IMPRINT-10
System Manual

August 1983
System Version 1.8

IMAGEN Corporation

PRELIMINARY

IMAGEN Corporation
2660 Marine Way
Mountain View, California 94304
(415) 960-0714

Chris Ryland is the principal author of this manual; Eduardo Martinez, Jan Stoeckenius and Julia Tien wrote portions of the Impress chapter.

This manual was created and edited with Gosling Emacs under Berkeley VAX/Unix, composed to final form from the source /rd/imp/doc/imp.mss by Scribe (version 3C(1300)) on August 23, 1983, and printed on the IMPRINT-10. Your comments are more than welcome and should be forwarded to IMAGEN at the above address with an "Attention: Documentation Department."

Table of Contents

CHAPTER 1
Introduction

Overview (1.1)

The IMPRINT-10 is a medium-speed, low-cost, high-capability electronic printing system. Its major features are:

- process: Canon NP, using ordinary Canon copier smooth-finish paper (available from multiple sources)
- resolution: 240 dots per inch horizontally and vertically
- speed: 9.8 letter size pages per minute, with printing fully decoupled from virtual page image production to drive the LBP-10 at this rated speed
- proven reliability: conservatively 6-12 months between failures, based on experience with over 100 systems
- paper handling flexibility: 13 international paper sizes
- paper handling reliability: will not lose output because of paper problems (such as paper jams)
- copy handling flexibility: multiple copies, collation, reversal
- document user support: records significant events, as well as problems encountered during document processing on a set of optional job "banner" pages
- applications flexibility: emulation of various devices, including line printer, daisy-wheel, Tektronix 4014, pen plotters
- virtual page orientation flexibility: any page or part of a page can be laid out in any of four orientations
- typesetting capability via the Impress language
- graphics capability via the Impress language and other device emulators: polygon drawing, region filling with user-definable pen sizes and textures
- high-resolution screen-dump capability via the Impress language general "bit map" facility, with magnifications 1, 2, and 4, in all four page orientations
- host interface flexibility: serial (byte stream XON/XOFF and CTS-toggle flow control, error correcting/detecting protocols), parallel (Centronics, Dataproducts), IBM binary synchronous RJE (2780/3780), and Ethernet (TCP/IP)
- printing system control via an optional operator's console
- printer-resident, orientation-independent, compactly-encoded fonts (up to several hundred)
- compatibility with future, higher- and lower-speed IMAGEN printing engines via image processor modularity

The rest of this chapter provides a brief introduction to the remaining chapters.

Part I: The System (1.2)

The IMPRINT-10 is capable of executing documents written in a variety of forms; each such form is called a *document language*. For example, the document language named "Printer" is a stream of characters intended for an ASCII character or line printer. The Chapters in Part I describe the document-language-independent aspects of the IMPRINT-10.

Day-to-day operational procedures, as well as periodic maintenance tasks, are covered in Chapter 2.

Each document can contain a document header which specifies, in a fashion independent of the document contents, parameters needed for document processing, such as the language in which the document's body is written. See Chapter 3 for more information.

The IMPRINT-10 supports, but does not require, an operator's console terminal, useful for viewing and controlling system operation. For example, the operator can cancel a job, temporarily hold page printing or job processing, find out about the state of communications, etc. Chapter 4 documents the console command interface.

The IMPRINT-10 supports various paper sizes. It also provides for multiple document copies, with collation and reversal. See Chapter 5 for more details on paper handling.

Documents are presented over some communication interface, one at a time, by a host driving the IMPRINT-10, and are normally printed in the order they arrive. Otherwise, printing is not tied to document communication. For the communications interfaces which provide job and printer status to the host, the host can watch job progress, and wait for each document to finish printing before sending the next. See Chapter 6 for more information concerning communications.

The IMPRINT-10 can print a set of *job header* pages following the document, containing identifying data, such as document control information, as well as complaints and comments produced by the system during document processing. See Chapter 7 for more information about job header messages.

The IMPRINT-10 can be configured to match its operating environment. For example, the console and host (if applicable) serial line speeds can be set, the default document language can be chosen, etc. See Chapter 8 for more detail.

Part II: Document Languages (1.3)

As mentioned above, the IMPRINT-10 is capable of interpreting documents in a variety of document languages. Part II of this manual documents these languages.

Chapter 9 describes the Impress language, the full native language of the IMPRINT-10, which provides typesetting capabilities along with high-level graphics primitives, as well as bit-raster images. It is through this language that the document composition systems TEX, TROFF, Scribe, and Script drive the IMPRINT-10.

Chapter 10 describes the Printer language, a generic ASCII printer emulation providing orientation flexibility ("landscape" and "portrait" layouts), and up to two logical forms per physical page.

Chapter 11 documents the Daisy language, an emulation of the popular "daisy wheel" printers.

Chapter 12 describes the Tektronix language, an emulation of the Tektronix 4014 display terminal.

CHAPTER 2
Basic Operation

The next few sections will introduce you to the IMPRINT-10's daily operation. They cover starting the machine up, basic operation of the console, replacing the printer's supplies, and a few simple maintenance items. You will quickly discover that the IMPRINT-10 doesn't need much attention; operating it is very simple, and problems are few.

Anyone who will have direct contact with the printer should read the next few sections; if everyone can free a paper jam, nobody will have to wait when the inevitable jam occasionally occurs. Likewise, everyone should know how to add toner and pre-mix. These operations are all simple and take very little time.

In normal operation, the printer should need little maintenance. Should you want to touch up the printer's tuning, we have provided directions for tuning in the last part of this chapter. However, even without frequent tuning the IMPRINT-10 will produce 30 to 40 thousand pages of quality printing with no problems.

Starting up (2.1)

Power (2.1.1)

The printer's power switch *(2.1.1.1)*

The IMPRINT-10 has three power switches. The printer's power switch is in the upper left-hand corner of the cabinet. This turns off the printer's electronic and mechanical parts. Turn the printer off at night or when it is not being used much; this will save toner and pre-mix.

When you turn this switch on, the printer will take 15 to 75 seconds to warm up. During this period, the light labelled READY/WAIT will flash. When the printer has warmed up, the light will stop flashing and will glow steadily. At this point, check the SERVICE CALL DIAGNOSIS lights on the rights side of the cabinet. If one of these is flashing, turn the power off, wait a few seconds to allow the printer's moving parts to stop, and turn the printer on again. If the SERVICE CALL DIAGNOSIS lights continue to flash after the printer has warmed up again, call your service representative.

Opening one of the doors on the printer has the same effect as turning off this switch. Closing the door will automatically turn the the printer back on; but it will have to warm up again before it is ready to print. Don't open any doors while the printer is in the middle of printing a page; this could cause the paper to jam.

Don't be surprised if the console prints messages saying that it is processing documents while the

printer is warming up, or even while it is turned off; the processor has its own power switch, and will work as long as it is turned on. These jobs will be printed as soon as the printer is ready for them.

The master power switch *(2.1.1.2)*

The master power switch, at the bottom of the printer on the front, must be left on at all times; turning it off will allow the drum to degrade and hurt the quality of your printing. Only use this switch in case of an emergency.

The processor's power switch *(2.1.1.3)*

The processor power switch is on the back of the processor box; this is the small box attached to the right side of the printer. As long as this switch is turned on, the system will be able to process new documents, whether or not the printer is ready. It can store up to 250 pages without printing them (not counting multiple copies of the same page); when this storage fills up, the IMPRINT-10 will refuse to accept new jobs from your host system. Since the processor uses very little power, and since the processor can do useful work without the printer, there is really no reason to turn this switch off.

Initial dialogue (2.1.2)

Whenever you turn the processor on, it will print the following set of messages on the operator's console:

```
Ports: console: A at console speed baud, host: B at host speed baud
DDT68 (V version date) (C) 1983 IMAGEN Corp
Type any character to stop auto-boot...booting
File system: ..................... [i proms]
Loading IMP: ................
Loading IMP.sym: not found
Onboard nk, offboard mk
Starting
Communications configuration messages
 -- IMPRINT-10 Printing System vmajor.minor --
IMP>
```

When the message `Type any character to stop auto-boot` appears, the system will give you five seconds to type something before proceeding. Don't do anything; this feature is provided only for IMAGEN customer support. For most purposes, you can ignore the entire start-up sequence; if you need more information about it, see subsection 4.7. When the message `IMP>` appears on the console, the system will begin operating normally.

If you follow our advice and leave the processor running continuously, you will never see this group of messages.

The console (2.1.3)

The console terminal gives you a running account of what the printer is doing. While we highly recommend that you use a console terminal, it is not absolutely necessary. Most of the messages it prints are self-explanatory; if you need more information about anything, see section 4.2. The most common messages will tell you that the printer has "compiled" a page (i.e. it has prepared a page for printing) or that it has actually printed a page. Compilation is usually faster than printing, so don't be surprised if the printer has compiled page 30 before it has printed page 5. Other common messages will tell you that the printer is starting a new job, that it is printing the job header (a sheet identifying the job and listing any errors in it), or that it has finished a job. The

first of these messages will identify the document's *language*, *name*, *owner*, the number of *copies* requested, and the type of *paper* requested. This last item is worth noting, since the IMPRINT-10 will print the document regardless of the type of paper you have loaded. Therefore, if a document requests some other type of paper, the operator must stop printing (using the HOLD command explained below) and change paper cassettes.

