, a facsimile editing system can be supported whereby a picture can be changed and/or combined with other

pictures. This is described in section 2.4.

In our system, coded character text can be converted into its bit-map representation format so that it can be handled as a facsimile image and merged with pictures. This provides an environment where multi-type information can be dealt with. This is discussed in section 2.5.

2.1 Communication

The first goal of our computerised facsimile system is to use a computer network to transmit data between facsimile machines which are geographically separated.

Normally, facsimile machines are used in association with telephone equipment, the data being sent along telephone lines. Placing the facsimile machines on a computer network presents a problem as the facsimile machine does not have the ability to use a computer network directly. To perform the network tasks a computer is required, and so the first phase was to attach the facsimile machine to a computer.

The facsimile machine is not like a standard piece of computer equipment. We required a special hardware interface to enable communication between the facsimile machine and a small computer. This interface was made to appear exactly like the telephone system to the facsimile machine. Furthermore, the computer was programmed to act exactly as if it were another facsimile machine on the end of a telephone line. Thus the local facsimile machine could transmit data to the computer quite happily, believing that it was actually talking to a remote facsimile machine on the other end of a telephone wire. Because of the property of the DACOM 6450 used in the experiment [16], the interface could be identical to one developed for connecting to an X25 network. The binary synchronous mode of the chip used (SMC COM5025) was appropriate to drive the DACOM machine.

At the other side of the computer network there was a similar computer with an identical facsimile machine. The problem of transmitting a facsimile picture now appeared simple: data was taken from the facsimile machine into the computer, transmitted over the network as if it was normal computer data, and then sent from the computer to the facsimile machine at the remote end. The data being sent over the network appears

exactly as any other computer data; there is nothing special about it to signify that it came from a facsimile machine. The schematic of such facsimile transfer system is shown in Fig. 2.

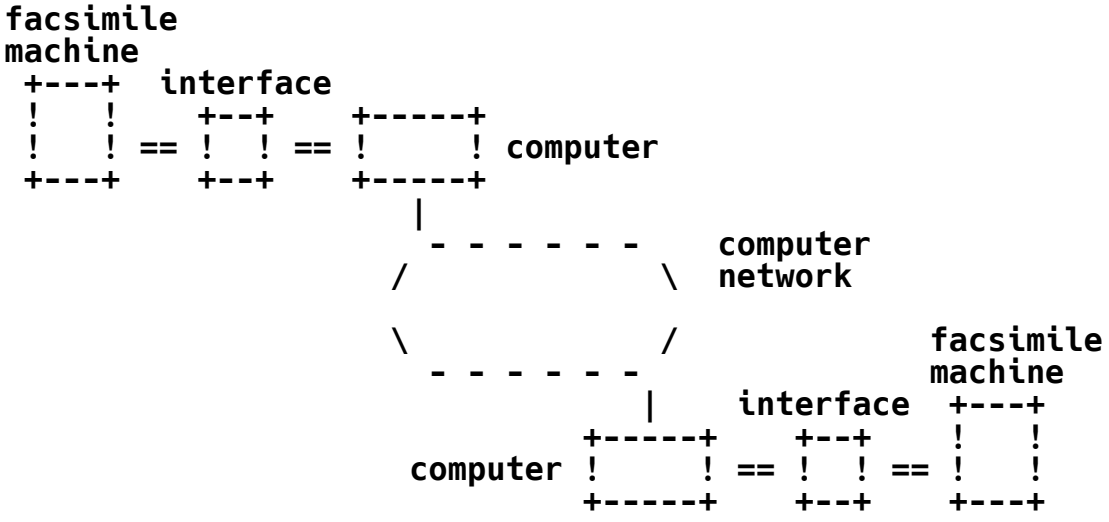


Fig. 2 Facsimile transfer system

The experimental system was used to perform a joint experiment between UCL and two groups in the United States. Pictures were exchanged via the ARPANET/SATNET [21], [22] between UCL in London, ISI in Los Angeles, and COMSAT in Washington D.C. (Fig. 3). This environment was chosen because no equivalent group was available in the UK.

One problem concerned with such image data transmission is the quantity of data. Even with data compression, a single page of facsimile data can produce as much computer data as would normally be sufficient for sending over 20,000 alphabetic characters - or over a dozen typed pages. Thus for a given number of pages put into the system, an immense amount of computer data is produced. This means that the transmission will be slower than for sending text, and that far more storage will be required to hold the data.

Another problem was encountered which became only too apparent when we implemented this system. The network we were using was often unable to keep up with the speed of the facsimile machine. When this happened the

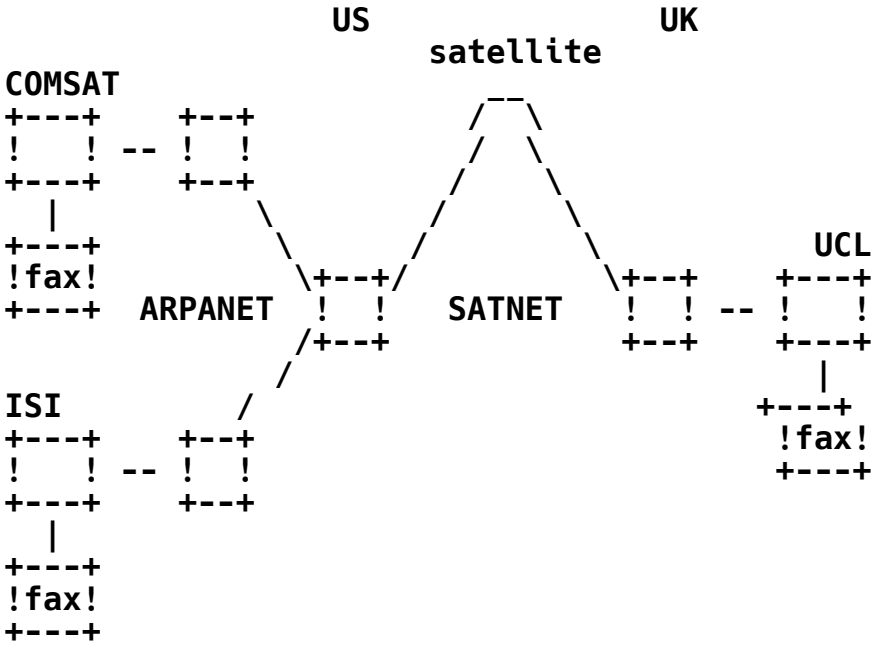


Fig. 3. The three participants of the facsimile experiments

computer tried to slow down the facsimile machine. The facsimile machine would detect this 'slowness' as a communication problem (as a telephone line would never act in this manner), and would abandon the transfer mid-way through the page.

This is because the the facsimile machine we were using was never intended for use on a computer; it was designed and built for use on telephone lines. Indeed, being unaware that it was connected to a computer, the facsimile machine transmitted data at a constant rate, which exceeded the limit that the network could accept. In other words, the computer network we were using was not designed for the transfer rate that we were trying to use over it.

Both these problems are surmountable. Facsimile machines are coming on the market that are designed for direct communication with a computer. These machines do not mind the delays on the computer interface and are tolerant of the stops and re-starts. On the other hand, if there were a serious use of facsimile machines on a computer network, the network could be designed for the high data rate required. Our problem was aggravated by

using a network that was never designed for the data rates required in our mode of usage.

Despite the problems we encountered being a result of the experimental equipment we were working with, we still had to improve the situation to permit more extensive communications to take place. The easiest way to do this was to introduce a local storage area in our computer where the data could be held prior to transmission. The transfer of a page is now done in three stages. First, the facsimile data is read from the facsimile machine and stored on a local disk. This takes place at high speed as this is just a local operation. When this is complete, the data is sent over the network to a disk on the remote computer. Finally, the data from that disk is output to the remote facsimile machine. This improved system is shown in Fig. 4.

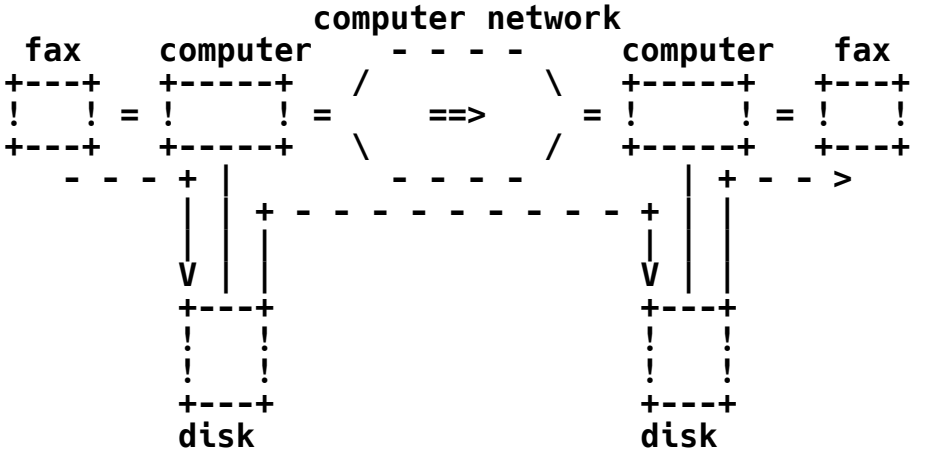


Fig. 4. The improved facsimile transfer system

The idea behind this method is to decouple the facsimile machine from the network communications. The data is read from the facsimile machine at full speed, without the delays caused by the computer network. This also has the effect of being more acceptable to the human operators: each page is now read in less than a minute. The transmission over the network then takes place at whatever speed the network can sustain. This does not affect the facsimile machines at all; they are not involved in the sending or receiving. Only when all the data has been received at the remote disk is the remote facsimile machine told that the data is ready.

The facsimile machine is then given the data as fast as it will accept it.

The disadvantage of such a system is that the person sending the pages does not know how long it will be before they are actually printed at the other side. If several pages are input in quick succession by the operator, they will be stored on disk; it may then be some time before the last page is actually delivered to the destination. This is not always a disadvantage; where many operators are sending data to the same destination, it is a definite advantage to be able to input the pages and have the system deliver them when the destination becomes free. Such a system is preferable to use of the current telephone system where the operator has to keep re-dialing the remote facsimile machine until the call is answered.

2.2 Interworking with Other Equipment

2.2.1 Facsimile machines

As was mentioned earlier, facsimile machines produce a large amount of data per page due to the way in which the pages are encoded. To reduce the data that has to be transmitted, various compression techniques are employed. The manufacturers of facsimile machines have developed proprietary ways in which the data is compressed and encoded. Unfortunately this has meant that interworking of different facsimile machines has been impossible. In the system described in the last section, exchange of pictures was only possible between sites that had identical facsimile machines. The new set of CCITT recommendations will reduce the extent to which differences in equipment persist.

Having the data on a computer gives us the opportunity to manipulate data in any way we wish. In particular we could convert the data from the form used in one facsimile machine to that required by another. This means that interworking between different types of facsimile machines can be achieved.

The development of this system took place in two stages: the decompression of the facsimile data from the coded form used in our machine into an internal data form and the recompression of the data in the internal form into the encoded form required for the destination machine. Two programs were developed to perform these two operations.

At the same time we were developing compression and decompression programs for machines that use other techniques. In particular, we developed programs to handle the recently approved CCITT recommendation for facsimile compression [15]. The CCITT came up with two varieties of compression, depending upon the resolution being used.

Unfortunately there were no facsimile machines on the network that use the CCITT compression technique. However, the programming of the new methods achieved two goals: it proved that the data could be converted inside a small computer, so that machines of different types could be supported on the network, and it enabled us to compare the compression results. These are described in more detail in [13]. Essentially, these show that the DACOM technique used by our facsimile machine is comparatively poor, and that considerably less data need be transmitted if some other method is used. This brings up another possibility: we could change the compression of the data to reduce the volume for transmission and then change the data back again at the destination. This may save considerable transmission time, especially if fast computers or special hardware was easily available. This has not been tried yet in our system, as none of the other users on the network have the capability of changing the data format back into that required by their machines.

There are many other more efficient compression schemes, e.g. block compression [7] and predictive compression [8], but we have not yet incorporated them into our system.

2.2.2 Output Devices

One area that we have explored is the use of devices other than facsimile machines for outputting the data. Facsimile machines are both expensive to buy and relatively slow to operate. We have investigated the use of a TV-like screen to display the data, just as character VDUs are commonly used to display text. This activity requires bit-map displays, with an address in memory for each postion on the screen. Full colour and multiple shades can be used with appropriately large bit-map storage. Although simple in principle, the implementation of the relevant techniques took considerable effort.

The problems arise in the way that the facsimile image is encoded. Raw facsimile images consist of rows of small dots, each dot recorded as a black or white space. When these dots are arranged together they build up a picture in a similar manner to the way in which a newspaper picture is made up. Unfortunately the number of dots used in a facsimile page is not the same as the number used on most screens. For instance, the DACOM facsimile machine uses 1726 dots across each page, but across a screen there are usually just 512 dots. Thus to show the picture on the screen the 1726 dots must be 'squeezed' into just 512 dots; stated another way, 1214 dots must be thrown away without losing the picture!

It is in reducing the number of picture elements that the problem arises. We could just every third dot or so from the facsimile page and just display those. Alternatively, we could take three or more at a time and try to convert the group of them into a single black or white dot. Unfortunately, in both these cases, data can get lost that is necessary to the picture. For instance, a facsimile encoding of an architect drawing could easily end up with a complete line removed, radically changing the presentation of the image.

After much experimentation, we developed a method of reducing the number of dots without destroying the picture. This is a thinning technique, whereby key elements of the picture are thinned, but not removed. Occasionally, when the detail gets too fine, some elements are merged, but under these circumstances the eye would not have been able to see the detail anyway. The details of this technique are described in [3] and [4].

It may also be required that a picture be enlarged. This enlargement can be done by simply duplicating each pixel in the picture. For a non-integral ratio, the picture can be expanded up to the nearest integer and then shrunk to the correct size. However, this method may degrade the image quality, e.g. the oblique contour may become stepped, especially when the picture is enlarged too much. This problem can be solved by using an iterative enlargement algorithm. Each time a pixel is replaced with a 2x2 array of pixels, whose pattern depends on the original pixel and the pixels surrounding it. This procedure is repeated until the requested ratio is reached. If the ration is not a power of 2's, the same method as that for non-integral ratios is used.

As a side effect of developing this technique, we could freely change the size and shape of an image. The picture can be expanded or shrunk, or it can be distorted. Distortion, whereby the horizontal and vertical dimensions of the image may be changed by different amounts, is often useful in image editing.

The immediate consequence of this ability to change the image size meant that we could display the image on a screen as well as output the image on a facsimile machine. To a user of a computerised facsimile system this could be a very useful feature: images can be displayed on screen much faster than on a facsimile machine, and displays are significantly cheaper than the facsimile machines as well. It is possible that an installation could have many screen displays where the image could be viewed, but perhaps only one facsimile machine would be available for hard copy. This would be similar to many computer configurations today where the number of printers is limited due to their cost, and display screens are far more numerous.

2.3 Image Enhancement

One aspect of computer processing that we wanted to investigate was that of image enhancement. Enhancing the image is a very tricky operation; as the name implies it means that the image is improved in some sense. Under program control this is difficult to achieve: what the program thinks is an improvement, the human might judge to be distinctly worse.

Our enhancement attempts were aimed particularly at printed documents and other forms of typed text. The experiment was double pronged: we hoped to make the image easier to read by humans while also making the image easier for the computer to handle.

In our earlier experiments we had noticed that the encoding of printed matter was often very poor. This was especially noticeable when we enlarged an image. Rather than each character having smooth edges as on the original document, the edges were very rough, unexpected notches and excrescences being caused by the facsimile scanner. They not only degrade the image quality but also decrease the compression efficiency. A typical enlargement of several characters is shown in Fig. 5.

Fig 5. An enlargement of an typed text

The enhancement method we adopted was first employed at Loughborough University [5]. This method has the effect of smoothing the edges of the dark areas on the image. The technique consists of considering each dot in the image in turn. The dot is either left as it is

or changed to the opposite colour (white to black or black to white) depending upon the eight dots that surround it. The particular pattern of surrounding dots that are required to change the inner dot's colour is used to control the harshness of the algorithm [6], [8].

In our first set of experiments the result was definitely worse than the original. Although square-like characters such as H, L, and T came out very well, anything with slope (M, V, W, or S) became so bad that the oblique contours were stepped. The method was subsequently modified to produce a result that was far more acceptable; the image looked a lot cleaner than the original. Fig. 6 shows the same text as that in Fig. 5, but after it has been cleaned.

Fig. 6 A cleaned text

The effect of these can be difficult to see clearly. We have used the colour on our Grinnell display to show the original picture and the outcome of various picture processing operations superposed in different colours. This brings out the effect of the operations very

vividly.

It was mentioned above that the enhancement was done not only to improve the image for reading but also for easier processing by the computer. As described earlier, the image from the facsimile machine is compressed in order to reduce the amount of data. The cleaning allows a higher compression rate so that more efficient transmission and/or storage can be achieved.

We learned some important lessons from the enhancement exercise. Originally we thought that the main attraction in enhancement would be to improve the readability. In the end, we found that improving the readability was very difficult, especially because the facsimile image was so poor. Instead we found that the effect of reducing the compressed output was more important. By reducing the data to be transmitted by a quarter, significant savings could be made. But before such a technique could be used in a live system, the time it takes to produce the enhancement must be weighed against the time that would be saved in transmission.

2.4 Image Editing

By editing we mean that the facsimile picture can be changed, or combined with other pictures, while it is stored inside the computer. In previous sections it was mentioned that we could change the size and shape of a facsimile image. This technique was later combined with an overlaying method that enabled one picture to be combined with another [12].

In order to perform any editing it is necessary to have the picture displayed for the user to see. In our case we displayed the picture on the bit-map screen. The image took up the left-hand side of the screen, the right side being reserved for the picture that was being built. The user could select an area of the left-hand screen and move it to a position on the right-hand screen. Several images could be displayed in succession on the left, and areas selected and moved to the right. Finally, the right-hand screen could be printed on the facsimile machine.

The selection of an area of the picture was done by the use of a coloured rectangular subsection, controlled by a program in the computer, that could be moved around on the screen. The rectangular subsection

was moved with instructions typed in by the operator; it could be moved up or down, and increased or decreased in size. When the appropriate area of the screen had been selected, the program remembered the coordinates and moved the coloured rectangular subsection to the right-hand side of the screen. The user then selected an area again, in a similar manner. When the user finished the editing, the program removed the part of the picture selected from the left-hand screen and converted it to fit the shape of the rectangular subsection on the right-hand screen. The result was then displayed for the user to see.

When an image was being edited, the editor had to keep another scaled copy for display. This is due to the fact that the screen had a different dimension to that of the facsimile machine. The editing operations, e.g. chopping and merging, were performed on the original image data files with the full resolution available on the facsimile machine.

2.5 Integration with Other Data Types

The facsimile machine can be viewed in a wider context than merely a facsimile input/output device. It can work as a printer for other data representation types, such as coded character text and geometric graphics. At present, text can be converted into facsimile format and printed on the facsimile machine. Moreover, mixed pages containing pictures and text can be manipulated by our system. The integration of facsimile images with geometric graphics is a topic of future research.

In order to convert a character string into its facsimile format, the system maintains a translation table whereby the patterns of the characters available in the system can be retrieved. The input character string is translated into a set of scan lines, each of which is created by concatenating the corresponding patterns of the characters in the string.

The translation table is in fact a software font, which can be edited and modified. Even though only one font is available in our system for the time being, it is quite easy to introduce other character fonts. Furthermore, it is also possible for a font to be remotely loaded from a database via the communication network.

This allows for more interesting applications of the facsimile machine. For example, it could serve as a Teletex printer, provided that the Teletex character font is included in our system. In this case, the text images may be distorted to fit the presentation format requested by the Teletex service. Similarly, Prestel viewdata pages could be displayed on the Grinnell screen.

Moreover, pictures can be mixed with text by combining this text conversion with the editing described in the previous section. This should be regarded as a notable step towards multi-type processing.

Not only does this support a local multi-type environment but multi-type information can be transmitted over a network. So far as this facsimile system is concerned, a mixed page containing text and pictures can be sent only when it has been represented in a bit-map format. However, much more efficient transmission would be achieved if one could transmit the text and pictures separately and reproduce the page at the destination site. This requires that a multi-type data structure be designed which is understood by the two communication sites.

3. SYSTEM ARCHITECTURE

Now let us discuss the general disciplines for design and implementation of a computerised facsimile system which carries out the functions described in the previous sections. Having discussed the requirements of the system, a hierarchical model is introduced in which the modules of different layers are implemented as separate processes. The Clean and Simple interface, which is adopted for inter-process communication, is then described. The task controller, which is responsible for organising the tasks involved in a requested job, is discussed in detail. Some efforts have been made in our experimental work to provide a more convenient user programming environment and a more efficient data transfer method. This is finally described.

3.1 System Requirements

In a computerised facsimile system, the images are represented in a digital form. To carry out this

conversion, a page is scanned by the optical scanner of the facsimile machine, a digital number being produced to represent the darkness of each pixel. As high resolution has to be adopted to keep the detail of the image, the facsimile data files are usually rather large. In order to achieve efficient storage and transmission, the facsimile data must be compressed as much as possible.

Currently, the facsimile machines made by different manufacturers h different properties, such as different compression methods and different resolution. There are also some international standards for facsimile data compression, which are employed for the facsimile data to be transferred over the public data network. These require that the facsimile data be converted from one representation form to another, so that users who are separated geographically and use different machines can communicate with each other. More sophisticated applications, e.g. image editing, request processing facilities of the system as well.

When being processed, the facsimile image should be represented in a common format or internal data structure, which is used to pass the information between different processing routines. For the sake of convenience and efficiency, the internal data structure should be fairly well compressed and its format should be easy for the computer to manipulate. In our experimental work, the line vector is chosen as a standard unit, a simple run-length compression being employed [3]. Some processing routines may use other data formats, e.g. bit-map, but it is the responsibility of such routines to perform the conversion between those formats and the standard one.

The system should contain several processing routines, each of which performs one primitive task, such as chopping, merging, and scale-changing. An immense variety of processing operations can be carried out as long as those task modules can be organised flexibly. The capability for flexible task organisation should be thought of as one of the most important requirements of the system.

One possibility is for the processing routines involved to be executed separately, temporary files being used as communication media. Though very simple, this method is far too inefficient.

As described above, the information unit for the communication between the processing routines is the line vector, so that the routines can be organised as embedded loops, where a processing routine takes the input line from its source routine located in the inner loop, and passes the output line to the destination routine located in the outer loop [3]. Obviously this method is quite efficient. But it is not realistic for our system, because it is very difficult to build up different processing loops at run-time and flexible task organisation is impossible.

In a real-time operating system environment, the primitive tasks can be implemented as separate processes. This method, which is discussed in detail in the following sections, provides the required flexibility.

3.2 Hierarchical Model

As shown in Fig. 7, the modules in a single computer fall into three layers.

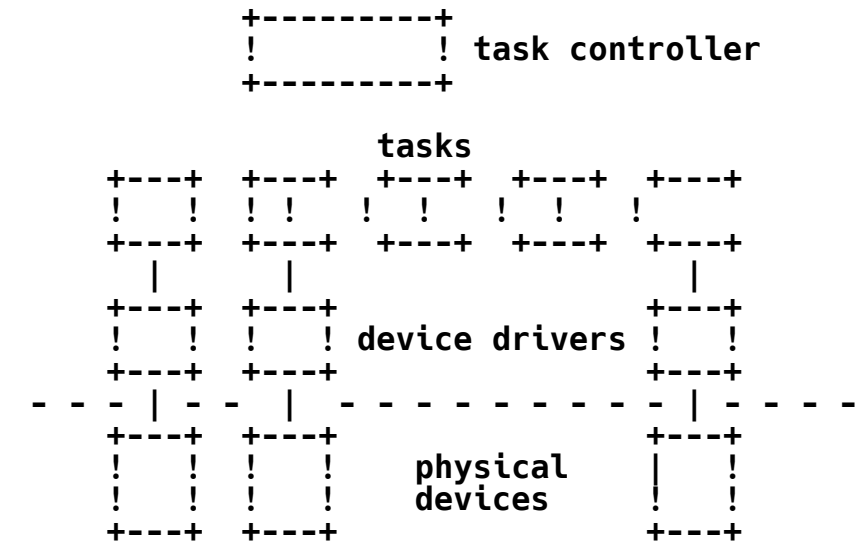


Fig. 7 The hierarchical model

These are:

(1) Device Drivers, which constitute the lowest layer in the model. The modules in this layer deal with I/O activities of the physical devices, such as

facsimile machine, display and floppy disk. This layer frees the task modules of upper layer from the burden of I/O programming.

(2) Tasks, which perform all processing primitives and handle different data structures. Above the driver of each physical device, there are one or more such device-independent modules, which work as information source or sink in the task chain (see below). A file system module allows other modules to store and retrieve information on the secondary storage device such as floppy disk. Decompression and recompression routines convert data structures of facsimile image information so that the facsimile machines can communicate with the rest of the system. Processing primitives, e.g. chopping, merging, scaling, are implemented as task modules in this layer. They are designed such that they can be concatenated to carry out more complex jobs. So far as the system is concerned, the protocols for data transmission over computer networks are also regarded as task modules in this layer.

(3) Task Controller, which organises the task processes to perform the specified job. It provides the users of the application layer with a procedure-oriented language whereby the requested job can be defined as a chain of task modules. Literally, the chain is represented by a character string:

<source_task>|{<processing_task>|}<sink_task>

According to such a command, the task controller selects the relevant task modules and concatenates them in proper order by means of logical links. Then the tasks on the chain are executed under its control, so that the data taken from the source are processed and the result is put into the sink.

3.3 Clean and Simple Interface

It is important, in this application, to develop the software in a modular way. It is desirable to put together a set of modules to carry out the different image processing tasks. Another set of transport modules must be developed for shipping data over the

different networks to which the UCL system is attached. In our computerised facsimile system, these task modules are implemented as separate processes. The operation of the system relies on the communication between these processes. The interface which is used for such communication has been designed to be universal; it is independent of these modules, and has been termed the Clean and Simple interface [20]. This interface is discussed in this section.

3.3.1 Principles

The Clean and Simple interface is concerned with the synchronisation and transfer of full-duplex data streams between two communicating processes. Thus the interface has three major components: connection synchronisation, data transfer and connection desynchronisation. These components are discussed below.

The connection between two processes is initiated by one of them, which, generally speaking, belongs to a higher layer. For example, the interface between protocols of different layers is always initiated by the higher layer, though, sometimes, the connection is initiated passively by the primitive 'listen'. It will be seen in the next section that task processes can communicate with each other via the connections to the higher layer (task controller) and this makes it possible to achieve flexible task organisation.

The process initiating the connection is called the 'master' process, while the other is called the 'slave' process. The 'master' process is also responsible for resource allocation for the two communicating processes. Here 'resource' refers mainly to the memory areas for the message structure and data buffer. This asymmetric definition of the interface eliminates any possible confusion in resource allocation.

The interface is implemented by using the signal-wait mechanism provided by the operating system. A data structure called CSB (Clean and Simple Block), which contains function, data buffer, and other information, is sent as the event message, when one process signals another [20].

3.3.2 Synchronisation and Desynchronisation

The procedure for connection synchronisation is composed of two steps. First, the two processes exchange their identifiers for the specific connection by means of a getcid primitive. Usually, the pointer to the task control structure of the process is used as the connection identifier.