The console is always waiting for commands from the operator. Typing an carriage-return at any point will produce a *prompt*, IMP>, on the console. This shows that the printer is ready for a command. Remember, however, that you don't need to give any commands for the printer to print. It will do just fine by itself. For more information about the prompt, see section 4.3.1.

To issue any command, simply type the first letter of each part of its name. for example, for the `Information (about) Communications` command, type "`i`" "`c`" in response to the prompt. The console will immediately print the rest of the command and some additional message telling you what it is doing. If you change your mind after you have started a command but before you have finished it, type "control U" by holding the `control` key and typing `u`. The printer will abandon the command and give you a new prompt on the console. If you are unsure of what to do at any point, type a question mark; the printer will then list your options on the console. The most important commands are `Information`, `Hold`, and `Release`; they have the following function:

`Hold`

The `Hold` command has two forms: `Hold jobs` and `Hold page-printing`. The first will temporarily stop the printer from processing jobs after the current job is finished; the second will stop the printer after it finishes the current page. These commands are most useful when you need to change the type of paper you are using. For `Hold jobs`, type "`h`" "`j`" on the console; for `Hold page-printing`, type "`h`" "`p`."

`Release`

The `Release` command similarly has two forms: `Release jobs` and `Release page-printing`, for which you type "`r`" "`j`" and "`r`" "`p`" respectively. The release command lets the printer start again after a `Hold`; for example, if a document requires a different kind of paper, you would type "`h`" "`j`," wait for the printer to stop the current job, change the paper cassette, then type "`r`" "`j`" to resume.

`Information`

The `Information` command will give you additional information about various things the system is doing. The most useful of these is `Information (about) jobs` (type "`i`" "`j`"); when you type this command, the printer will give you a report that includes whether or not the current job is being held, what page and copy is being printed, the total number of pages and copies requested, the job's name, owner, and the number of jobs that are waiting to be printed. The display is simple and self-explanatory. The other important information command is `Information (about) LBP-10` (type "`i`" "`l`"). This command gets you a report that includes whether or not the printer is ready; what problems need to be corrected before the printer will be ready; and what kind of paper has been loaded. Again, the display is self-explanatory. To correct any problems that the printer reports, see section 2.2.

There are several other commands which are not described here. They can be found in sections 4.3.2 and 4.4.

Daily operation (2.2)

Paper (2.2.1)

The IMPRINT-10 uses Canon NP hard-rolled paper. For the best results, take care to store this paper properly. It should be kept in a dry, cool, preferably dark place. Don't open more than one package at a time; the paper's moisture-proof wrapper helps to keep it from aging.

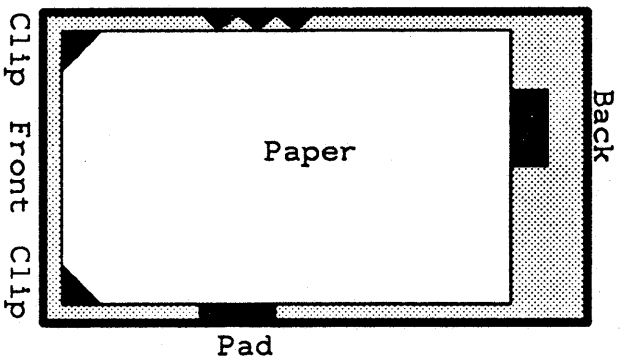
Adding paper *(2.2.1.1)*

Whenever the printer runs out of paper, it will print the message

```
--Attention--
LBP-10 not ready: no paper
```

When this happens, remove the empty paper cassette from the printer by pulling it out firmly (without forcing it). The paper cassette is located underneath the tray that holds the printer's output. Replace the cassette by sliding a new one in where the old one was; it goes as close to the front of the machine as possible. Again, push firmly, but do not force. After putting the new cassette in place, the printer will start printing again.

To fill the empty cassette with paper, first remove its plastic lid. This lid snaps over the top of the cassette. Then take about an inch of paper from a new package, "flex" the stack of paper in both directions to prevent it from getting stuck, and place it neatly in the tray with the "top," as marked on the package, facing up. Pushing back the small spring-loaded pad on the side of the tray will let you slip the paper in easily. At this point, press firmly on the two corners of the stack at the "front" of the cassette (the end that goes into the printer). This will force the paper under the two metal clips that hold it in place. When you load a cassette, be careful not to wrinkle or crimp the paper; wrinkled paper can make the printer jam. The diagram below shows a typical paper cassette:



A typical cassette

We recommend that you load an entire ream of paper at once into several cassettes (this should take three cassettes). This will make it easy to change paper quickly; you will also be less likely to load paper upside down if you load an entire package at once. Keep the cassette's hinged cover closed when the cassette is not in use; this helps protect the paper from moisture. Use only paper supplied by IMAGEN.

Freeing paper jams *(2.2.1.2)*

The LBP-10 printer rarely experiences paper jams under normal conditions. However, they will occur from time to time, particularly if you use paper that hasn't been approved by IMAGEN or if you wrinkle the paper while loading it into the cassette. When there is a paper jam, the printer will display the message

```
--Attention--
LBP-10 not ready: paper jam
```

on the console. Take these steps to clear a jam:

- Remove the paper cassette. If there is a paper sticking partially out of the cassette or out of the printer, remove it; this may be causing the jam. Try putting the cassette back in and seeing if this works.
- If this is not the problem, open the top cover. At this point, you should see the jammed paper somewhere in the machine.
- Lift the metal handle located on the left side of the printer above the "high temperature warning" sign. The outside of this handle holds a bar from which the paper tray and the paper feeder assembly hang. Lift this bar from the hooks supporting it and let the entire assembly drop down and away from the drum. This will give you room to work.
- Carefully remove the piece of paper that is causing the jam. **Be careful not to touch the drum; you may scratch it!**
- Put everything back in place and close the printer. **Make sure you push the red "Reset" button before closing the top door; forcing the door shut without pressing this button will damage the printer.**

Places where paper is likely to get caught are the separator belt (towards the front of the feeder; it looks like a long rubber band), the cleaning roller (on top of the drum) and the cleaning blade (just behind the drum).

After you have solved the problem and closed the printer, it will begin where it has left off. If the parameter `jamresistance` is "on," the printer will put a message on the console telling you that it is reprinting the page it has lost because of the jam.

Pre-mix (2.2.2)

Adding pre-mix *(2.2.2.1)*

Whenever you need to add pre-mix, the printer will write the following message on the console and stop printing:

```
--Attention--
LBP-10 not ready: no pre-mix
```

The pre-mix indicator on the front panel should also be flashing; it is the light labelled with a "teardrop" shape. Before adding premix, open and shut the access door on the front once. If the light stops flashing and the printer resumes printing, do not add any more premix yet. This will prevent you from over-filling the developer tray.

To add premix, open a new bottle, shake it, and pour it *slowly* into the hole on the left side of the developer tray. This tray is at the bottom right-hand side of the space behind the front access door; the hole is underneath the yellow piece labelled "C" and immediately to the left of the large yellow handle. The tray should hold an entire bottle. If you overfill the tray, drain the excess as explained below:

Removing excess pre-mix *(2.2.2.2)*

If you overfill the developer tray, excess premix will start spilling out of the hole on the top. When this happens, slide the developer tray out four or five inches by pulling on the yellow handle. Free the flexible black drain-tube from the bottom of the tray and open an empty pre-mix bottle (keep one on hand for this purpose). Pinch the drain tube tightly; unfasten the cap on the end; place the end of the tube in the empty bottle; and let the tray drain through the tube. When enough pre-mix has been removed, pinch the black tube again, remove it from the bottle, and screw the cap back on to the tube. If you do this carefully, you will be able to drain pre-mix without spilling any on the floor or in the cabinet. After you have finished, put the tube back in its clip under the developer tray, push the tray back in, fold the yellow handle back against the printer, and close the access door.

After adding pre-mix, the printer will begin to operate again on its own.

Toner (2.2.3)

If the printer runs out of toner, it will print the following mesage on the console:

```
--Attention--
LBP-10 not ready: no toner
```

The toner is in a flat plastic bottle on top of the developer tray. To remove the old bottle of toner, open the front access door, turn the toner bottle 120 degrees counter-clockwise, and pull it out. Next, take a new bottle of toner and shake it thoroughly. Remove the cap from the new bottle and replace the cap with a valve assembly; there should be one in the toner-bottle box. The valve looks much like the cap and screws on in exactly the same way. Finally, put the toner bottle back into the developer tray with the valve between "one-o'clock" and "two o'clock." Push the bottle in as far as it will go. Then turn the bottle counter-clockwise until it stops and close the door. The printer will begin printing again where it left off.

When you restart the printer, the toner mechanism will make a few clunking sounds. These noises will also occur from time to time while the machine is on; again, the printer is adding toner. However, if the noise doesn't stop quickly, or if it occurs repeatedly before the toner bottle is empty, call your service organization.

Supplies (2.2.4)

Use only supplies provided by IMAGEN, or supplies we have approved. For convenience, we provide a prepackaged kit containing enough supplies to print 40,000 pages. This package, called **IMPAK**, contains 9 bottles of toner, 36 bottles (6 cartons) of premix, 80 reams (10 cartons) of NP paper, and 5 separator belt brushes. Of course, all these supplies may be ordered separately. The package does not contain separator belts or the photosensitive drum; these are usually replaced by service representatives, although they are available from IMAGEN if necessary.

Routine maintenance (2.3)

Routine maintenance schedule (2.3.1)

Unless you are under the *Self-Maintenance Policy*, IMAGEN or our service representative will do preventive maintenance every 48,000 pages. Use the counter, which is towards the front on the lower-right side of the printer, to keep track of how many pages you have printed. 48,000 pages is not a lot; if you print 200 pages per day, which is a fairly light load, you will need preventive maintenance about every 10 months. It would be worth your effort to figure out how many pages you print on an average day to get a rough idea about when to schedule regular service.

In addition to this regular schedule, IMAGEN will tune the printer, both after it has "broken in," (after its first 2,000 pages), and when it needs tuning later. During the "break-in" period, the print quality will slowly degrade. Don't worry about this; it is normal. After the printer has broken in, the print quality will remain consistently good for tens of thousands of pages. IMAGEN will also replace the separator belt when necessary.

The rest of this section describes these routine maintenance procedures; unless you are under the *Self-Maintenance Policy*, you need only concern yourself with cleaning the corona wires and replacing the separator brush, both of which are fairly simple operations. It will be easiest if one person is responsible for all these operations. Whether we perform routine maintenance or you do it yourself, we recommend the following schedule:

- Replace the separator brush every 8000 pages
- Replace the separator belt every 24,000 pages
- Clean the corona wires once a week
- Tune the printer whenever the print quality is poor after inserting new paper. This should not need to be done often; every 30 or 40 thousand pages should suffice. If your printer needs tuning more often, and you have checked to make sure that you are using the correct paper, that the paper is fresh, and that you are putting it in right-side up (see section 2.2.1.1), call your service representative.

Replacing the separator belt and brush (2.3.2)

Replacing the separator brush *(2.3.2.1)*

Before replacing the separator brush, you must get yourself some room to work. First remove the paper cassette, open the printer's top cover, and lift the metal handle located on the left side of the printer above the "high temperature warning" sign. The outside of this handle holds a bar from which the paper tray and the paper feeder assembly hang. Lift this bar from the hooks supporting it and let the entire assembly drop down and away from the drum. This process is also described in section 2.2.1.2, "Freeing Paper Jams."

After doing this, you will be able to replace the brush easily. The separator assembly is the cluster of pulleys mounted on a silver metal piece at the side of the printer. On the right hand side of the assembly is a small yellow lever marked with a black arrow. Move this lever as far as it goes in the direction of the arrow. The brush looks like a miniature toothbrush with a yellow handle about one-half inch long. Slide the brush out of its holder and replace it with a new one.

Removing the separator assembly *(2.3.2.2)*

The separator assembly contains the separator brush, the separator belt, and the assorted pulleys that guide the separator belt. To replace the belt, you will have to remove this assembly from the printer.

To remove the separator, first release the paper transport mechanism, as you did before replacing

the brush. Next to the separator, you will find a yellow lever labelled "Release Lever: PULL UP." To free the separator assembly, lift this lever. Be careful to support the assembly while you are doing this so that it doesn't fall back into the drum. At this point, simply lift the separator out of the printer.

To put the assembly back in, fit the metal wheel at the bottom back into notch it came from; this notch is at the bottom of a long, black curved piece at the side of the printer. Then fit the white plastic wheel at the top of the assembly back into the curved slot at the top of the same black piece; slide it back to the beginning of the slot and push down on the "release lever," which will lock the assembly back into place.

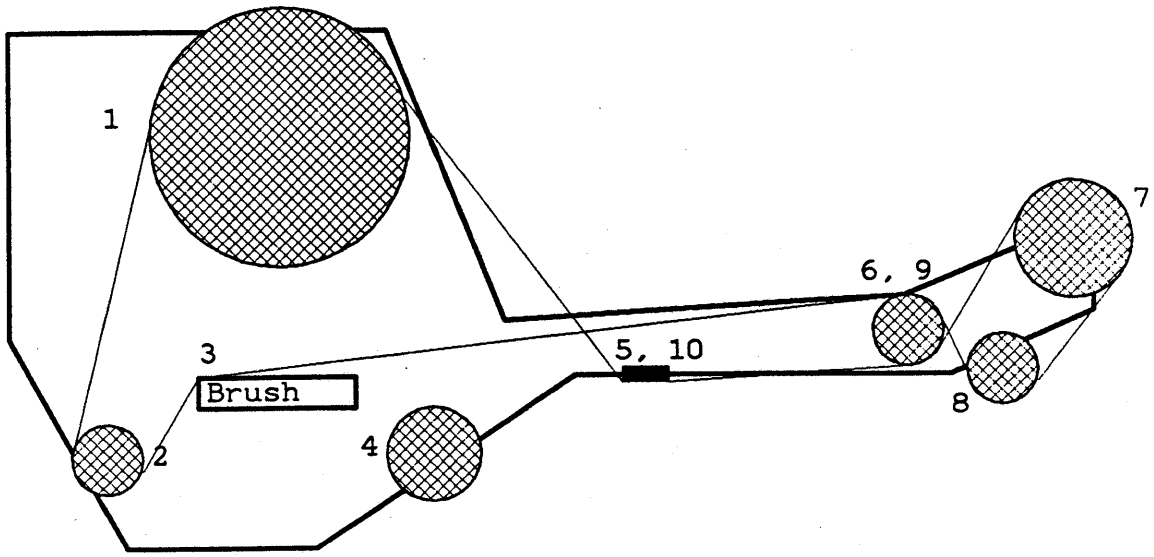
Before you close the cabinet, be sure that you push the red "RESET" button in the direction of the arrow until it clicks. If you don't, you could damage the printer.

Replacing the separator belt *(2.3.2.3)*

The separator belt is the long, thin loop threaded around the pulleys on the separator assembly. After you have removed the brush, there will be enough "give" in the assembly to let you remove the belt and replace it with a new one easily. The new belt must follow the same path as the old one; since this is probably too complicated to remember, it must go

- Around the large white pulley
- Around the small pulley above the brush
- Past the brush
- Past, but not around, the medium-sized pulley beneath the brush
- Over the metal "tounge" that sticks out from the assembly
- Over the inside of the double pulley near the bottom of the assembly ("inside" means the side towards you when you hold the assembly as it fits into the printer.)
- Under the small white pulley immediately immediately below the double pulley
- Around the pulley mounted at an odd angle at the bottom of the assembly
- Under the "outside" of the double pulley
- Under the metal "tounge"
- Over the large pulley

Before replacing the belt, follow its path several times with this list, making sure that you know where it goes. The following picture gives an abstract diagram of the separator belt assembly, showing its path:



Separator assembly

Cleaning the corona wires (2.3.3)

Location *(2.3.3.1)*

First, open the front left-hand panel on the printer. You will see four yellow handles labelled "A," "B," "C," and "D." Pulling out the holders, which are long metal pieces attached to the plastic handles you see, will expose the corona wires; they are very thin, uninsulated wires running down the center of the holders.

Cleaning *(2.3.3.2)*

To clean the corona wires, brush them with the corona wire cleaner supplied with the printer. Continue wiping them until they appear bright gold; when they are dirty, they will be various shades of black. This should not present any difficulty, except for wire C, which has an additional cover protecting the wire. A single black screw on the side of the holder away from the handle holds this cover in place. To remove the cover, take out this screw and slide the cover towards the handle about one half inch. This will free the two tabs near the handle from the slots that hold them in place. When the tabs are free, lift the cover off gently; when you do this, the two remaining tabs in the center of the cover will pop free. Now clean wire "C" just as you cleaned the others. After cleaning the wire, replace the cover by reversing this procedure. Be careful to get the two tabs in the middle of the cover into their slots and make sure that you don't try to put the cover on upside down; in the correct position, the cover rises above the edge of the holder.

After cleaning the corona wires, put them back in the printer carefully. Each corona wire slides in a metal track and can only fit into the track one way; but if you have any doubt about how they go in, the top of the large letter on the plastic handle always faces *towards* the roller.

When you clean the corona wires, check to make sure that all 15 exposure lamps are working. To do this, you will need to defeat the printer's interlock system by inserting a thin pen into the small hole to the right of the printer's power switch. At this point, all 15 exposure lamps should

light. You can see 10 from looking down from the top; 5 are at the front of the drum, near the roller, and five at the back, near the "A" corona wire. The last five lights are behind the toner bottle. After checking, remove the pen and close the door. **Caution: Do not defeat the interlocks if the corona wires are not in place. This might expose you to laser light from the printer.**

Tuning the printer (2.3.4)

Tuning the printer involves adjusting the position of the corona wires so that they produce the best possible image on the paper. This process is more tedious than difficult; the biggest problem is learning how to discern which of two slightly different images is better. The process we give here is designed to help you learn this and get a feel for the effect each adjustment has on the printing quality. After you have gotten comfortable with this, you will be able to make your own shortcuts in the procedure. Tuning the printer should take you about two hours the first time; it should become much quicker thereafter.

Theory *(2.3.4.1)*

Each corona wire has a different effect on the print quality. Along with the laser, wires "A" and "B" put an electrical charge on the drum which determines where the drum will pick up toner. These two wires determine whether or not the printer is will "focused"; light, fuzzy characters point to badly adjusted "A" and "B" wires. Wire "C" controls the amount of toner that remains on the drum; it adjusts the print's "darkness." If this wire is badly adjusted, the print will be too light or will tend to smear; more will be said about this later. Wire "D" helps to transfer the toner from the drum to the paper. This usually will not need to be adjusted.