Then, the 'master' sends an open CSB with appropriate parameter string passing the initialisation information. This information, which can also be called open parameter, is process dependent, or more accurately, task dependent. For example, the parameters for the file system should be the file name and the access mode. Provided the 'slave' accepts the request, the connection is established successfully and data can be transferred via the interface.

In order to desynchronise the connection, the 'master' initiates a 'close' action. On the other hand, an error state or EOF (end of file) state can be reported by the 'slave' to request a connection desynchronisation.

The listen primitive in our system is reserved for the processes that receive a request from the remote hosts on the networks.

3.3.3 Data Transfer

While the Clean and Simple interface is asymmetric in relation to connection synchronisation, data transfer is completely symmetric so long as the connection has been established. Data flows in both directions are permitted, though the operations are quite different.

The interface provides two primitives for data transfer -- read and write. To transfer some data to the 'slave', the 'master' signals it with a CSB containing the write function and a buffer filled with the data to be transferred. Having consumed the data, the 'slave' returns the CSB to report the result status of the transmission.

On the other hand, in order to receive some data from the 'slave', the 'master' uses a read CSB with an empty buffer. Having received the CSB, the 'slave' fills the buffer with the data requested and, then, returns the CSB.

3.4 Control and Organisation of the Tasks

Another important aspect of the multi-process architecture of the UCL facsimile system, is the need to systematise the control and organisation of the tasks. This activity is the function of the task controller, whose operations are discussed in this section.

3.4.1 Command Language

As mentioned earlier, the task controller supports a procedure-oriented language by means of which the user or the routines of the upper layers can define the jobs requested. A command should contain the following information:

1. the names of the task processes which are involved in the job.
2. the open parameters for these task processes.
3. the order in which the tasks are to be linked.

The last item is quite important, though, usually, the same order as that given in the command is used.

A command in this language is presented as a zero-ended character string. In the task name strings and the attribute strings of the open parameters, '|', '"', and ',' must be excluded as they will be treated as separators. The definition is shown below, where '|', which is the separator of the command strings in the language, does not mean 'OR'.

```
<command_string> ::= <task_string>
<command_string> ::= <task_string>|<command_string>
<task_string> ::= <task_name>
<task_string> ::= <task_name>"<open_parameter>
<open_parameter> ::= <attribute>
<open_parameter> ::= <attribute>,<open_parameter>
```

3.4.2 Task Controller

In our experimental work, the task controller module is called fitter. This name which is borrowed from UNIX hints how the module works. According to the command string, it links the specified tasks into a chain, along which the data is processed to fulfil the

job requested (Fig. 8).

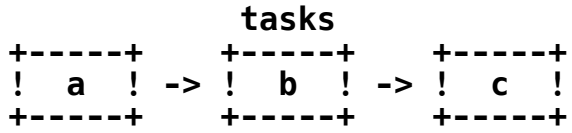


Fig. 8 The task chain

Since all modules, including fitter itself, are implemented as processes, the connections between modules should be via the Clean and Simple interfaces. Upon receiving the command string, the fitter parses the string to find each task process involved and opens a connection to it. Formally, the task processes are chained directly, but, logically, there is no direct connection between them. All of them are connected to the fitter (Fig. 9).

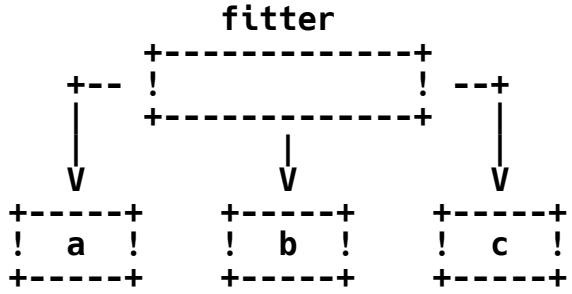


Fig. 9 The connection initiated by the fitter

For each of the processes it connects, the fitter keeps a table called pipe. When the command string is parsed, the pipe tables are double-linked to represent the specified order of data flow. So far as one process is concerned, its pipe table contains two pointers: a forward one pointing to its destination and a backward one pointing to its sources. Besides the pointers, it also maintains the information to identify the task process and the corresponding connection.

Fig. 10 illustrates the chain of the pipe tables for the job "a|b|c". Note that the forward (output) chain ends at the sink, while the backward (input) chain ends at the source. In this sense, the task processes are chained in the specified order via the fitter (Fig. 11). The data transfer along the chain is initiated and controlled by the fitter, each process getting the input from its source and putting the output to its destination.

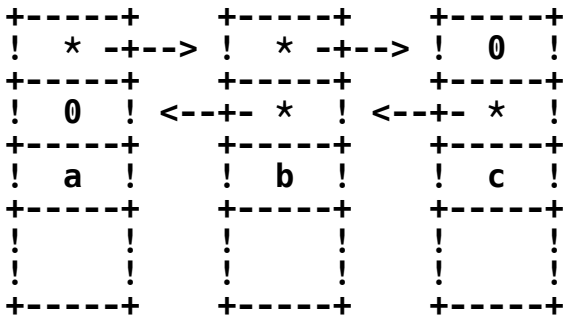


Fig. 10 The pipe chain

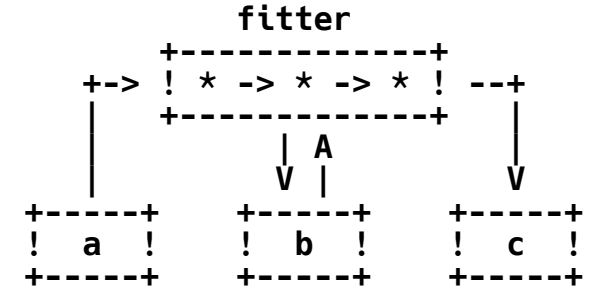


Fig. 11 The data flow

This strategy makes the task organisation so flexible that only the links have to be changed when a new task chain is to be built up. In such an environment, each task process can be implemented independently, provided the Clean and Simple interface is supported. This also makes the system extension quite easy.

The fitter manipulates one job at a time. But it must maintain a command queue to cope with the requests, which come simultaneously from either the upper level processes or other hosts on the network.

3.5 Interface Routines

In a modular, multi-process system such as the UCL facsimile system, the structure of the interface routines is very important. The CSI of section 3.3 is fundamental to the modular interface; a common control structure is also essential. This section gives some details both about the sharable control structure and the buffer management.

3.5.1 Sharable Control Structure

Though the CSI specification is straightforward, the implementation of the inter-process communication interface may be rather tedious, especially in our system, where there are many task processes to be written. Not only does each process have to implement the same control structure for signal handling, but also the buffer management routines must be included in all the processes.

For the sake of simplicity and efficiency, a package of standard interface routines is provided which are shared by the task processes in the system. These routines are re-entrant, so that they can be shared by all processes.

The 'csinit' primitive is called for a task process to check in. An information table is allocated and the pointer to the table is returned to the caller as the task identifier, which is to be used for each call of these interface routines.

Then, each task process waits by invoking the 'csopen' primitive which does not return until the calling process is scheduled. When the connection between the process and the fitter is established, the call returns the pointer to the open parameter string of the task, the corresponding task being started. A typical structure of the task process (written in c) is shown below. After the task program is executed, the process calls the 'csopen' and waits again. It can be seen that the portability of the task routines is improved to a great extent. Only the interface routines

should be changed if the system were to run in a different operating environment.

```
static int mytid;       /* task identifier */

task()
{
        char *op;       /* open parameter */

        mytid = csinit();
        for(;;) {
                op = csopen(mytid);
                ...     /* the body of the task */
        }
}
```

3.5.2 Buffer Management

The package of the interface routines also provides a universal buffer management, so that the task processes are freed from this burden. The allocation of the data buffers is the responsibility of the higher level process, the fitter. If the task processes allocated their own buffers, some redundant copying would have to be done. Thus, the primitives for data transfer, 'csread' and 'cswrite', are designed as:

```
        char *csread(tid, need);
        char *cswrite(tid, need);
```

where 'tid' is the identifier of the task and 'need' is the number of data bytes to be transferred. The primitives return the pointer to the area satisfying the caller's requirement. The 'csread' returns an area containing the data required by the caller. The 'cswrite' returns an area into which the caller can copy the data to be transferred. The copied data will be written to its destination at a proper time without the caller's interference. Obviously the unnecessary copy operations can be avoided. It is recommended that the data buffer returned by the primitives be used directly to attain higher performance.

In order to implement this strategy, each time a piece of data is required, the size of the buffer needed is compared with that of the unused buffer area in the current CSB. If the latter is not less than the former, the current buffer pointer is returned. Otherwise, a temporary buffer has to be employed. The data is copied into the buffer until the requested size is reached. In this case, instead of a part of the current buffer, the temporary buffer will be returned.

A 'cswrite' call with the 'need' field set to zero tells the interface routine that no more data will be sent. It causes a 'close' CSB to be sent to the destination routine.

If there is not enough data available, 'csread' returns zero to indicate the end of data.

4. UCL FACSIMILE SYSTEM

Now we discuss the implementation of the computerised facsimile system developed in the Department of Computer Science at UCL.

This system has several components. Since the total system is a modular and multi-process one, a specific system must be built up for a specific application. The way that this is done is discussed in section 4.1. The specific devices and their drivers are described in section 4.2. The system can be attached to a number of networks. In the UCL configuration, the network interface can be direct to SATNET [22], SERC NET [23], PSS [24], and the Cambridge Ring. The form of network connection is discussed further in section 4.3. The system must transfer data between the facsimile devices and the disks, and between the networks and the disks. For this a filing system is required which is discussed in section 4.4.

A key aspect of the UCL system is flexibility of devices, networks, and data formats. The flexibility of device is achieved by the modular nature of the device drivers (section 4.2). The flexibility of network is discussed in section 4.8. The additional flexibility of data structure is described in section 4.5. The flexibility can be utilised by incorporating conversion routines as in section 4.6. An important aspect of the UCL system is the ability to provide local manipulation facilities for the graphics files. The facilities implemented for the local manipulation are discussed in

section 4.7. In order to transfer files over the different networks of section 4.3. a high level data transmission protocol must be defined. The procedures used in the UCL system are discussed in section 4.8.

4.1 Multi-Task Structure

The task controller and processing tasks are implemented as MOS processes. A number of utility routines are provided for users to build new task processes and modules at application level.

In the environment of MOS, a process is included in a system by specifying a Process Control Table when the system is built up. The macro 'setpcte' is used for this purpose, the meaning of its parameters being defined in [14].

```
#define setpcte(name,entry,pridev,prodev,stklen,
    relpid,relopc)
  {0,name,entry,pridev,prodev,stklen,relpid,relopc}
```

A Device Control Table (DCT) has to be specified for each device when the system is built up. A DCT can be defined anywhere as devices are referenced by the DCT address. The macro 'setdcte' is designed to declare devices, the meanings of its parameters being specified in [14]. This method is used in the device descriptions.

```
#define setdcte(name,intvec,devcsr,devbuf,devinit,
    ioinit,intrpt,mate)
  {04037,intrpt,0,0,name,mate,intvec,devinit,
    devcsr,devbuf,ioinit}
```

4.2 The Devices

As mentioned in section 2, apart from the general purpose system console, there are three devices in the system to support the facsimile service. These are:

(1) AED62 Floppy Disk, which is used as the secondary memory storing the facsimile image data. Above its driver, a file system is implemented to manage the data stored on the disks, so that an image data

file can be accessed through the Clean and Simple interface. This file system is dicussed in detail in the next section. For some processing jobs, the image data has to buffered on a temporary file lest time-out occurs on the facsimile machine.

(2) DACOM Facsimile Machine, which is used to input and output image data. It reads an image and creates the corresponding data stream. On other hand, it accepts the image data and reproduces the corresponding image. Above its driver, there is a interface task to fit the facsimile machine into the system, the Clean and Simple interface being supported. The encoding algorithm for the DACOM machine is described in [19].

(3) Grinnell Colour Display, which is used as the monitor of the system. Above its driver, an interface task is implemented so that the image data in standard format can be accepted through the Clean and Simple interface.

The detailed description of these devices can be found in Appendix 1. The interface task and the description for each device are listed in the following table. The interface tasks can be directly used as data source or sink in a task string.

Device	Interface Task	Description
AED62 Floppy Disk	fs()	aed62(device)
DACOM fax Machine	fax()	dacom(device)
Grinnell Display	grinnell()	grinnell(device)

Note that the DCTs for the facsimile machine and Grinnell display have been included in the corresponding interface tasks, so that there is no need to declare them if these tasks are used.

4.3 The Networks

There are three relevant wide-area networks terminating in the Department of Computer Science at the end of 1981. These are:

(1) A British Telecom X25 network (PSS, [24]).

(2) A private X25 network (SERC NET, [23])

(3) A Defence network (ARPANET/SATNET, [21], [22])

In addition there is a Cambridge Ring as a local network.

For the time being, the UCL facsimile system is directly attached to the various networks at the point NI (Network Interface) of Fig. 1.

As mentioned earlier, pictures can be exchanged via the SATNET/ARPANET, between UCL in London, ISI in Los Angeles, and COMSAT in Washington D.C.. The Network Independent File Transfer Protocol (NIFTP, [9]) is used to transfer the image data. This protocol has been implemented on LSI under MOS [10]. In addition, we at UCL have put NIFTP on an ARPANET TOPS-20 host, which can act as an Internet File Forwader (IFF). In this case, TCP/IP ([28], [29]) is employed as the underlying transport service. Since TCP provides reliable communication channels, the provision of checkpoints and error-recovery procedures are not included in our NIFTP implementations.

In the X25 network, the transport procedure is NITS/X25 ([25], [26]). Though pictures can be transferred to the X25 networks, no experimental work has been done, because:

(1) There is at present no collaborative partner on these networks.