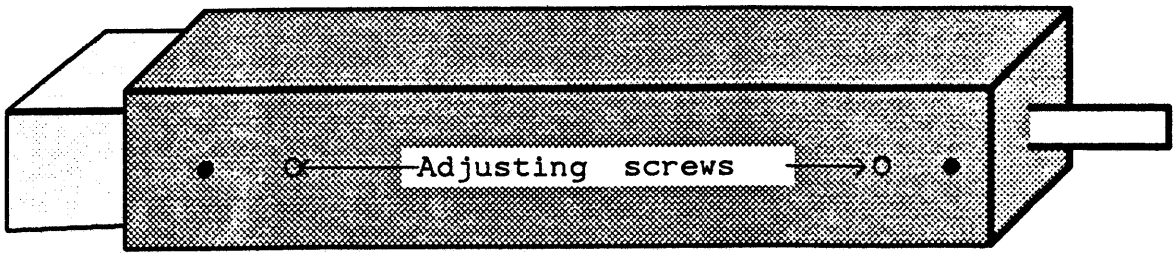
Precautions *(2.3.4.2)*

When tuning the printer, please take these precautions:

- Insert the corona wires into the printer gently and carefully; otherwise, you may scratch the drum. Likewise, do not touch the drum at any time.
- While you are tuning, watch out for very small dots on the page. They will appear if either the "A" or the "B" wire has been moved too close to the drum and is arcing. If this continues for long, the drum will be damaged.
- *Do not* defeat the interlocks when the corona wires are not in place. The laser in the printer is powerful enough to injure you if you are exposed to it.

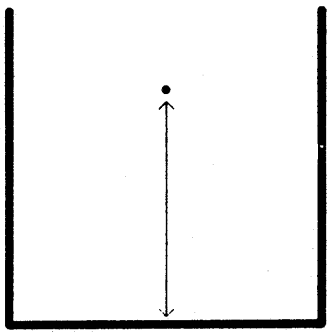
Making tuning adjustments and measurements *(2.3.4.3)*

All corona wire adjustments are made the same way. Two plastic screws determine the position of each wire (the screws are made of clear plastic, but may appear black when they are dirty). They are both on the back side of the long corona wire holder (the side away from the drum when the wires are in place), one at either end; and they are located "inside" of the two metallic black Phillips-head screws. Turning the screws clockwise moves the corona wire away from the drum; turning them counter-clockwise moves them towards the drum. Whenever you are adjusting these screws, be careful to turn both of them the same amount so that the corona wire remains parallel to the drum. In addition, be careful not to force these screws. Since they are plastic, it is easy to damage them. The diagram shows the position of these screws on the back side of the holder:



Corona wire holder: back view

When you need to measure the position of the corona wires, use a small, accurate millimeter scale. Measure the position from the bottom of the holder (on the inside) to the wire, sighting along the corona wire holder to eliminate parallax. Again, the bottom of the holder is the side facing away from the drum, or the side at the bottom of the label on the handle. The arrows in the figure below show the proper distance to measure for a typical corona wire. Take some care and time with this measurement; it is a bit tricky to get it right. Note that, since the open side of the holder (the "top" of the "u") faces towards the drum, larger measurements here mean that the corona wire is closer to the drum.



Corona wire cross-section

Tuning *(2.3.4.4)*

To begin the tuning process, start with these four steps:

- Begin by loading the printer with *fresh* paper; make sure that the *top* side of the paper, as marked on the package, is facing "up" when loaded into the paper cassette.

- Clean all of the corona wires (see section 2.3.3).
- Adjust all four wires to the recommended position for each wire. The initial position for "A" and "B" is listed inside the front cover, below the power switch. It will give separate positions for both the front (handle) and rear ends of the wire. The initial position for the "C" wire is 14 mm from the back of the holder; the initial position for "D" is 9 mm. See section 2.3.4.3 for a discussion of how to make these adjustments and measurements.
- Print any test file and mark it as your starting point (i.e. "A nominal, B nominal, C nominal"). From now on, all the adjustments you make will be relative to this starting point. You can use any file for a test sample, provided that it contains both large and small characters. However, it is probably easiest to use the built-in test pattern, which you can print by pushing the "Test" button on the front panel.

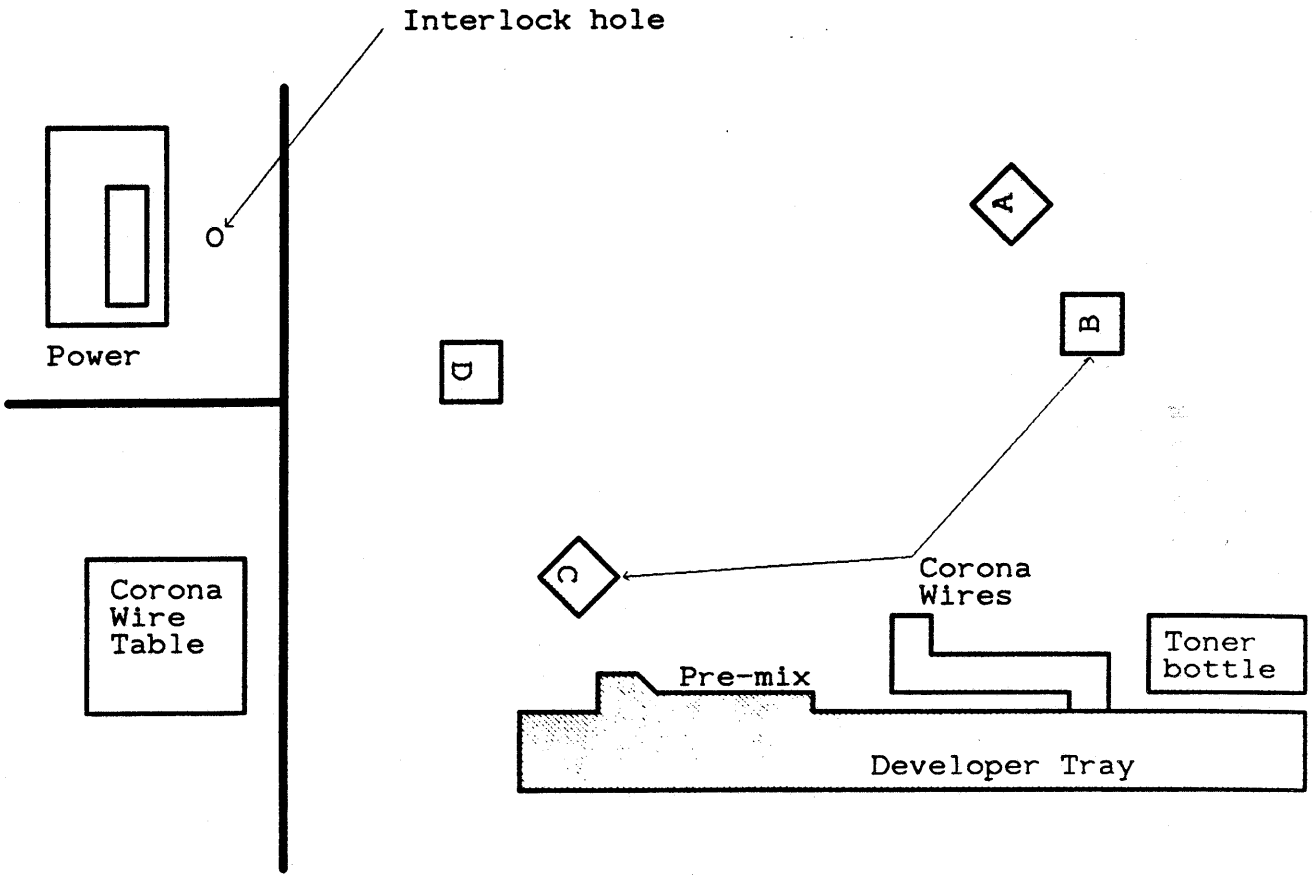
The next three steps describe the tuning process itself. The adjustment of wire "D" is not extremely critical; once it has been set at 9 mm, you can leave it. While the adjustment of wires "A" and "B" effect each other, adjusting "A" first, as we suggest here, reduces this problem enough so that you will not have to touch up "A" after adjusting "B." However, the reverse is *not* true; if you adjust "B" first, you will probably have to re-adjust it after adjusting "A." Be aware of this if you depart from our suggested procedure.

- Move the "A" corona wire away from the drum by turning both adjusting screws 1 turn clockwise. Print a test sheet, marking it with the wire's new position (e.g. "A out 1, B nominal, C nominal"). Repeat this process until you have moved the wire out as far as it will go, being careful not to force the adjusting screws. After the wire is as far away from the drum as possible, reverse the process, now turning the adjustment screws one turn counter-clockwise each time. Continue this process, making (and labelling) test sheets as you go along until the adjustment reaches its limit *or* the printer begins to arc (producing small random spots on the page). At this point, look at all the test sheets, decided which one is the clearest, and return the corona wire to this setting. **Reminder: Keep good records while doing this or you will get confused very quickly.**
- Do the same thing with the "B" wire, adjusting the screws one half turn each time instead of a full turn. When you have finished, again decide which sample looks best and return the "B" screws to that setting.
- Do the same thing with the "C" wire, again adjusting the screws a full turn each time. If the print smears or rubs off, the wire is too far away from the drum, and you will have to move it closer by turning the screws counter-clockwise. If the characters are blurry or not black enough, the wire is too close to the drum; turn the screws clockwise. Note that the best setting here may not simply be the one that gives the best appearance; watch for print that smears, in addition to print that looks crisp and clear. You may want to run the printer a little "wet" (i.e. a little on the "smeary" side) to get the best printing results.

Use this procedure the first few times you tune the printer. When it has become routine, you can try leaving out steps; rather than going through the whole range of "A" adjustments, for example, you might just judge from the printing quality what direction you want to move in. If you get too far off track, you can always return to this procedure. Still, don't experiment until you are completely comfortable with what you are doing.

Diagram (2.4)

The following diagram shows what appears when you open the front access door on the IMPRINT-10. This door exposes most of the parts discussed under "Daily Operation" and "Routine Maintenance"; the corona wire holders are labelled A, B, C, and D as they are in the printer.



Behind the front access door

CHAPTER 3
Document Structure

Introduction (3.1)

Each distinct IMPRINT-10 printing task, or job, is called a *document*. Documents may contain printing instructions in a variety of forms; each such form recognized by the IMPRINT-10 is called a *document language*. This implies that each document must, at the least, specify the language in which it is written. Other document parameters can be given to control the interpretation of the language in question, as well as the printing process itself (e.g., the number of copies, the paper type, etc.). To this end, a document may have any number of *document headers*, each specified in a universal *document control language, DCL* for short.

A document, then, is a sequence of document headers whose structure is defined below, followed by a sequence of bytes which comprise the body of the document, whose form is dictated by the document language specified in the document headers. We allow multiple document headers so that various processes involved in producing and communicating a document for the IMPRINT-10 may specify something about the document, without knowledge of its content.

Document Header Structure (3.2)

Each document header is the ASCII string `@document(` (case is unimportant), followed by a comma-separated list of *document control items*, terminated by the ASCII close-parenthesis (the character)). Document control items are comprised of one or two *atoms*, the first being a keyword, and the second, an optional value. An atom is either a white-space-separated sequence of the ASCII characters

```
a-z A-Z 0-9 - . _ /
```

(alphanumerics, dash, period, underscore, and slash) or a sequence of any ASCII characters delimited by two double-quote characters ("), what we will call a *string*; to include a quote character in a string, double it. *White space* is a sequence one or more ASCII spaces (32_{10}), carriage-returns (13_{10}), line-feeds (10_{10}), and tabs (9_{10}). Atoms' case distinctions and form, either quoted or unquoted, are indifferent. Atoms of either form must be less than 256 characters in length. (See section 3.4 for examples.)

Document Header Semantics (3.3)

Document header keyword/value pairs specify the document language, language-dependent document options, language-independent printing-system control options, and comments.

In the presence of multiple occurrences of a given keyword, a later occurrence overrides any earlier occurrence, under the assumption that the "innermost" document control "knows better" about the document content. Any keyword not understood by the system is a comment, allowing for document header page annotations; however, these keywords will be marked specially on the job header page. (See section 7.2.1, page 55.)

A keyword's value may be an integer, a boolean, or a string. In the case of an integer, the value must consist of ASCII digits (0-9), optionally preceded by an arithmetic sign, the ASCII hyphen (-); further, the integral value must lie in the range $[-2^{16}, 2^{16})$. A boolean value must belong to the set of atoms `on`, `off`, `true`, `false`, `yes`, `no`. A string's contents are usually uninterpreted. In the sequel, we designate meta-values of these three types by *int*, *bool*, and *string*, respectively.

A keyword's value may be entirely omitted, which, in the case of integer values, is equivalent to not specifying the keyword at all, in the case of boolean values, is equivalent to a `true` value, and, in the case of string values, is equivalent to the empty string (`""`).

Document Language (3.3.1)

The `language` keyword's value specifies the document's intended language. The current set of choices includes:

`language Impress`
: the Impress language is a full page-layout and graphics language (see Chapter 9);

`language Printer`
: the Printer language fully emulates an ASCII line printer, with several options for portrait and landscape orientation, 1 or 2 logical pages per physical page, etc. (see Chapter 10);

`language Daisy`
: the Daisy language emulates a subset of the *de facto* daisy-wheel standard, the Diablo 1640 printer command set (see Chapter 11);

`language Tektronix`
: the Tektronix language provides a subset of the Tektronix 4014 display terminal command set. (see Chapter 12)

Language Dependent Document Control (3.3.2)

Most of the document languages obey document control items private to their task. For example, the Printer language uses several keywords with integral values to control printed page layout, e.g., `formsperpage` and `formwidth`. See the appropriate language Chapter, section "Document Control," for more information.

Language Independent Document Control (3.3.3)

There are document control items which direct the IMPRINT-10's treatment of a document, regardless of the document language. They are listed below, broken down into groups by the interested IMPRINT-10 subsystem.

Paper Handling *(3.3.3.1)*

`copies` *int*
: controls the number of document copies printed; also see `pagereversal` and pagecollation for related copy control (default `copies 1`)

`jamresistance` *bool*
: controls whether the printing process guarantees that a page cannot be lost due to paper problems (default `jamresistance off`)

`pagecollation` *bool*
: controls, in the presence of a multiple copies request, whether the copies will be collated (default `pagecollation off`)

`pagereversal` *bool*
: controls whether the document will be printed in reverse page order (which, for the LBP-10, is actually forward page order, assuming the document is presented in forward order) (default `pagereversal off`)

`paper` *papervalue*
: declares the document's assumed paper type, which implies a specific paper size; any mismatch between this value and a loaded paper cassette will produce a warning on the document header page; *papervalue* is chosen from:

`Letter`	11.0 inches high x 8.5 inches wide
`A4`	11.7 x 8.3
`A6`	6.7 x 4.1
`B5`	10.1 x 7.2
`B6`	7.2 x 5.1
`GLetter`	10.0 x 8.0 (short for Government Letter)
`GLegal`	13.0 x 8.0 (short for Government Legal)
`Foolscap`	13.0 x 8.5
`Folio`	13.0 x 8.3
`A5`	8.3 x 5.8
`Statement`	5.5 x 8.5
`Legal`	14.0 x 8.5
`B4`	14.3 x 10.1, but see `paperwidth` description, below

(default `paper letter`)

`paperheight` *int*
: declares the document's desired paper height, in pixels (not honored if greater than 3600 (15 inches)) (defaults to the height of the document's `paper` type)

`papermargin` *marginvalue*
: declares the document's desired left margin handling, with values chosen from

`old`	the virtual left edge of each page is actually about 1/8" to the left of the physical edge
`zero`	the virtual left edge of each page coincides with the physical edge

`visible` the virtual left edge of each page is approximately 3/8" to the right of the physical left edge (the leftmost 3/8" of the physical printed page is not visible due to drum limitations)

(default `papermargin zero`)

`paperwidth` *int*
declares the document's desired paper width, in pixels (not honored if less than 960 (4 inches) or greater than 2304 (about 9.6 inches)) (defaults to the width of the document's paper type)

Message Subsystem *(3.3.3.2)*

`jobheader` *bool*
controls whether a job header page is printed or not (default `jobheader off`); note that a header may be produced in any event, if an error occurs during document processing

`jobmemory` *int*
an integer from the range [1, 10] which controls the relative amount of memory allocated for recording job messages; 1 is the minimum, and 10 the maximum (default `jobmemory 1`)

`messagedetail` *bool*
controls whether messages are printed on the job header page in full (in particular, whether the detailed information about each occurrence is printed) (default `messagedetail off`); if `false`, then only the occurrence count for each message is printed, unless the message in question always includes its occurrence detail

`maxerrors` *int*
sets the error limit for document processing; if more than this number of errors are encountered, document processing will be abandoned (default `maxerrors 10000`)

Console Handling *(3.3.3.3)*

`name` *string*
names the document; used in messages to the console to identify the job (default `name ""`)

`owner` *string*
names the document owner (the person or entity the document was produced by or for); used in messages to the console to further identify the job (default `owner ""`)

Examples (3.4)

Two examples of valid document headers are:

```
@document(LANGUAGE imPRESS, jobmemory 10, pageReversal)
@Document("language" impress,
  jobmemory "5", jobheader on, copies
   5,
  name "A document named ""name"""
    , spooldate 10/12/83-1010pm-est)
```

Note that the language specifications are identical, even though the case and atom types differ (i.e., `"language"` and `LANGUAGE`, and `imPRESS` and `impress`); also note that the `spooldate` keyword and value are comments for the job header page.

The IMAGEN-distributed spoolers for UNIX[tm] add the following document header to the start of every document sent to the IMPRINT-10:

```
@document(owner "unix-login-id (full-name)",
  spooldate "today's date and time",
  jobheader on, jamresistance on) any other document headers...
```

The first three items identify the document for sorting into output bins by requesting a job "header page" (actually printed last) with information about the user who spooled the job, and the fourth insists on conservative paper handling to ensure no pages are lost due to LBP-10 problems.