(2) The LSI-11, on which our system is implemented, has no direct connection to these networks.

Locally, image data can be transmitted to the PDP11-44s running the UNIX time-sharing operating system. At present, the SCP ring-driver software uses permanent virtual circuits (PVCs) to connect the various computers on the ring.

4.4 File System

A file system has been designed, based on the AED62 double density floppy disk, for use under MOS. It is itself implemented as a MOS process supporting the Clean and Simple interface. The description of this task, fs(fax), can be found in Appendix 2.

In a command string, the file system task can only serve as either data source or data sink. In other words, it can only appear at the first or last position on a command string. In the former case, the file specified is to be read, while the file is to be written in the latter case.

Three access modes are allowed which are:

* Read a file
* Create a file
* Append a file

The file name and access mode are specified as the open parameters.

Let us consider an example. If a document is to be read on the facsimile machine and the data stream created is to be stored on the file system, the command string required is:

```
        fax"r|fs"c,doc

where:  fax - interface task for facsimile machine
        r   - read from facsimile machine
        fs  - file system task
        c   - create a new file
        doc - the name of the file to be created.
```

In order to dump a file, a task process od() is provided which works as a data sink in a command string.

4.5 Data Structure

Facsimile image data is created using a high-resolution raster scanner, so that the original picture can be reproduced faithfully. The facsimile data represents binary images, in monochrome, with two levels of intensity, belonging to the data type of bit-mapped graphics.

The simplest representation is the bit-map itself. The bits, each of which corresponds to a single picture element, are arranged in the same order as that in which the original picture is scanned, 1s standing for

black pixels and 0s for white ones. Operations on the picture are easily carried out. For example, two images represented in the bit-map format can be merged together by using a simple logic OR operation. Any specific pixel can be retrieved by a simple calculation. However, its size is usually large because of the high resolution. This makes it almost unrealistic for storage or transmission.

Facsimile image data should therefore be compressed to reduce its redundancy, so that the efficient storage and transmission can be achieved.

Run-length encoding is a useful compression scheme. Instead of the pattern, the counts of consecutive black and white runs are used to represent the image.

Vector representation, in which the run-lengths are coded as integers or bytes, is a useful internal representation of images. Not only is it reasonably compressed, but it is also quite easy for processing. Chopping, scaling and mask-scanning are examples of the processing operations which may be performed. Furthermore, a conversion between different compression schemes may have to be carried out in such a way that the data is first decompressed into the vector format and then recompressed. The difficulty in retrieval can be overcome by means of line index, which gives the pointers to each lines of the image.

A higher compression rate leads to a more efficient transmission. But this is at the expense of ease of processing. An example of this is the use of Huffman Code in the CCITT 1-dimensional compression scheme. While the data can be compressed more efficiently, it is rather difficult to manipulate the data direclty.

Taking the correlation between adjacent lines into account, 2-dimensional compression can achieve an even higher compression rate. CCITT 2-dimensional compression and the DACOM facsimile machine use this method.

It is desirable to integrate facsimile images with other data types, such as text and geometric graphics; the structure of these other types must then be incorporated in the system. At present, only text structure is available, while the structure for geometric graphics is a topic for the further study.

In the facsimile system, the following data structures are supported. The corresponding descriptions, if any, are listed as well and they can be found in Appendix 3 (except of dacom(device)).

type	structure	compression	description
bit-map	bit-map	-	-
	vector	1D run-length	vector(fax)
	dacom block	2D run-length	dacom(device)
	CCITT T4	1D run-length	t4(fax)
		2D run-length	t4(fax)
text	text	-	text(fax)

As an internal data structure, vector format is widely used for data transfer between task processes. The set of interface routines has been extended by introducing two subroutines, namely getl() and putl(), which read and write line vectors directly through the Clean and Simple interface. These two routines can be found in Appendix 3 (getl(fax) and putl(fax))

In order to check the validity of a vector file, a check task process check() is provided which works as a data sink in a command string. It can also dump the vector elements of the specific lines.

4.6 Data Conversion

In order to convert one data structure into another, several conversion modules are provided in this system. These modules fall into two categories, task processes and subroutines. The task processes are MOS processes which can only be used in the environment described in this note, while the subroutines which are written in c and compatible under UNIX are more generally usable.

Character strings or text can be converted into vector format, so that an integrated image combining picture and text can be formed.

The following table lists these conversion modules, including their functions and descriptions (which can be found in Appendix 3).

module	type	from	to	description
decomp	process	dacom	vector	decomp(fax)
recomp	process	vector	dacom	recomp(fax)
ccitt	process	vector	t4	ccitt(fax)
		t4	vector	
bitmap	subroutine	vector	bitmap	bit-map(fax)
tovec	subroutine	bitmap	vector	tovec(fax)
ts	subroutine	ASCII string	vector	ts(fax)
string	process	ASCII string	vector	string(fax)
tf	process	text	vector	tf(fax)

Since each DACOM block contains a Cyclic Redundancy Check (CRC) field, the system supplies a subroutine crc() to calculate or check the CRC code. (see crc(fax))

If a vector file is to be printed on the DACOM facsimile machine, the image data should be re-compressed into the DACOM-block format, the required command string being shown below.

```
fs"e,pic|recomp|fax"w

where   fs     - file system task
        e      - read an existing file
    ic    - file name
        recomp - re-compression task
        fax    - interface task for facsimile machine
        w      - print an image on facsimile machine
```

4.7 Image Manipulation

Four processing task processes are provided in the system. These are:

(1) Chop, which applies a defined window to the input image.

(2) Scale, which enlarges or shrinks the input image to the defined dimensions.

(3) Merge, which puts the input image on the specified area of a background image.

(4) Clean, which removes the noise on the input image.

The Clean and Simple interfaces are supported in these processing tasks so that the tasks can be used in command strings. However, these tasks can be neither source nor sink in a command string. The data format of their input and output is vector.

For example, a facsimile page can be cleaned and then printed on the facsimile machine. Note that the image data must be recompressed before being sent to the facsimile machine. If the original data is the form of DACOM block, it has to be decompressed as the processing tasks only accept line vectors. The required command string is shown below.

```
fs"e,page|clean|recomp|fax"w

where   fs     - file system task
        e      - read an existing file
        page   - file name
        clean  - cleaning task
        recomp - re-compression task
        fax    - interface task for facsimile machine
        w      - print an image on facsimile machine
```

The descriptions of these processing tasks can be found in Appendix 2 (chop(fax), scale(fax), merge(fax), and clean(fax)).

In tasks 'chop' and 'merge', a window is set by giving the coordinates of its vertices. However, it is usually rather difficult for a human user to decide the exact coordinates. The system supplies a subroutine choice() which specifies a rectangular subsection of an image by interactive manipulations of a rectangular subsection on the screen of the Grinnell display displaying the image. It provides a set of interactive commands whereby a user can intuitively choose an area he is interested in. Note that this subroutine must be called by a MOS process and the Grinnell display must be included in the system.

By means of these image processing modules, the image editing described in section 2.4 can be carried out. Let us consider an example. An image abstracted from a picture 'a' is to be merged onto a specified area of another picture 'b'. First of all, the two pictures 'a'

and 'b' should be displayed on the left half and right half of the screen, respectively. Assume that the two pictures are standard DACOM pages whose dimensions are 1726x1200. They have to be shrunk to fit the dimension of the half screen (256x512). Note that if the data format is not vector, conversion should be carried out first. the required command strings are:

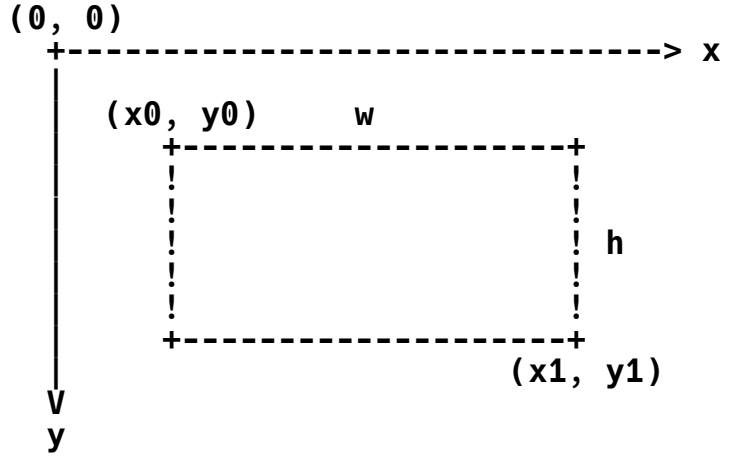
```
e,a|scale"1726,1200,256,512|grinnell"0,511,255,0,z,g
fs"e,b|scale"1726,1200,256,512|grinnell"256,511,511,0,z,b
```

```
where   fs             - file system task
        e              - read an existing file
        a              - file name
        b              - file name
        scale          - scale task
        1726,1200      - old dimension
        256,512        - new dimension
        grinnell        - grinnell display interface task
        0,511,255,0    - presentation area (the left half)
        256,511,511,0  - presentation area (the right half)
        z              - zero write mode
        g              - green
        b              - blue
```

In an application process, the subroutine choice() is called in the following ways for the user to choose the areas on both pictures.

```
choice(r, 1726, 1200, 1, 0, 0);
        /* choice the area on 'a' */
        /* r    - red
           1726 - width of the original picture
           1200 - height of the original picture
           1    - left half of the screen
           0    - the subsection can be of any width
           0    - the subsection can be of any height
         */
choice(r, 1726, 1200, 2, 0, 0);
        /* choice the area on 'b' */
        /* r    - red
           1726 - width of the original picture
           1200 - height of the original picture
           2    - right half of the screen
           0    - the subsection can be of any width
           0    - the subsection can be of any height
         */
```

When the user finishes editing, the coordinates of the chosen rectangular areas are returned. An example is given in the table below. The widths and heights listed in the table are actually calculated from the coordinates returned and they indicate that the source image has to be enlarged to fit its destination.



original	x0	y0	x1	y1	w	h
a	30	40	100	120	70	80
b	100	100	1100	1100	1000	1000

At this stage, our final goal can be achieved by performing a job specified below. It is assumed that the result image is to be stored as a new file 'c'.

```
fs"e,a|chop"30,40,100,120|scale"70,80,1000,1000
    |merge"b,0,100,100,1100,1100|fs"c,c

where   fs                - file system task
        e                 - read an existing file
        a                 - file name
        chop              - chop task
        30,40,100,120     - the area to be abstracted
        scale             - scale task
        70,80             - old dimension
        1000,1000         - new dimension
        merge             - merge task
        b                 - file name of the background image
        0                 - to be overlaid
        100,100,1100,1100 - the area to be overlaid
        fs                - file system task
        c                 - create a new file
        c                 - the name of the file to be
                            created
```

4.8 Data Transmission

In order to transmit facsimile image data over computer networks, using the configuration of Fig. 1, the Network Independent File Transfer Protocol [9] is implemented as a MOS task process, the Clean and Simple interface of section 3.3 being supported [10]. Thus this module can be used in a command string directly. In this case, the module always works in the initiator mode, though the server mode is supported as well. Its description can be found in Appendix 2 (ftp(fax)).

As a network-independent protocol, it employs a transport service to communicate across the networks. The Clean and Simple interface is also used for the communication between the module and transport service processes.

Suppose that an image file stored in a remote file system is to be printed on the local facsimile machine. Assume that the data is transmitted via the ARPANET [21], Transport Control Protocol (TCP) [28] being used as the underlying transport service. As was described

before, since the delay caused by the network may result in a time-out on the local facsimile machine, the job should be divided into two subjobs.

(1) The remote file is transmitted by using NIFTP module. However, instead of being put on the facsimile machine directly, the received data is store in a temporary file.

```
ftp"r,b,ucl,fax,pic;tcp:1234,10,3,3,42,4521|fs"c,tmp

where   ftp - NIFTP task
        t   - receive
        b   - binary
        ucl - remote user name
        fax - remote password
        pic - remote file name
        tcp - transport service process

        parameters for the transport service:

            1234      - local channel number
            10,3,3,42 - remote address
            4521      - channel reserved for the
                        remote server

        fs  - local file system task
        c   - create a new file
        tmp - the name of the file to be created
```

(2) The temporary file is read and the image is sent to the facsimile machine for printing. Here it is assumed the data received is in the form of DACOM block so that no conversion is needed.

```
fs"e,tmp|fax"w

where   fs     - file system task
        e      - read an existing file
        tmp    - file name
        fax    - interface task for facsimile machine
        w      - print an image on facsimile machine
```

We are able to exchange image data with ISI and COMSAT. At present DACOM block is the only format that can be used as all the three participants in this experiment possess DACOM facsimile machines and no

other data format is available in both ISI and COMSAT. However, it is the intention of the ARPA-Facsimile community to adopt the CCITT standard for future work. As mentioned earlier, UCL already has this facility.

Above NIFTP, a simple protocol was used to control the transmission of facsimile data. In this protocol, the format of a facsimile data file was defined as follows: Each DACOM block was recorded with a 2-byte header at the front. This header was composed of a length-byte indicating the length of the block (including the header) and a code-byte indicating the type of the block. This is shown in the following diagram.

The Length-byte is 76 (decimal) for all DACOM blocks. The code-byte for a setup block is 071 (octal) and 072 for a data block. A special EOP block was used to indicate the end of a page. This block had only the header with the length-byte set to 2 and the code-byte undefined. A facsimile data file could contain several pages, which were separated by EOP blocks.

5. CONCLUSION

5.1 Summary

Though techniques for facsimile transmission were invented in 1843, it was not until the recent years that integration with computer communication systems gave rise to "great expectation". The system described in this note incarnates the compatibility and flexibility of computerised facsimile systems.

In this system, facsimile no longer refers simply to the transmission device, but rather to the function of transferring hard copy from one place to another. Not only does the system allow for more reliable and accurate document transmission over computer networks but images can also be manipulated electronically. Image is converted from one representation format to another, so that different makes of facsimile machines can communicate with each other. It is possible for a

picture to be presented on different bit-map devices, e.g. TV-like screen, as it can be scaled to overcome the incompatibilities. Moreover, the system provides windowing and overlaying facilities whereby a sophisticated editor can be supported.

One of the most important aspects of this system is that text can be converted into its bit-mapped representation format and integrated with pictures. Geometric graphics could also be included in the system. Thus, the facsimile machine may serve as a printer for multi-type documents. It is clear that facsimile will play an important role in future information processing system.

As far as the system per se is concerned, the following advantages can be recognised. Though our discussion is concentrated on the facsimile system, many features developed here apply equally well to other information-processing systems.

(1) Flexibility: The user jobs can be easily organised. The only thing to be done for this purpose is to make the logical links for the appropriate task processes.

(2) Simplicity: The interface routines are responsible for the operations such as signal handling and buffer management. By avoiding this burden, the implementation of the task processes becomes very "clean and simple".

(3) Portability: The interface routines also makes the task processes totally independent of the operating environment. Only these routines should be modified if the environment were changed.

(4) Ease of extension: The power of the system can be simply and infinitely extended by adding new task processes.

(5) Distributed Environment: This approach can be easily extended to a distributed environment, where limitless hardware and software resources can be provided.

5.2 Problems

As discussed earlier, the network we were using for the experimental work was not designed for image data

transmission. The data transfer is so slow that a time-out may be caused on the facsimile machine. Though this problem was solved by means of local buffering and pictures were successfully exchanged over the network, the slowness is rather disappointing because of the quantity of image data. The measurement showed that the throughput was around 500 bits/sec. In other words, it took at least 5 minutes to transfer a page. This was caused by the network but not our system. The situation has been improved recently. However, It is nevertheless required that more efficient compression schemes be developed.

At present, the system must be directly attached to the network to be accessed. However, the network ports are much demanded, so that frequent reconfiguration is required.

The facsimile system can be connected only to the local network, the Cambridge Ring, while the foreign networks are connected via gateways to the ring. This is shown in Fig. 12. Now the X25 network is attached to the Ring via an X25 gateway, XG [25], while SATNET is connected by another gateway, SG [25]. Both network are at the transport level; XG and SG support the relevant transport procedures. In the case of XG, this is NITS/X25 ([26], [27]); in the case of SATNET, it is TCP/IP ([28], [29]).

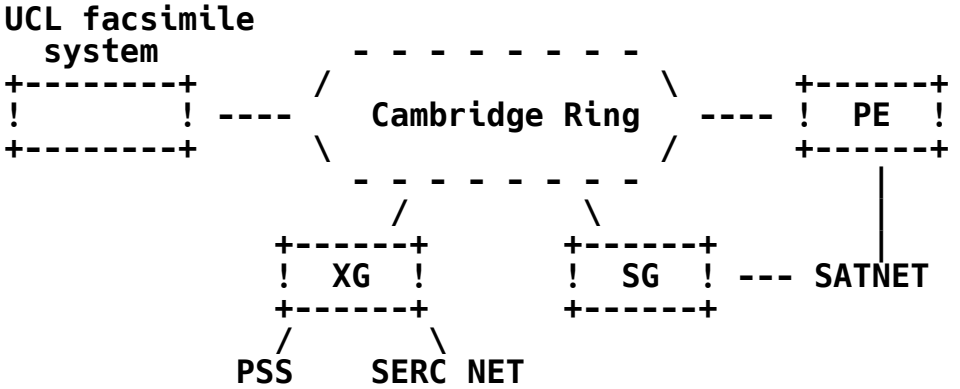


Fig. 12 Schematic of UCL network connection

When the network software runs in the same machine as the application software, the Clean and Simple interface of section 3.5 was used as an interface between the modules. When the gateway software was removed to a separate machine, an Inter-Processor Clean

and Simple [30] was required. The appropriate transport process is transferred to the relevant gateway, and appropriate facilities are implemented for addressing the relevant gateway. Otherwise, the software has to be little altered to cater for the distributed case.

In our experimental work, the following problems were also encountered.

(1) The primary memory of the LSI-11 is so small that we cannot build up a system to include all the modules we have developed. In order to transfer an edited picture using the NIFTP module, we have to first load an editor system to input and process the picture, and then an NIFTP system is then loaded to transmit it.

(2) The execution of an image processing procedure becomes very slow. For example, it takes several minutes to shrink a picture to fit the screen of the Grinnell display. This prevents the system from being widely used in its present form.

(3) As secondary storage, floppy disks are far from adequate to keep image data files. At present, we have two double-density floppy disk drives, the capacity of each disk being about 630K bytes. However, an image page contains at least 50K bytes and, sometimes, this number may be doubled for a rather complex picture. Only a limited number of pages can be stored.

On the other hand, in our department, we have two PDP11-44s running UNIX together with large disks supplying abundant file storage. Their processing speed is much higher than that of the LSIs. The UNIX file system supports a very convenient information-management environment. This inspired the idea that the UNIX file system could pretend to be a file server responsible for storing and managing the image data, so that all the processing tasks may be carried out on UNIX. Not only does this immediately solve the problems listed above, but the following additional advantages immediately accrue.

(1) UNIX provides a far better software-development environment than LSI MOS ever can or will.

(2) The facsimile service can be enhanced to be able

to support many users at a time.

(3) The UNIX file system is so sophisticated that more complex data entities can be handled.

In fact the 44s and the LSI-11, to which the facsimile machine and Grinnell display are attached, are all connected to the UCL Cambridge Ring. A distributed processing environment can be built up where a job in one computer can be initiated by another and then the job will be carried out by cooperation of both computers.

In such a distributed system, the LSI-11 micro-computer, together with the facsimile machine, constitutes a totally passive facsimile server controlled by a UNIX user. A page is read on the facsimile machine and the image data stream produced is transmitted to the UNIX via the ring. The image data is stored as a UNIX file and may be processed if necessary. It can also be sent via the ring to the facsimile server where it will be reprinted on the facsimile machine.

In order to build up such a distributed environment, IPCS [30] is far from adequate for this purpose, as it does not provide any facility for a remote job to be organised. In our system, the task controller can be modified so that the command strings can be supplied from a remote host on the network. Having accepted the request, the task controller organises the relevant task chain and the requested job is executed under its control. The execution of the distributed job may require synchronisation between the two computers. These problems are discussed in detail in [31].

Generally speaking, a distributed system based on a local network, which supplies cheap, fast, and reliable communication, could be the ultimate solution of the operational problems discussed in this section. In such a system, different system operations are carried out in the most suitable places.

For the time being, only a procedure-oriented task-control language is available in this system. The command string of the fitter can be typed from the system console directly, the corresponding job being organised and executed. Theoretically, this is quite enough to cope with any requirement of a user. However, when the job is complex, command typing becomes very tedious and prone to error.

Above the task-controller, a job-controller layer is required which provides a problem-oriented language whereby the user can easily put forward his requirement to the system. On receipt of such a command, the job controller translates it into a command string of the task controller and passes the string to the task controller so that operation request can be done. Sometimes, one job has to be divided into several subjobs, which are to be dealt with separately. The job controller should be also responsible for high level calculation and management, so that the user need not be concerned with system details.

In the system supporting facsimile service under UNIX, a set of high-level command is provided, while the command strings for the facsimile station are arranged automatically and they are totally hidden from a UNIX user.

5.3 Future Study

At the next stage, our attention should be moved to a higher-level, more sophisticated system which supports a multi-type environment. In such a system, not only does the facsimile machine work as an facsimile input/output device, but it should also play the role of a printer for the multi-type document. This is because other data types, e.g. coded character text and geometric graphics can be easily converted into bit-mapped graphics format which the facsimile machine is able to accept.

First of all, a data structure should be designed to represent multi-type information. In a distributed environment, such a structure should be understood all over the system, so that multi-media message can be exchanged.

In a future system, different services should be supported, including viewdata, Teletex, facsimile, graphics, slow-scan TV and speech. The techniques developed for facsimile will be generalised for use of other bit-mapped image representations, such as slow-scan TV.

To improve the performance of the facsimile system, we are investigating how we could use an auxiliary special purpose processor to perform some of the image processing operations. Such a processor will be essential for the higher data rate involved in slow-

scan TV.

Reference

[1] P. T. Kirstein, "The Role of Facsimile in Business Communication", INDRA Note 1047, Jan. 1981.

[2] T. Chang, "A Proposed Configuration of the Facsimile station", INDRA Note 922, May, 1980.

[3] T. Chang, "Data Structure and Procedures for Facsimile Signal Processing", INDRA Note 923, May, 1980.

[4] S. Treadwell, "On Distorting Facsimile Image", INDRA Note No 762, June, 1979.

[5] M. G. B. Ismail and R. J. Clarke, "A New Pre-Processing Techniques for Digital Facsimile Transmission", Dept. of Electronic Engineering, University of Technology, Loughborough.

[6] T. Chang, "Mask Scanning Algorithm and Its Application", INDRA Note 924, June, 1980.

[7] M. Kunt and O. Johnsen, "Block Coding of Graphics: A Tutorial Review", Proceedings of the IEEE, special issue on digital encoding of graphics, Vol. 68, No 7, July, 1980.

[8] T. Chang, "Facsimile Data Compression by Predictive Encoding", INDRA Note No 978, May. 1980.

[9] High Level Protocol Group, "A Network Independent File Transfer Protocol", HLP/CP(78)1, alos INWG Protocol Note 86, Dec. 1978.

[10] T. Chang, "The Implementation of NIFTP on LSI-11", INDRA Note 1056, Mar. 1981.

[11] T. Chang, "The Design and Implementation of a Computerised Facsimile System", INDRA Note No. 1184, Apr. 1981.

[12] T. Chang, "The Facsimile Editor", INDRA Note 1085, Apr. 1981.

[13] K. Jackson, "Facsimile Compression", Project Report, Dept. of Computer Science, UCL, June, 1981.

[14] R. Cole and S. Treadwell, "MOS User Guide", INDRA Note 1042, Jan. 1981.

[15] CCITT, "Recommendation T.4, Standardisation of Group 3 Facsimile Apparatus for Document Transmission", Geneva, 1980.

[16] "DACOM 6450 Computerfax Transceiver Operator Instructions", DACOM, Mar. 1977.

[17] "AED 6200LP Floppy Disk Storage System", Technical Manual, 105499-01A, Advanced Electronics Design, Inc. Feb. 1977.

[18] "The User Manual for Grinnelll Colour Display".

[19] D. R. Weber, "An Adaptive Run Length Encoding Algorithm", ICC-75.

[20] R. Braden and P. L. Higginson, "Clean and Simple Interface under MOS", INDRA Note No. 1054, Feb. 1981.

[21] L. G. Roberts et al, "The ARPA Computer Network", Computer Communication Networks, Prentice Hall, Englewood, pp485-500, 1973.

[22] I. M. Jacobs et al: "General Purpose Satellite Network", Proc. IEEE, Vol. 66, No. 11, pp1448-1467, 1978.

[23] J. W. Burren et al, "Design fo an SRC/NERC Computer Network", RL 77-0371A, Rutherford Laboratory, 1977.

[24] P. T. F. Kelly, "Non-Voice Network Services - Future Plans", Proc. Conf. Business Telecommunications, Online, pp62-82, 1980.

[25] P. T. Kirstein, "UK-US Collaborative Computing", INDRA Note No. 972, Aug. 1980.

[26] "A Network Independent Transport Service", PSS User Forum, Study Group 3, British Telecom, London, 1980.

[27] CCITT, Recommendation X3, X25, X28 and X29 on Packet Switched Data Services", Geneva 1978.

[28] "DoD Standard Transmission Control Protocol", RFC761, Information Sciences Inst., Marina del

Rey, 1979.

[29] "DoD Standard Internet Protocol", RFC760, Information Sciences Inst., Marina del Rey, 1979.

[30] P. L. Higginson, "The Orgainisation of the Current IPCS System", INDRA Note No. 1163, Oct. 1981.

[31] T. Chang, "Distributed Processing for LSIs under MOS", INDRA Note No. 1199, Jan. 1982.

Appendix I: Devices

NAME

aed62 - double density floppy disk

SYNOPSIS

```
DCT aed62
setdct("aed62", 0170, 0170450, 0170450,
        aedini, aedsio, aedint, 0);
```

DESCRIPTION

The Double Density disks contain 77 tracks numbered from 0 to 76. There are 16 sectors (sometimes called blocks) per track, for a total of 1232 sectors on each side of the disk. These are numbered 0 to 1231. Each sector contains 512 bytes, for a total of 630,784 bytes on each side of the floppy.

Only one side of the floppy can be accessed at a time. There is only one head per drive, and it is located on the underside of the disk. To access the other side, the disk must be manually removed and inserted the other way up.

Each block is actually two blocks on the disk: an adddress ID block and the data block. The address ID block is used by the hardware and contains the track number, the block number and the size of the data block that follows. When an operation is to take place, the seek mechanism first locates the block by reading the address ID blocks and literally 'hunting' for the correct one. It will hunt for up to 2 seconds before reporting a failure.

Both the address ID and the data blocks are followed by a checksum word that is maintained by the hardware and is hidden from the user. On writing, the checksum is calculated and appended to the block. On reading it is verified (both on reading the ID and data blocks) and any error is reported as a Data Check. No checking on the data block takes place on a write, and the hardware has no idea if it was written correctly. The only way to verify it is to read it.

Although there are two drives in the unit, they cannot be used simultaneously. If an operation is in progress on one, no access can be made to the other until the first operation is complete. The driver will queue requests for both drives however, and ensure that are performed in order.