As another example, `imprint`, the UNIX line-printer emulator interface to the IMPRINT-10, simply prepends the document header

```
@document(language Printer, files "list of files following",
          pagereversal on, copies # copies requested, pagecollation on)
```

to the files to be listed and sends the result to the IMPRINT-10;[1] the `files` list is just a comment, but the `language Printer` specification is needed to tell the IMPRINT-10 what sort of document follows (in this case, line-printer output). The `pagereversal` and `pagecollation` items (see section 5.1, page 37) result in the most natural order for listings, viz., first page on top, with any copies fully collated.

As a warning, note that though document headers are intended to be human-readable, the actual document contents start *immediately* after the closing parenthesis of the last document header, and not on the next "line." Thus, the following document header will result in a line printer listing with an initial blank line, since a new-line character is present after the last (and first, in this case) document header:

```
@document(language Printer)
This is the second line of text (the first is blank).
```

Multiple document headers make sense in many host environments. For example, any software generating Impress documents will prefix the document with the simple header

```
@document(language Impress)
```

and any software spooling the generated document will prefix its output with its own header, e.g.,

```
@document(owner "jqj (John Q. Johnson)")
```

The resulting document, as the IMPRINT-10 sees it, will have the two headers concatenated directly, the second preceeding the first, i.e.,

```
@document(owner "jqj (John Q. Johnson)")@document(language Impress)
```

[1] The details are actually a little more complicated, depending on the type of listing format requested.

CHAPTER 4
Operator's Console

Introduction (4.1)

The IMPRINT-10 provides a serial port for an optional ASCII operator's console terminal; you don't need a console terminal to use the IMPRINT-10, but it is useful for viewing and controlling system operation. Interactions over this port take place in the *operator's console language*, described in this chapter.

This chapter is broken into seven sections. The first discusses the messages that the console will produce on its own, whether or not you enter a command. The second describes how to use the console to enter commands to the printer. The third lists the commands that are available; the fourth describes aids we have provided to make entering commands easier; and the last three sections describe the control panel lights on the front of the printer, the messages produced whenever the processor is turned on, and the messages which appear when the printer has some internal problem.

The console messages described in this chapter are those relevant to system operation and control, as distinct from the set of messages relevant to document processing. The latter set is documented in Chapter 7, page 55.

To get the most out of this chapter, try reading it over once, and then going over it again at the console terminal, trying out the examples below, and further exploring the various commands.

Notation (4.1.1)

In this chapter, the notation "c-*keyname*" will refer to typing the key named *keyname* with the console terminal's control key held down. For example, c-S is the code generated by holding down the control key and typing an s (or S). Also, we refer to the standard, unshifted ASCII control keys by their commonly-understood names, e.g., "carriage-return," "line-feed," "break," "escape," "rubout" or "delete," etc. These keys may be labelled as something else on your console's keyboard, e.g., RET, CR, or RETURN, LF or LINE, BREAK or BRK, ESC or ALT, RUB or DEL, etc.

Console Messages (4.2)

This section describes messages that the Imprint-10 will produce without any intervention to give you information about what it is doing. Messages which the Imprint-10 produces in response to specific console commands are covered later.

If a message is considered important to the console user, such as an LBP-10 problem, or a job or page needing release, then the IMPRINT-10 prefixes the message with

```
-- Attention --
```
and rings the console terminal's bell.

Job boundary messages (4.2.1)

The IMPRINT-10 keeps the operator informed about crucial points in a document's progress through the system, using the following messages:

`Starting new job`
A new document has arrived, and the IMPRINT-10 is looking at the document's header (see Chapter 3, page 19). When finished with the header, the IMPRINT-10 produces one or more of the following messages:

`Language:` *document language*
`Name:` *document name*
`Owner:` *owner name*
`Copies:` *# copies*, `collation` *on/off*, `reversal` *on/off*
`Paper:` *paper type*

where *document language* is the name of the document form (e.g., `Impress`, `Printer`, etc.), the *document name* and *owner name* are optional document control information items (useful for console reporting, mostly), the `Copies:` line tells you how the system is grouping the document's output page set (see Chapter 5, page 37), and the *paper type* is that requested in the document header (see section 3.3.3.1, page 21 for the list of IMPRINT-10 paper types).

`Job failed`
A `Print file` request has failed for some reason, reported earlier.

`Printing job header`
The IMPRINT-10 has entirely finished processing a document, and is starting to produce a set of job header pages with information about the job (see Chapter 7, page 55).

`Job finished`
The document and any job header have been completely processed. The IMPRINT-10 is now ready to process another document.

Page handling messages (4.2.2)

When `Set page-messages` is in effect (see section 4.4.6.2), the IMPRINT-10 reports page activity with the following messages:

`Compiled page` *page #*
Whenever the processor has finished preparing a page for printing, it will print this message giving you the number of the page it has completed. When the printer is ready for it, the page will be printed. In most cases, the processor works faster than the printer itelf, so you may well see that the processor has compiled page 30 before the printer has printed page 10. *Page #* is the page number, beginning at 1.

`Printing page` *page #*, `copy` *copy #*
The IMPRINT-10 is starting to print the named page and copy. Note that some delay may be apparent between the appearance of this message and the actual start of paper movement, as the LBP-10 may be in the middle of a warm-up or catch-up cycle.

`Waiting for release of page` *page #*, `copy` *copy #*
The IMPRINT-10 is ready to print the named page and copy, but `Hold page-printing` is in effect (see section 4.4.2.2).

Communications messages (4.2.3)

With some communications interfaces, the sending host can request the prompt abandonment of the current job. In that case, the IMPRINT-10 will display the following message, and proceed as if an `Abort (current job)` command were given at the console:

```
Host requesting job abort
```

LBP-10 messages (4.2.4)

When LBP-10 problems occur, the IMPRINT-10 will report them asychronously and immediately. Each problem will produce a message on the console; these messages all begin `LBP-10 not ready`. Most of these reports deal with simple problems that can be corrected easily; for example, the printer may have run out of toner or some other supply. To correct these problems, see the chapter on Basic Operation, section 2.2.

The system will also produce a message when it restarts after a problem has been corrected. These messages all begin `LBP-10 ready`, and report the current state of the system.

Here are the messages used to report the condition of the LBP-10:

`LBP-10 ready; Paper cassette:` *type*
: The LBP-10 is fully operational, with the paper cassette *type* loaded. *Type* is one of `A4`, `A6/B5/B6/G.Letter`, `G.Legal/Foolscap/Folio`, `A5/Statement`, `Universal`, `Letter`, or `Legal/B4`.

`LBP-10 not ready:` *reasons*
: The LBP-10 has one or more problems which must be fixed before printing can resume; see the chapter on Basic Operation, chapter 2, page 5 for detailed information on the operation and care of the LBP-10. *Reasons* is one or more of:

  `no paper`
  : Either the paper supply cassette is empty, or there is no cassette loaded; add some paper.

  `no pre-mix`
  : The LBP-10 is too low on dispersant to continue; add some.

  `no toner`
  : The LBP-10 is too low on toner to continue; add some.

  `paper jam`
  : The LBP-10 has detected a paper jam (a piece of paper fed from the paper cassette has not made its way to the output bin in time). You must clear the paper path (opening the top door and resetting the door interlock) before printing can resume.

  `power off or door(s) open`
  : The LBP-10 is dead as far as the IMPRINT-10 is concerned. This can happen in various ways: the power can be off, one or both of the doors (top or front) can be open, or the LBP-10 could have detected an internal fault which caused it to shut itself off. In any case, the green READY light on the LBP-10 front panel is either off or flashing.

  `service call needed`
  : The LBP-10's internal microprocessor has detected some severe problem (e.g., laser power too low) which requires a service call.

Basics (4.3)

This section describes how to enter commands on the console; the previous section only required you to watch. The console may be either a CRT or a printing terminal; the IMPRINT-10 can take advantage of a CRT, as described in section 8.4, page 62.

The Prompt (4.3.1)

The IMPRINT-10 is always listening to its console terminal, though it may be slow to respond when printing is in progress. It displays a *prompt* of the form

```
IMP>
```

to indicate its readiness for a command. This prompt may be "lost" in the midst of unsolicited messages which appear on the console, but it is easily "retrieved" by typing a carriage-return or space.[2] You may type characters without waiting for the prompt, as nothing you type will be lost.

The IMPRINT-10 attempts to keep your command input lines visually distinct from its output by indenting its output lines. For example, in the exchange

```
IMP>Information (about) LBP-10
 LBP-10 ready
 Paper cassette: Letter
```

the IMP> prompt starts at the left margin of the console terminal, and the response lines are indented one space.

Entering Commands (4.3.2)

All console language commands and subcommands are given by typing their first letter (ignoring alphabetic case distinctions). For example, the command `Hold page-printing` is given by typing `h` and then `p`; after you type the `h`, the IMPRINT-10 completes the command by typing `Hold`, waits for you to type the first letter of the subcommand (in this case, `p`), completes the subcommand by typing `page-printing`, and, finally, types a carriage-return and its prompt. This example would look like

```
IMP>Hold page-printing
IMP>
```

(remember, you don't type the IMP> prompts--they are given by the IMPRINT-10).