The MOS driver is called aed62.obj. It operates on the following IORB entries:

irfnc

The operation to be performed, as follows:

0 - Read
1 - Write
2 - Verify
3 - Seek

Read and Write cause data to be transferred to and from disk. Verify does a hardware read without transferring the data to memory and is used for verifying that the data can be successfully read. The checksum at the end of the block of each sector is verified by the hardware. The seek command is used to move the disk heads to a specified track.

irusr1

The drive number. Only Zero or One is accepted. This is matched against the number dialed on the drive. If the number is specified on both drives, or neither, a hardware error will be reported.

irusr2

The Sector or Block Number. Must be in the range 0 to 1231 inclusive. irusr2 specifies the block number that the transfer is to begin at for Read and Write, the beginning of the verified area for the Verify command, and the position of the head for the Seek command. In the latter case the head will be positioned to the track that contains the block.

iruva

This specifies the data adress, which must be even (word boundary). If an odd address is given, the low order bit is set to zero to make it even. Not required for the Seek or Verify commands.

irbr

Transfer length as a positive number of bytes. Not required for the seek command, bit IS used by Verify command so that the correct number of blocks may be verified. The disk is only capable of transferring an even number of bytes. If an odd length is given the low order bit is made zero to reduce the length to the lower even value. The length is NOT restricted to the sector size of 512 bytes. If the length is greater than 512, successive blocks are read/written until the required transfer

length has been satisfied. If the length is not an exact multiple of 512 bytes, only the specified length will be read/written. Note that the hardware always reads and writes a complete sector, so specifying a shorter length on a read will cause the remainder of the block to be skipped. On a write, the hardware will repeat the last specified word until the sector is full.

The driver will attempt to recover from all soft errors. There is no automatic write/read verify as on mag tapes, so that data that is incorrectly written will not be detected as such until a read is attempted. For this reason, the verify feature can be used (see above) to force the checking of written data. When an error is detected while performing a read, the offending block will be re-read up to 16 times and disk resets will be attempted during this time too. If all fails a hardware error indication is returned to the user. Other errors possible are Protection Error (attempt to write to a read-only disk) and User Error, which indicates that the parameters in the IORB were incorrect. Errors such as there being no disk loaded, or the drive door being open are NOT detectable by the program. The interface sees these as Seek Errors (i.e. soft errors), and thus the driver will retry several times before returning a Hardware Error indication to the user. It should be noted that error recovery can take a long time. As mentioned above, there is a 2 second delay before a seek error is reported by the hardware, for instance.

NAME

grinnell - colour display

SYNOPSIS

```
DCT grndout
setdct("grndout", 03000, 0172520, 0172522,
        grnoi, grnot, grnoti, &grndin);
DCT grndin
setdct("grndin", 03000, 0172524, 0172526,
        grnoi, grnot, grnoti, &grndout);
```

DESCRIPTION

The Grinnell colour display has a screen of 512x512 pels. Three colours (red, green and blue) can be used, but no grey scale is supported. Three graphics modes are available. These are:

(1) Alphanumeric: The input ASCII characters are displayed at the selected positions on the screen.

(2) Graphic: Basic geometric elements, such as line and rectangle, are drawn by means of graphics commands.

(3) Image: The input data is interpreted as bit patterns, the corresponding images being illustrated.

The values used to construct commands are described in the Grinnell User Manual. They are also listed below.

```
#define LDC     0100000   /* Load Display Channels */

#define LSM     0010000   /* Load Subchannel Mask */
#define   RED   0000010   /* Read Subchannel */
#define   GREEN 0000020   /* Green subchannel */
#define   BLUE  0000040   /* Blue subchannel */

#define WID     0000000   /* Write Image Data */
#define WGD     0020000   /* Write Graphic Data */
#define WAC     0022000   /* Write AlphanumCh */

#define LWM     0024000   /* Load Write Mode */
#define   REVERSE  0200   /* Reverse Background */
#define   ADDITIVE 0100   /* Additive (not Replace) */
#define   ZEROWRITE 040   /* Dark Write */
#define   VECTOR    020   /* Select Vector Graph */
#define   DBLEHITE  010   /* Double Height write */
#define   DBLEWIDTH 004   /* Double Width write */
#define   CURSORAB  002   /* Cursor (La+Lb,Ea+Eb) */
```

```
 #define   CURSORON  001   /* Cursor On */

 #define LUM     0026000   /* Load Update Mode */
 #define   Ec          001   /* Load Ea with Ec */
 #define   Ea_Eb       002   /* Load Ea with Ea + Eb */
 #define   Ea_Ec       003   /* load Ea with Ea + Ec */
 #define   Lc          004   /* Load La with Lc */
 #define   La_Lb       010   /* Load La with La + Lb */
 #define   La_Lc       014   /* Load La with La + Lc */
 #define   SRCL_HOME 020   /* Scroll dsiplay to HOME */
 #define   SRCL_DOWN 040   /* Scroll down one line */
 #define   SCRL_UP   060   /* Scroll up one line */

 #define ERS     0030000   /* Erase */
 #define ERL     0032000   /* Erase Line */
 #define SLU     0034000   /* Special Location Update */
 #define   SCRL_ZAP 0100   /* unlimited scroll speed */

 #define EGW     0036000   /* Execute Graphic Write */
 #define LER     0040000   /* Load Ea relative */
 #define LEA     0044000   /* Load Ea */
 #define LEB     0050000   /* Load Eb */
 #define LEC     0054000   /* Load Ec */
 #define LLR     0060000   /* Load La Relative */
 #define LLA     0064000   /* Load La */
 #define LLB     0070000   /* Load Lb */
 #define LLC     0074000   /* Load Lc */
 #define   LGW     02000   /* perform write */

 #define NOP     0110000   /* No-Operation */

 #define SPD     0120000   /* Select Special Device */
 #define LPA     0130000   /* Load Peripheral Address */
 #define LPR     0140000   /* Load Peripheral Register */
 #define LPD     0150000   /* Load Peripheral Data */
 #define RPD     0160000   /* ReadBack Peripheral Data */
 #define MEMRB     00400   /* SPD - Memory Read-Back */
 #define DATA      01000   /* SPD - Byte Unpacking */
 #define   ALPHA   06000   /* LPR - Alphanumeric data */
 #define   GRAPH   04000   /* LPR - Graphic data */
 #define   IMAGE   02000   /* LPR - Image data */
 #define   LTHENH  01000   /* take lo byte then hi byte */
 #define   DROPBYTE 0400   /* drop last byte */
 #define INTERR    02000   /* SPD - Interrupt Enable */
 #define TEST      04000   /* SPD - Diagnostic Test */
```

The MOS driver is called grin.obj. It operates on the following IORB entries.

iruva

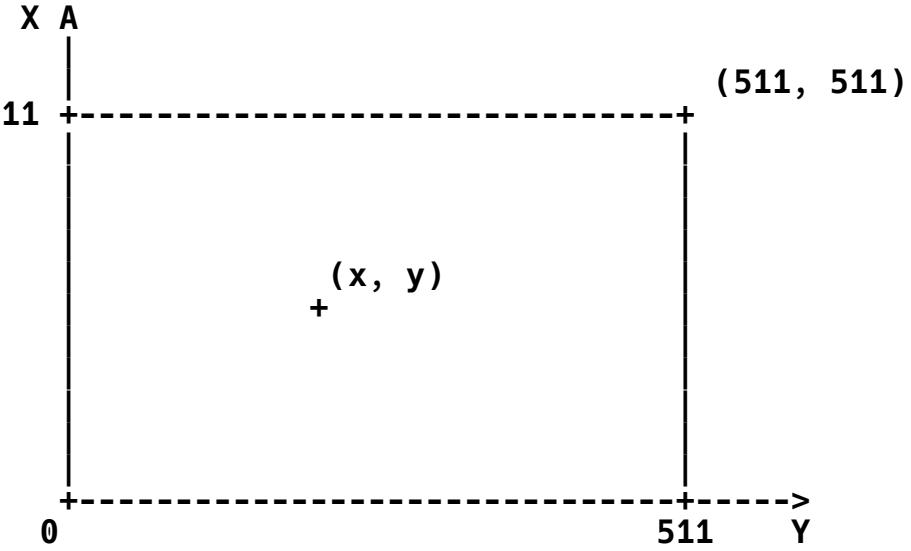
This is a pointer to the buffer where the data is stored.

This data must be ready formtatted for the Grinnell, since no conversion is performed by the driver.

irbr

This transfer length as a positive number of bytes.

Addressing the grinnell. Rows consist of elments numbered 0 to 511 running left to right. The lines are number from 0 to 511 running from bottom to top. It is thus addressed as a conventional X-Y coordinate system. Note that this coordi-
e system is different the one used for the image.



SEE ALSO

grinnell(fax)

NAME

dacom - facsimile machine

SYNOPSIS

```
DCT faxinput
setdct("faxin", 0350, 0174750, 0174740,
        faxii, faxin, faxini, &faxoutput);
DCT faxoutput
setdct("faxout", 0354, 0174752, 0174742,
        faxoi, faxot, faxoti, &faxinput);
```

DESCRIPTION

The DACOM facsimile machine can read a document, creating the corresponding image data blocks. It can also accept the data of relevant format, printing the correponding image.

Each data block consists of 585 bits, and is stored in a block of 74 bytes starting on a byte boundary. The final 7 bits of the last byte are not used and they are undefined. The 585 bits in each block need to be read as a bit stream: the bits in each byte run from the high orger end of the byte to the low order end. The last 12 bits of the 585 bits in each block consistute the CRC field whereby the block can be validated.

There are two kinds of blocks: SETUP blocks and DATA blocks. The first of block of an image data file should be a single SETUP block. All following blocks in the file must be DATA blocks. Note that the second block is a DATA block that contains ZERO samples, i.e. a dummy data blocks. Form the third block, the DATA blocks store the reall image data.

A standard dacom page contains about 1200 scan lines, each of which has 1726 pels. One can choose

Appendix II: Task Controller and Task Processes

NAME

ccitt - conversion between vector and CCITT T4 format

SYNOPSIS

ccitt() - a MOS task

command string (task name is defined as ccitt):
ccitt"<function>

DESCRIPTION

This routine operates as a MOS pipe task to convert the vectors to CCITT T4 format or inversely.

The parameter function specifies what the task is to do.

value	function
1c	one-dimensional compression
1d	one-dimensional decompression
2c[<k>]	two-dimensional compression
2d	two-dimensional decompression

Note k is the maximun number of lines to be coded two-dimensionally before a one-dimensionally coded line is inserted. If k is omitted, the default value 2 is adopted.

SEE ALSO

vector(fax), t4(fax), fitter(fax)

NAME

check - check the validity of a vector file.

SYNOPSIS

check() - a MOS task

command string (the task name is defined as check):
check"<function>,<width>,<height>,[<from>,<to>]

DESCRIPTION

This routine operates as a MOS pipe task checking the validity of the input vector file.

The number of lines to be checked is specified by the parameter height. If the height of the image is less than the parameter, the actual height is printed. Thus, one can set the parameter height to a big number in order to count the number of lines of the input image.

The run lengths in each of these lines are accumulated and the sum is compared with the parameter width.

These are the basic functions which are performed whenever the task is invoked. However, there are several options one can choose by setting the one-character parameter function.

value	function
'n'	basic function only
'c'	print the count of each line
'l'	print all lines
's'	print the lines in the interval specified by parameter from and to

DIAGNOSTICS

A bad line will be reported and it will cause the job aborted.

SEE ALSO

vector(fax), getl(fax), fitter(fax)

NAME

chop - extract a designated rectangular area from an image

SYNOPSIS

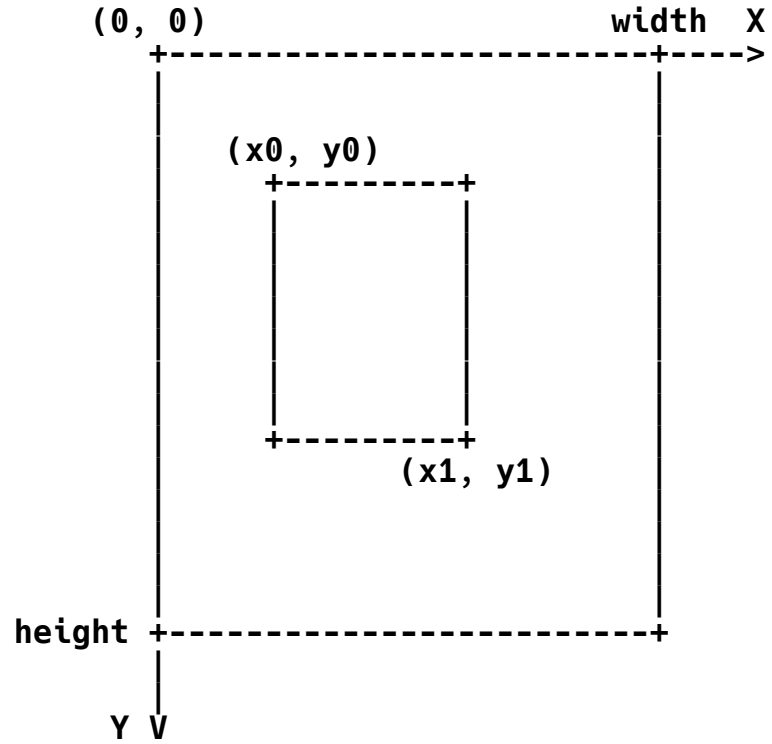
chop() - a MOS task

command string (task name is defined as chop):
chop"<x0>,<y0>,<x1>,<y1>

DESCRIPTION

This routine operates as a MOS pipe task extracting a designated rectangular area from an input image. Input and output are image data files in the form of vectors.

The following diagram shows the coordinate system being used. Note that the lengths are measured in number of pels.



As can be seen in the diagram, the rectangular area to be extracted is specified by the parameters x0, x1, y0, y1, which are decimal strings.