Typing Text Lines (4.3.3)

Some commands prompt for a line of text with a colon, as in

```
IMP>Print file:
```

If you make a mistake while typing a line of text, you can use several editing characters to fix your mistake. Some of these have been mentioned earlier; here is the complete list:

return, line-feed	terminate input
c-H, rubout, delete	back up one character
c-W	back up one "word," where word is defined as a sequence of numeric and alphabetic characters

[2] A null (c-space or c-@ on most terminals), a break, line-feed, or a tab will also re-elicit the system prompt.

c-U	erase everything you've typed so far and start over
c-R	retype the line accumulated so far
c-C	abort this line of input
c-@ (null)	ignored

Also, each text line has a maximum number of characters. If you try to type more characters than the line permits, the IMPRINT-10 will simply beep at you.

Each control character not described above will be "echoed" as an ASCII uparrow, followed by its printable cognate; e.g., if you typed a c-T to the `Print file:` prompt, the result would look like

```
IMP>Print file: ^T
```

(which doesn't make any sense, of course, as no file name has a c-T in it).

Since some commands using texts are sensitive to leading or trailing blank characters, be careful not to include spaces before or after text strings. The `Print file:` command is one of these.

Abandoning a Command (4.3.4)

At any point, a c-U will abandon the current command; the IMPRINT-10 will display a XXX, followed by a new-line, to indicate its compliance with your decision. For example, imagine that you had typed an `s` to give a `Set` command, changed your mind, and then typed a c-U to give up; the console dialogue would look like:

```
IMP>Set XXX
IMP>
```

Some commands require *confirmation* before they will do anything, which you give by typing a carriage-return; typing a c-U will, as before, abort the command.[3] For example, the command to end the current job requires confirmation before taking its rather drastic action; if you were to type an `e` to select the `end` command, you'd see

```
IMP>End (current job) [Confirm]
```

to which you'd reply with a carriage-return if you wanted to go ahead, looking something like

```
IMP>End (current job) [Confirm]
 Ending current job
IMP>
```

and with c-U if you had changed your mind, which would look like

```
IMP>End (current job) [Confirm] XXX
IMP>
```

Commands (4.4)

This section details each console command and its response, in alphabetic order. (See section 4.2, page 25 for console messages which don't occur as a response to a command, and see Chapter 7, page 55, for more information on messages which are the result of document processing.) Our use of the term IMPRINT-10 *job* is synonymous with the term *document* (see Chapter 1, page 1), but with an active sense, i.e., the actual process of producing a set of printed pages from a given document specification in a particular language.

[3] Actually, typing anything but a carriage-return will abort the command.

End Command (4.4.1)

The End (current job) command forces the IMPRINT-10 to "see" an end of document, as if the document were complete. This command requires confirmation (see section 4.3.4, page 29), and has no effect if there is no current job (and nothing special is reported in this case). This is something of a drastic measure, as it can wreak all sorts of havoc with all but the simplest document structures; it is really only useful with the simplest of languages such as Daisy or Printer.

```
IMP>End (current job) [Confirm]
 Forcing end of job
 Operator ended job
IMP>
```

The End command has the following response:

`Forcing end of job`
Self-explanatory

Hold Commands (4.4.2)

The Hold commands temporarily stop either job (document) processing or page printing. (Also see the corresponding Release commands, section 4.4.5, page 32.)

Hold jobs Command *(4.4.2.1)*

The Hold jobs command stops job processing until released with a Release jobs command (see section 4.4.5.1, page 32). This command elicits no direct response, but can indirectly produce the following message:

`Waiting for release of new job`
A new job has arrived, but jobs are being held, and the system is otherwise ready to process it.

Hold page-printing Command *(4.4.2.2)*

The Hold page-printing command stops page printing until released. This command elicits no direct response, but can indirectly produce the following message:

`Waiting for release of page` *page #*`, copy` *copy #*
The given page (and copy) is ready to be printed, but page printing is held, and the system is otherwise ready to print.

Information Commands (4.4.3)

The Information commands let you inspect various aspects of the printing system state. They follow, in alphabetic order.

Information (about) communications Command *(4.4.3.1)*

This command elicits a response from the communications subsystem, detailing various things about the state of communications, e.g., the number of bytes processed in the current document so far. See the communications option manual in question for more specific information.

Information (about) file-system Command *(4.4.3.2)*

For each file in the file system, the Information (about) file-system command displays the following information: the file name, its creation date, the author, and source file (the last three referring only to the IMAGEN environment). This information can help IMAGEN Customer Support determine which versions of each file you have in your IMPRINT-10. A file in the file

system can be a font, an emulator for a given document language, a document (e.g, `Testjob`, the document printed when the LBP-10 TEST button is used), or the system's bootable image (the file `IMP`).

Information (about) jobs Command *(4.4.3.3)*

The `Information (about) jobs` command will display one or more of the following statements about the current job (in order of appearance, where /brackets/ denote optional items):

`Waiting for release of page` *page #*`, copy` *copy #*
: The given page (and copy) is being held by the `Hold page-printing` command (see section 4.4.2.2, page 30).

`Page printing held`
: `Hold page-printing` is in effect, but there is no page currently being held.

`Printing page` *page #*`, copy` *copy #*
: The given page (and copy) is currently being printed, or was the last page printed.

`Waiting for release of new job`
: The `Hold jobs` command is in effect, there is a new job ready to process, and the system is ready for it.

`Holding jobs`
: The `Hold jobs` command is in effect, but either there is no new job to process, or the system isn't ready to process it.

/`Not`/ `Reporting page printing`, /`Not`/ `Reporting job messages`
: Tells you whether the IMPRINT-10 is reporting page printing or job messages (see sections 4.4.6.2, page 33, and 4.4.6.1, page 33).

`No job current`
: The system is currently not processing any document.

`Initializing job`
: The system is initializing itself to process a job (this is an extremely short phase of document processing, so you'll rarely see this message).

`Interpreting document control information of new job`
: The IMPRINT-10 is looking at the document control header of a job (see Chapter 3, page 19, for more information about document structure), so nothing else is known yet about the job.

`Printing` /`header of`/ /`aborted`/ `job:` *description*
: The IMPRINT-10 is processing a document; the *description* is structured as:

    `Language:` *document language*
    `Name:` *document name*
    `Owner:` *owner name*
    `Copies:` *# copies*`, collation` *on/off*`, reversal` *on/off*
    `Paper:` *paper type*
    `Jobs queued:` *queued*`; jobs processed:` *processed*

  where *document language* is the name of the document form (e.g., `Impress`, `Printer`, `Daisy`, `Tektronix`), *document name* and *owner name* are optional document control information items, the `Copies:` line tells how the system is grouping the document's output page set (see Chapter 5, page 37), and the *paper type* is that requested in the document header (see section 3.3.3.1, page 21, for the list of paper types). If the `header of` message appears in the first line of the job description, then the system is preparing and printing the job header page(s). If the `aborted` message appears, the job has been aborted for some reason (and the reason has already been reported, e.g., too many document processing

errors). Finally, the number of jobs pending processing (*queued*) is displayed, as well as the number of jobs processed so far (*processed*, which includes the current job).

`Flushing input leftover from /aborted/ job:` *description*
The IMPRINT-10 is flushing the rest of a document, either because the document has been aborted, or because the document interpretation has finished before the physical end of the document (e.g., an Impress EOF command was seen before the end of the document). The job description is as above.

Information (about) LBP-10 *(4.4.3.4)*
The `Information (about) LBP-10` command requests the system to display the state of the Canon LBP-10 marking engine. The system's report uses the same set of messages discussed earlier under the heading "Console Messages," section 4.2.4.

Print file Command (4.4.4)

The `Print file` command prompts you for the name of a file whose contents are to be printed; the result looks like:

`IMP>Print file:` *filename*

where *filename* is a text line, with no leading or trailing blanks permitted. (See section 4.3.3, page 28, for information about entering text lines.) If the named file contains something other than a document, then you will get nonsense (it will most likely be aborted quickly). This is normally only useful with the file `Testjob` in a standard system; printing this file is equivalent to pushing the TEST button on the LBP-10 console (see section 4.6, page 34). The responses to this command are:

`File '`*filename*`' queued for printing`
The IMPRINT-10 is happy with your request and will print the file named *filename* at the next opportunity (documents queued in this fashion take priority over documents arriving from the host).

`Ignored (file '`*otherfilename*`' already queued)`
The file named *otherfilename* is already queued, and thus this new request can't be honored.

Release Commands (4.4.5)

The `Release` commands resume either job (document) processing or page printing (Also see the corresponding `Hold` commands, section 4.4.2, page 30.)

Release jobs Command *(4.4.5.1)*
The `Release jobs` command resumes job processing until held with a `Hold jobs` command (4.4.2.1). This command elicits no direct response, but can indirectly produce the messages for job startup (see section 4.2.1, page 26).

Release page-printing Command *(4.4.5.2)*
The `Release page-printing` command releases any output held as a result of a `Hold page-printing` command (see section 4.4.2.2, page 30). This command elicits no direct response, but can indirectly produce a message about page printing, as any released pages are printed (see section 4.2.2, page 26).

Set/Set no Commands (4.4.6)

The `Set` commands control some facet of IMPRINT-10 system operation; the `Set no` commands reverse the effect of the corresponding `Set` command. For example, `Set job-messages` tells the system to report document processing messages on the console, and `Set no job-messages` disables them.

Set job-messages *(4.4.6.1)*

The `Set job-messages` command turns on console reporting of job messages (those messages interesting to the document owner, which are produced as a result of document content processing, and which are reported on the job header page).

Set page-messages *(4.4.6.2)*

The `Set page-messages` command turns on console reporting of page-processing activity.