BUGS

One has to make sure that

```
0 < x0 < width
0 < y0 < height
0 < x1 < width
0 < y1 < height
```

SEE ALSO

vector(fax), getl(fax), putl(fax), fitter(fax)

NAME

clean - clean an image.

SYNOPSIS

clean() - a MOS task

command string (task name is defined as clean):
clean"<width>,<height>

DESCRIPTION

This routine operates as a MOS pipe task cleaning an image by means of mask scanning. Input and output are image data files in the form of vectors.

The width and height should be given as the parameters.

SEE ALSO

vector(fax), getl(fax), putl(fax), fitter(fax)

NAME

decomp - decompress DACOM blocks

SYNOPSIS

decomp() - a MOS task

command string (task name is defined as decomp):
decomp

DESCRIPTION

This task takes DACOM blocks from the Clean and Simple interface, and decompresses them into vector format. Then it writes the vectors to the Clean and Simple interface.

SEE ALSO

dacom(dev), vector(fax), fitter(fax)

NAME

fax - interface process for DACOM facsimile machine

SYNOPSIS

fax() - a MOS task

command string (task name is defined as fax):
fax"<function>

DESCRIPTION

This task uses the Clean and Simple interface to read or write facsimile image data.

The one character parameter function specifies whether the data is to be read or written. Character w is for writing. In this case, 74 byte DACOM blocks containg correct CRC fields are expected. On the other hand, character r is for reading. In this case, a document is read on the facsimile machine, the DACOM blocks being created.

SEE ALSO

dacom(dev), fitter(fax)

NAME

fitter - fit processes together to form a data pipe

SYNOPSIS

fitter() - the MOS task controller

DESCRIPTION

According to the command string typed on the console, fitter links the specified processes together to form a task chain. The name of the processes is the name given in the PCB. The processes must communicate using the C+S interface. Only one C+S interface is opened per process - data is pushed in with a cswrite and pulled out with a csread. The fitter does not inspect the data in any way but merely passes it from one process to another.

The format of command string is:

```
A | B | C.
```

The fitter takes data from the process called A, write it to the process called B, reads data from the process B and write that data to the process C. Note that all middle processes are both read and written, while the first one in the list is only read from and the last in the list is only written to.

A double quote is used as the separator between the task name and the open parameter string, e.g.

```
A"500 | B"n,xyz | C,
```

where the strings '500' and 'n,xyz' are the open parameter stings for tasks A and B, respectively. The parameter stirng is passed to the corresponding task routine when the csopen call returns.

DIAGNOSTICS

The command string containing undefined task will be rejected.

SEE ALSO

csinit(fax), csopen(fax), csread(fax), cswrite(fax)

NAME

fs - file system for use under MOS

SYNOPSIS

fs() - a MOS task

command string (task name is defined as fs):
fs"<funciton>,<file_name>

DESCRIPTION

This is a file system, based on the Double Density floppy disk, for use under MOS. The fs task is used for manipulate the files, managed by the file system. This task can only appear at the first or last position on a command string. In the former case, the file specified is to be read, while the file is to be written in the latter case.

The <function> field contains only one character indicating the function to be performed. The possible values are:

e - open an existing file (for reading).
c - open an existing file, and set the length to zero (for rewriting).
a - append to an existing file.

If the capitals A, C, and E are used, the functions are the same as described above but the specified file is created if it does not exist.

BUGS

This task is for reading and writing only. As for the other facilities, e.g. seek, delete, status and sync, one has to use C+S interface directly.

Note that only 15 files are permitted per disk, only drive 0 is supported at present, and no hierarchical directory is allowed.

SEE ALSO

aed62(dev), fitter(fax)

NAME

ftp, pftp - NIFTP task processes

SYNOPSIS

ftp(), pftp() - MOS tasks

command string (task name is defined as ftp):
ftp"<function>,<code>,<user_name>,<password>,<file_name>;
<trasport_service_process>:<transport_service_parameters>

DESCRIPTION

These tasks are implementation of Network Independent File Transfer Protocol (NIFTP) for LSIs under MOS. They employ a transport service for communication with a remote host on the network, where the same protocol must be supported. They communicate with the user process and transport service processes thourgh the Clean and Simple interface, so that they can be used in a fitter command chain directly.

The code is available in two versions: ftp which is a P+Q version supporting both server and intitiator and pftp which is a P version working only as an initiator. Both of them are capable of sending and receiving.

This implementation of NIFTP is just a subset of the protocol as its main purpose is to provided the facsimile system with a data transmission mechanism. For the sake of simplicity, only the necessary facilities are included in the module, while more complex facilities, such as data compression and error recovery are not implemented. The following table shows the transfer control parameters being used.

Attribute	Value	Mod.	Remarks
Mode of access	0001	EQ	Creating a new file
	8002	EQ	Retrieving file
Codes	-	-	Text file, any parity
	1002	EQ	Binary file
Format effector	0000	EQ	No interpretation
Binary mapping	0008	EQ	Default byte size
Max record size	00FC	EQ	Default record size
Transfer size	0400	LE	Default transfer size
Facilities	0000	EQ	Minimum service

The meanings of the parameters in the command string are listed below:

function is the NIFTP function of our site. Any ASCII string beginning

beginning with 't' means the file is to be transmitted to the remote site. Otherwise, the file will be retrieved from the remote site.

code specifies the type of the file to be transferred. Any ASCII string beginning with 'b' means it is a binary file, while others mean text file.

user_name is the login name of the server site.

password is the password of the server site.

file_name is the name of the file to be transmitted.

transport_service_process is the process name of the transport service to be used.

transport_service_parameters are the parameter string required by the transport service. They are network dependent and specified by the corresponding transport service.

SEE ALSO

fitter(fax)

NAME

grinnell - task to convert and display fax vector data

SYNOPSIS

grinnell() - a MOS task

command string (task name is defined as string):
grinnell"<x0>,<y0>,<x1>,<y1>,<mode>,<colour>

DESCRIPTION

This task takes the vector data from a Clean and Simple interface and displays it on the Grinnell screen. The Grinnell screen is viewed as an X-Y plane with (0,0) being the lower left hand corner, (512, 0) being the lower right hand corner, etc.

The parameters x0, y0, x1, y1 are decimal strings defining the rectangular space on the screen where the image is to be displayed. If the image is smaller than this area, it is artificially expanded to the size of this area. If the image is larger than this area it is truncated to the size of the area.

The colour field consists of any combination of the characters r,g or b to define the colours red, green and blue respectively. For instance "gb" would write the image as yellow.

The mode defines how the image is to be displayed. Any combination of the characters r,a and z may be used, to the following effect:

r = reverse image
a = additive image
z = zerowrite image.

There are three bit planes to define the three colours. Normally the bit planes corresponding to the selected colours have either zero bits or one bits written to them depending upon whether the image or the background is being written. For zerowrite, all non-selected bit planes (i.e. colours) are always set to zero, thus erasing any unselected colours in the area. Additive mode means that in the selected colour planes the new bits are ORed in, rather than just written. Thus the image is added to. In reverse mode, the image written as one bits is written as zero bits and the bits written as zero bits are written as one bits, i.e. the bits are flipped before being used.

SEE ALSO

grinnell(dev), vector(fax), fitter(fax)

NAME

merge - merge two images together

SYNOPSIS

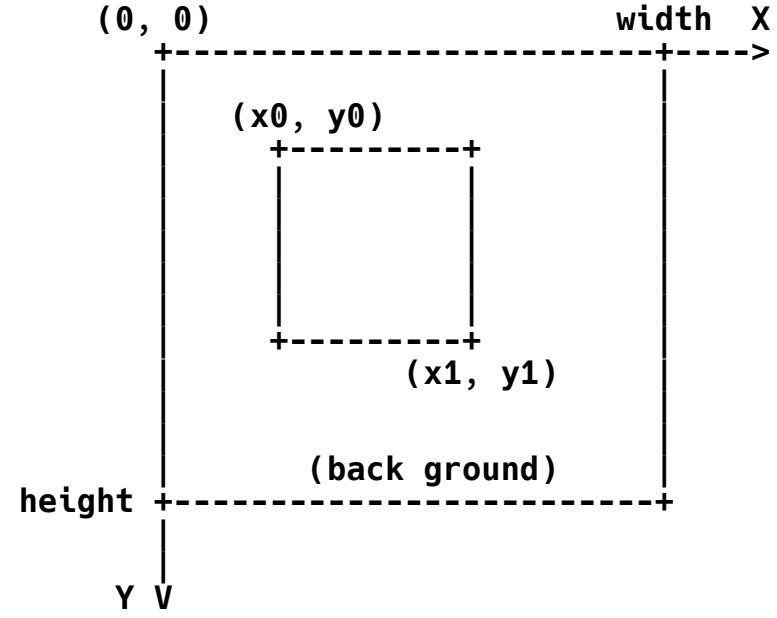
merge() - a MOS task

command string (task name is defined as merge):
merge"<file_name>,<action>,<x0>,<y0>,<x1>,<y1>

DESCRIPTION

This routine operates as a MOS pipe task merging two images together to form the result image. Input and output are image data files in the form of vectors.

One of the two input images is called background which is to be copied directly. This is specified by the parameter file_name. The image data of the back ground is read via a 'tunnel', maintained by this task. Another input image is taken form the Clean and Simple interface managed by the fitter. As shown in the following diagram, the position where it is to be put on the background image is specified by the parameters x0, y0, x1, y1, which are decimal strings. This implies that the dimension of the image is x1 - x0 and y1 -y0.



The parameter action indicates how the two images are merged. If it set to 0, The second image is simply overlaid on the back ground image. On the other hand any non-zero value

causes the second image to replace the specified area of the back ground image.

BUGS

One has to make sure that

```
0 < x0 < width_of_back_ground
0 < y0 < height_of_back_ground
0 < x1 < width_of_back_ground
0 < y1 < height_of_back_ground
```

In addition, x0, y0, x1, y1 must be consistent with the dimension of the image

SEE ALSO

vector(fax), getl(fax), putl(fax), chop(fax), fitter(fax)

NAME

od - dump the input data

SYNOPSIS

od() - a MOS task

command string (task name is defined as od):
od"<format>

DESCRIPTION

This routine operates as a MOS pipe task dumping the input data in a selected format. The input data is taken from the Clean and Simple interface.

The meanings of the one character parameter format are:

value	format
'd'	words in decimal
'o'	words in octal
'c'	bytes in ASCII
'b'	bytes in octal

SEE ALSO

fitter(fax)

NAME

recomp - compress the vectors to form the DACOM blocks

SYNOPSIS

recomp() - a MOS task

command string (task name is defined as recomp):
recomp

DESCRIPTION

This task takes vectors from the Clean and Simple interface, and recompresses them into DACOM blocks. Then it writes the blocks to the Clean and Simple interface.

SEE ALSO

dacom(dev), vector(fax), fitter(fax)

NAME

scale - scale an image to a specified dimension

SYNOPSIS

scale() - a MOS task

command string (task name is defined as scale):
scale"<old_width>,<old_height>,<new_width>,<new_height>

DESCRIPTION

This routine operates as a MOS pipe task scaling the input image to the specified dimension. Input and output are image data files in the form of vectors.

The dimension of the input image is given by the parameters old_width and old_height, while the dimension of the output is specified by the parameters new_width and new_height.

SEE ALSO

vector(fax), getl(fax), putl(fax), fitter(fax)

NAME

string - convert an ASCII string to the vector format

SYNOPSIS

string() - a MOS task

command string (task name is defined as string):
string"<s>

DESCRIPTION

This routine operates as a MOS pipe task converting the parameter string s to the corresponding vectors.

SEE ALSO

vector(fax), ts(fax)

NAME

tf - convert a text to the vector format.

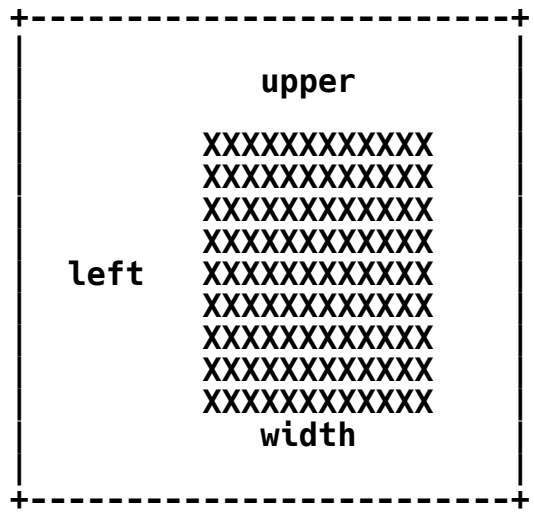
SYNOPSIS

tf() - a MOS task

command string (task name is defined as tf):
tf"<width>,<line_sp>,<upper>,<left>

DESCRIPTION

This routine operates as a MOS pipe task converting the input text to the corresponding vectors. The input text, taken from the Clean and Simple interface should be in the format defined in text(fax).



As shown in the diagram, the parameters give the information for the formating. The parameter width is the maximum width of the text lines.

Every vector will be padded to fit this width. White pels may be padded to the left of each vectors, and the number of pel to be padded is specified by the parameter left.

Empty lines may also be inserted. They are defined by parameters upper and line_sp, the number of pels being used as the unit.

SEE ALSO

vector(fax), text(fax), ts(fax), fitter(fax)

Appendix III: Utility Routines and Data Formats

NAME

bitmap - convert vector format to core bit map

SYNOPSIS

```
int  bitmap(ivec, cnt, buff);

int  *ivec;
int  cnt;
char *buff;
```

DESCRIPTION

Bitmap converts the fax vector format into a bit map, using each bit of the area pointed to by buff. The number of elements in ivec is given by cnt, and the first element of ivec is taken as a white pel count, the second as a black pel count, etc. The resultant bit map is placed in the area pointed to by buff. The actual number of bits stored is returned from the function. The bits in buff are stored in byte order, with the highest value bit of the byte taken as the first bit of the byte.

BUGS

You have to make sure that buff is big enough for all the bits.

SEE ALSO

vector(fax), tovec(fax)

NAME

tovec - convert bitmap to vector format

SYNOPSIS

```
int  *tovec(buff, nbits);

char *buff;
int  nbits;
```

DESCRIPTION

The bitmap in the buffer pointed to by buff is converted to vector format. The length of the bitmap in bits is passed in nbits. As the caller would normally not know how many vector elements are going to be needed, the tovec routine allocates this area for the user.

Buff is assumed to be organised in byte order with the highest value bit of each byte being the first bit of the byte. The counts of white and black pels are placed into an integer vector, the first element of which is the length of the rest of the vector. The vector information proper starts in the second element which is the count of the number of leading white pels. This is followed by the count of the numbr of black pels, etc.

The routine goes to great lengths to make sure only enough vector storage is allocated. Temporary storage is allocated in small chunks and then, when the length of the whole vector is known, the chunks are contacenated into a contiguous vector. The pointer to this vector is returned to the user.

SEE ALSO

vector(fax), bitmap(fax)

NAME

choice - specify a rectangular area on Grinnell

SYNOPSIS

```
struct  square  {
        int  x0, y0;
        int  x1, y1;
};
struct  square  *choice(colour, height, width, area, fw, fh)

char colour;
int  height, width, area, fw, fh;
```

DESCRIPTION

This subroutine is called by a MOS task. to specify a rectangular area of an image by manipulating a square on the Grinnel display being illustrating the image. The dimension of the original image is defined as height and width. The area on which the original image is shown is specified by the parameter area.

value	area	dimension	coordinates
0	the whole screen	512x512	0,511,511,0
1	the left half	256x512	0,511,255,0
2	the right half	256x512	256,511,511,0

The square will be drwan in a colour defined by the parameter colour, which can only be:

value	colour
'r'	red
'g'	green
'b'	blue

There are two modes being supported:

(1) Fixed: The square will have a fixed dimension specified by the parameters fw and fh. The operator can move the square around as a whole within the predetermined area by using following commands, each of which is invoked by typing the corresponding characer on the keyboard of the system console.

command	function
'u'	move the square up one step
'd'	move the square down one step
'l'	move the square one step left
'r'	move the square one step right
'f'	move fast - set the step to 8 pel
'o'	move slowly - set the step to 1 pel
<CR>	ok - the area has been chosen, and return its coordinates

(2) Arbitrary: This mode is set up when the subroutine is called with the parameters fw and fh set to 0. Any edge of the square can be selected to be moved on its own by using the same commands described above. The following commands are required to select the relevant edge as well as switching the operation mode.

command	function
'e'	select the right ('east') edge.
'w'	select the left ('west') edge.
'n'	select the upper ('north') edge.
's'	select the lower ('south') edge.
'a'	move the square as a whole

As soon as the user types <CR>, the coordinates of the current square, which are accommodated in a square structure, are returned. Note these are concerned with the coordinate system defined for the image but not for the grinnell.

BUGS

Currently, only three working areas can be used.

SEE ALSO

vector(fax), grinnell(dev), grinnell(fax)

NAME

crc - calculate or check the DACOM CRC code

SYNOPSIS

```
int  crc(buff, insert);

char *buff;
int  insert;
```

DESCRIPTION

This routine will check/insert the 12-bit CRC code for a DACOM block, pointed to by buff. The block contains 585 bits, the last 12 bits being the CRC code. The block is checked only when the parameter insert is set to 0, otherwise the CRC code is created and inserted into the block. When the block is checked, the routine returns the result: 0 means OK and any non-zero value means the block is bad. On the other hand, when the CRC code is inserted, the routine returns the CRC code it has created.

This routine uses a tabular approach to determine the CRC code, processing a whole byte at a time and resulting in a high throughput.

BUGS

Do not forget to supply enough space when the 12-bit CRC code is to be inserted.

SEE ALSO

dacom(dev)

NAME

csinit - initiate the Clean and Simple interface

SYNOPSIS

```
int  csinit();
```

DESCRIPTION

This routine is called to initiate the Clean and Simple interface for the calling process. Its code is re-entrant, so that only one copy is needed for all processes in a system.

This routine returns the task identifier, which must be used on all subsequent interface calls.

SEE ALSO

csopen(fax), csread(fax), cswrite(fax), fitter(fax)

NAME

csopen - establish the Clean and Simple connection

SYNOPSIS

```
char *csopen(tid);

int  tid;
```

DESCRIPTION

A process calls this routine, waiting to be scheduled. Its code is re-entrant, so that only one copy is needed for all processes in a system.

The task identifier tid is the word returned from the csinit call. When the fitter process has established the Clean and Simple connection for the process, this routine returns the pointer to the parameter string of the corresponding task command.

SEE ALSO

csinit(fax), csread(fax), cswrite(fax), fitter(fax)

NAME

csread - read data from the Clean and Simple interface

SYNOPSIS

```
char *csread(tid, need);

int  tid, need;
```

DESCRIPTION

This routine is called to read data from the Clean and Simple interface. Its code is re-entrant, so that only one copy is needed for all processes in a system.

The task identifier tid is the word returned from the csinit call. The need parameter indicates the number of bytes that are required. This routine returns a pointer to a buffer with this much data in it. This is usually more efficient as it means that the data does not have to be reblocked.

DIAGNOSTICS

If the returned value is 0, the end of data is reached.

BUGS

Funnies happen at the end of data to be read. The csread() call has no way of saying that the final buffer is partly filled. Thus if you ask for more data, you hang forever. But if the data structures are working correctly, this should never happen.

SEE ALSO

csinit(fax), cswrite(fax), fitter(fax)

NAME

cswrite - write data to the Clean and Simple interface

SYNOPSIS

```
char *cswrite(tid, need);

int  tid, need;
```

DESCRIPTION

This routine is call to write data to the Clean and Simple interface. Its code is re-entrant, so that only one copy is needed for all processes in a system.

The task identifier tid is the word returned from the csinit call. The need parameter indicates the number of bytes that are to be written. This routine returns a write buffer of the required length, to which the user data can be copied. The subsequent cswrite() call automatically releases the previous write buffer.

The cswrite() call with need set to 0 indicates the end of data, closing the current Clean and Simple connection.

BUGS

As indicated, the write buffer must be filled up before the next cswrite() call.

SEE ALSO

csinit(fax), csread(fax), fitter(fax)

NAME

getl - get a line vector from the Clean and Simple interface

SYNOPSIS

```
int  *getl(tid);

int  tid, need;
```

DESCRIPTION

This routine is called to read a line vector from the Clean and Simple interface. Its code is re-entrant, so that only one copy is needed for all processes in a system.

The task identifier tid is the word returned from the csinit call. The routine returns the pointer to the buffer where the line vector is stored.

DIAGNOSTICS

0 will be returned when end of file is reached.

BUGS

Any memory violation causes the whole task chain to be aborted.

SEE ALSO

vector(fax), putl(fax), fitter(fax)

NAME

putl - put a line vector to the Clean and Simple Interface

SYNOPSIS

```
putl(tid, buf);

int  tid, *buf;
```

DESCRIPTION

This routine is called to write a line vector to the Clean and Simple interface. Its code is re-entrant, so that only one copy is needed for all processes in a system.

The task identifier tid is the word returned from the csinit call. The line vector is stored in a buffer pointed by buf.

SEE ALSO

vector(fax), getl(fax), fitter(fax)

NAME

t4 - the data format defined in CCITT recommendation T4

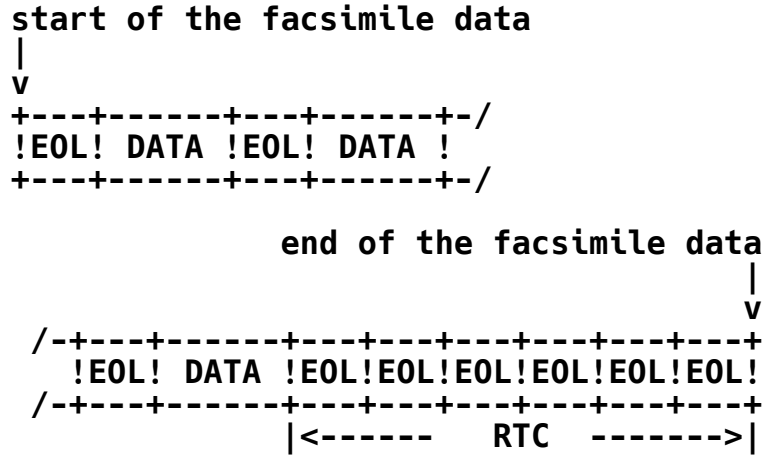
DESCRIPTION

Dimension and Resolution: In vertical direction the resolution is defined below.

Standard resolution:	3.85 line/mm
Optional higher resolution:	7.70 line/mm

In horizontal direction, the standard resolution is defined as 1728 black and white picture elements along the standard line length of 215 mm. Optionally, there can be 2048 or 2432 picture elements along a scan line length of 255 or 303 mm, respectively. The input documents up to a minimum of ISO A4 size should be accepted.

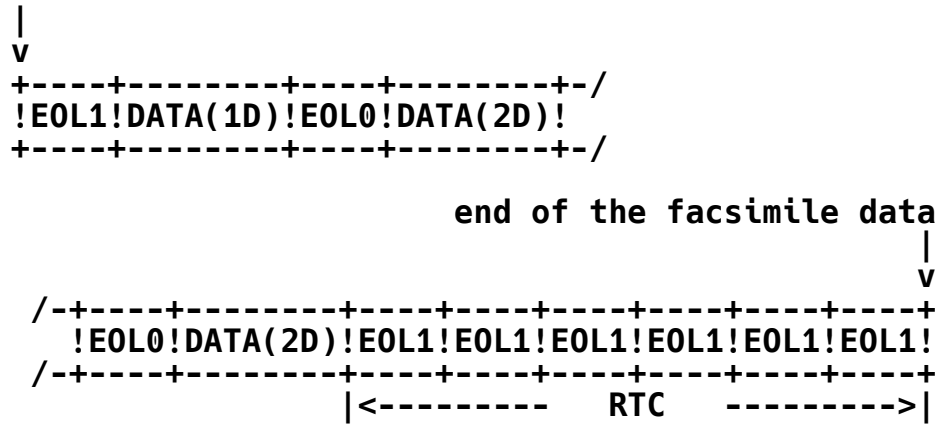
One-Dimensional Coding: The one-dimensional run length data compression is accomplished by the popular modified Huffman coding scheme. In this scheme, black and white runs are replaced by a base 64 codes representation. Compression is achieved since the code word lengths are invertly related to the probability of the occurrence of a particular run. A special code (000000000001), known as EOL (End of Line), follows each line of data. This code starts the facsimile message phase, while the control phase is restored by a combination of six contiguous EOLs (RTC). The data format of a facsimile message is shown below.



Two-Dimensional Coding: The two-dimensional coding scheme is labeled as the Modified READ Code. It codes one line with reference to the line above,correlation between adjacent lines allowing for more efficient compression. In order to limit the disturbed area in the event of transmission errors,

a one-dimensionally coded line is transmitted after one or more two-dimensionally coded lines. A bit, following the EOL, indicates whether one- or two-dimensional coding is used for the next line:

EOL1: one-dimensional coding;
EOL0: two-dimensional coding.



NAME

text - the text format for use in the facsimile system

DESCRIPTION

This is the representation structure for coded character text. It is used in the facsimile system.

The text structure consists of a series of character strings, each of which represents a text line. However no control characters, e.g. <CR> and <LF>, are used in the structure. Each text line is proeeded by a count byte, indicating the number of characters on the line. The character sting follows after the the count byte. A zero count indicates the end of file.

EXAMPLES

Here is an example text shown below:

```
        This is a text.
        This is a picture.
```

It can be represented as:

```
 <017> T  h  i  s <040> i  s <040> a <040> t  e  x  t  .
 <022> T  h  i  s <040> i  s <040> a <040> p  i  c  t  u
 r e  . <0>
```

NAME

ts - translate an ASCII string into vector format

SYNOPSIS

```
ts(ar_in, left, right, tid)

char *ar_in;
int  left, right, tid;
```

DESCRIPTION

This routine will convert a zero-ended ASCII string pointed to by ar_in into the corresponding vecter format. As the character font being used is a set of 12x20 matrices, there will be 20 line vectors created. These vectors are written to the Cleans and Simple interface by calling cswrite. The callers task identifier tid has to be provided.

At the two ends of the text line, blanks can be padded that are specified as left and right. Note that they are measured in pels.

Consequently, the result should be a image, whose dimension is:

```
width  = left + 12*length + right;
height = 20;
```

where length is the number of characters in the input string.

As an intermediate result the bitmap is first created which is then converted into the vector format, by calling tovec.

BUGS

The input string must be ended with a zero field.

SEE ALSO

vector(fax), tovec(fax), csinit(fax), cswrite(fax), fitter(fax)

NAME

vector - the internal data structure for a facsimile image

DESCRIPTION

This is the representation structure for binary images, a simple run length compression algorithm being used. Most of the image files are kept in vector format for ease of processing.

The vector format consists of a series of integer vectors, one vector for each row of pels in the image. Each vector is proceeded by a count word which indicates the number of integer words in the vector. The next element of the vector after the count field is the number of white pels in the first run of the line. The second word then gives the number of pels that follow the initial white run, and so on t the end of the vector. Note the first run length element must refer to a white run. It should be set to 0 if the first run is black.

EXAMPLES

A line consists of 20 pels as follows:

00011111111011100000

It can be represented as:

5, 3, 8, 1, 3, 5

The inverse of the line:

11100000000100011111

should be represented as:

6, 0, 3, 8, 1, 3, 5