Operating Aids (4.5)

These commands have no effect on the actual operation of the printer. They exist to help you use the console more effectively by letting you ask what you can do at any point, letting you get more time to look at the console's messages if they are coming too fast, and letting you clean up the screen after a message has been printed.

Regulating Output (4.5.1)

If you are using a non-printing console terminal (a CRT), you may find that messages fly off the screen before you get a chance to read them. To solve this problem, you can type a c-S (the ASCII X-OFF character) to hold output, and a c-Q (the ASCII X-ON character) to resume output, without losing any messages. When you type a c-S, the message

```
--Held--
```

will appear on the console the next time the system has some output; when you type the corresponding c-Q, the held message will be erased from the screen, and normal output will continue.

Cancelling Output (4.5.2)

If, for some reason, you don't want to see the response to a console command, you can "flush" the response with a c-O (that's an "oh", not a zero). On a CRT, the message

```
--Flushed--
```

will appear, and any further messages will simply be discarded, until either the IMPRINT-10 re-prompts you for a command or you type something further, at which time the `--Flushed--` message will disappear; on a printing console, the `--Flushed--` message will appear, followed by a carriage-return, and messages will be discarded until either you are re-prompted or you type something.

Getting Help (4.5.3)

At any point, you can find out what commands or subcommands are available by typing a question mark, after which you will be re-prompted to supply the next subcommand element. For example, at the top-level IMP> prompt, if you type a question mark, the IMPRINT-10 will respond with

```
IMP>? One of: Abort, End, Hold, Information, Print, Release, Set
IMP>
```

where the second IMP> prompt in this example reminds you that a command is still possible. As a further example, suppose that you had typed an `s`, to which the IMPRINT-10 responded

```
IMP>Set
```

At this point, if you type a question mark, you will get the response

```
IMP>Set ? One of: job-messages, LBP-10-faking, no, page-messages
IMP>Set
```

to which you select a subcommand by typing its first letter (again, alphabetic case distinctions are unimportant). To finish this example, if you type `j`, the entire transaction will look like

```
IMP>Set ? One of: job-messages, LBP-10-faking, no, page-messages
IMP>Set job-messages
IMP>
```

Retrieving a Command (4.5.4)

At any point during your command input, an unsolicited message may appear on the console. This can be confusing at times, and the IMPRINT-10 provides a keyboard character to get your partial command back: c-R (for *r*etype, or *r*etrieve). For example, if you started a `Set job-messages` command, and after the `Set`, a message about paper problems occurred, you could restore the partially-completed command with c-R, and finish with the `job-messages`. This exchange would look like:

```
IMP>Set                                  You were about to type the j
 --Attention--                             when these messages appeared.
 LBP-10 not ready: no paper
IMP>Set job-messages                     You typed a c-R, and then the j
IMP>                                       to complete the command.
```

LBP-10 Control Panel Operation (4.6)

The LBP-10 control panel provides several switches and indicators.

Various problems lights (no paper, no pre-mix, no toner, paper jam) second any LBP-10 problem reports produced on the console (see section 4.2.4, page 27); they are useful if you don't have a console terminal.

Pushing the TEST button is equivalent to giving the command `Print file: Testjob` (see section 4.4.4, page 32): a test document is printed. The light above the TEST button will light until the IMPRINT-10 processes the test request (look closely if you're watching for it, as the request may be processed immediately, and thus it may light for only a fraction of a second).

The other buttons and their lights (SELECT and PRINT CHECK REQUEST) are ignored by the IMPRINT-10.

Start-up Messages (4.7)

When the system is powered on, it will choose the console port, depending on configuration options, report configurations on the console, announce the version of the CPU PROM-resident debugger and library, and attempt to boot the system. (For more detail on configuration options, see Chapter 8, page 61.)

Normally, a system start-up sequence will look like:

```
Ports: console: A at console speed baud, host: B at host speed baud
DDT68 (V version date) (C) 1983 IMAGEN Corp
Type any character to stop auto-boot...booting
File system: ..................... [i proms]
Loading IMP: ................
Loading IMP.sym: not found
Onboard nk, offboard mk
Starting
Communications configuration messages
 -- IMPRINT-10 Printing System vmajor.minor --
IMP>
```

where *Communications configuration messages* are documented in the communication option manual relevant to your system, and where *major* is the major software release version and *minor* the sub-release of that version.

If you type a character during the 5 second wait after the `Type any character...` prompt, you will be left at a

```
BOOT>
```

prompt. Typing a question mark to this prompt will give you an idea of the various commands available, but they are for IMAGEN Customer Support tasks, so nothing further will be said about them.

During booting, you may see one of the following problem reports:

(nothing)
: If nothing appears at all at system start-up, either the system is completely failing, or else you have the console terminal connected to the wrong port or at the wrong speed. Port "A" (the lower connector on the back of the IMPRINT-10) is the normal console port, running at 9600 baud, but you may have configured port "B" (the upper connector) as your console port, or you may have configured a different speed for your console. (See Chapter 8, page 61, for more information about configuration.)

`No library`
: You are either missing the CPU board library PROMs, or else they are broken.

`No Canon board`
: Either your LBP-10 interface card is missing, or the system isn't seeing it on the Multibus.

`File system: empty`
: You have file system problems (e.g., the PROM card or cards aren't visible on the Multibus), and the further messages

```
Loading IMP: not found
>>Toplevel; DDT68.0
```

will appear.

For more help with start-up problems, contact IMAGEN Customer Support.

Software Failure Messages (4.8)

The IMPRINT-10 software performs a good deal of internal consistency checking; if any of these checks fail, the system will halt with a message of the form

```
<<<Bughalt label>>>
>>Called; DDT68.1
```

and leave you in the console debugger. The *label* is simply a short name used to identify the consistency check that failed.

Something may fail which isn't caught by one of the consistency checks, and the IMPRINT-10 will "crash" with one of the following reports:

```
>>Address error for address; DDT68.level
>>Bus error for address; DDT68.level
>>Exception "name"; DDT68.level
```

followed by the address and contents of the offending instruction, where *level* is a number, usually 1.

When any of these failures occur, please notify IMAGEN Customer Support immediately, with a complete description of the circumstances leading to the failure. In some cases, the best idea is to call the Support people from the console, ready to help them diagnose the failure. In any event, they will probably request you to send the problematic document to IMAGEN on a 1/2" magnetic tape.

Once it is clear that nothing useful will be gained by staying in the debugger, you can restart the system with the following sequence of commands (your input is underlined, and *return* denotes typing a carriage-return):

```
return
;reset [Confirm] return
DDT68 (V date) (C) 1983 IMAGEN Corp
BOOT>boot: imp return
Loading imp...
BOOT>go
```

which will respond with the usual start-up messages (see section 4.7).

CHAPTER 5
Paper Handling

Copies, Collation and Reversal (5.1)

The IMPRINT-10's translation subsystem produces compactly-encoded virtual page images from its input document. The IMPRINT-10 can take these page images and print them in forward or reverse order, one or more copies per image, collated or not, as requested in the document's header with the `pagereversal`, `copies`, `pagecollation` document control items, respectively (see section 3.3.3.1, page 21 for the details.) In the case that the IMPRINT-10 runs out of page image memory when reversing or collating multiple copies, it will regain its memory by printing the current set of page images, obeying reversal, multiple copies and collation, informing the document's owner via messages on the job header page(s) (see section 7.2, page 55, for more information about job headers).

Page Numbering (5.2)

The IMPRINT-10 refers to virtual page images by their temporal index, i.e., the first page image produced is numbered 1, etc. The final order of printed pages depends on the document's paper-handling document control. For example, if a document consists of 4 page images, then if it requests 3 copies with no collation, the printed pages will ensue in the order

$1_1, 1_2, 1_3, 2_1, 2_2, 2_3, 3_1, 3_2, 3_3, 4_1, 4_2, 4_3,$

where the subscripts indicate the copy number (as referenced in any page messages on the header page or console). If it requests 3 copies with collation, the order will be

$1_1, 2_1, 3_1, 4_1, 1_2, 2_2, 3_2, 4_2, 1_3, 2_3, 3_3, 4_3.$

And, if it requests 3 copies with collation and reversal, the order will be

$4_1, 3_1, 2_1, 1_1, 4_2, 3_2, 2_2, 1_2, 4_3, 3_3, 2_3, 1_3,$

which is probably the most natural order, as the first page image will be on top of the set of printed pages, and the copies will be collated.

The IMPRINT-10 can currently hold no more than 250 virtual page images in its memory, but note that it need not hold these images unless multiple-copy collation or page reversal is requested.

Paper Types and Sizes (5.3)

Through the `paper` document control item, a document can request a given paper type by name, chosen from the set given in section 3.3.3.1, page 21. Each such paper type has an associated physical size in pixels, and the IMPRINT-10 will use this paper size in most of the languages it supports; e.g., in the Impress language, the maximum coordinates are based on the paper type's dimensions, and, in the Daisy language, lines which are wider than the paper will be truncated with a special mark. The document can also define the paper size, independently of the paper type, with the `paperheight` and `paperwidth` document control items, which take integer values representing pixel dimensions. (Note: The IMPRINT-10 always rounds paper dimensions down to multiples of 32.)

Multiple paper types may share the same LBP-10 cassette. Thus, though the IMPRINT-10 can not enforce a match between paper cassette and paper type, it will warn you each time an LBP-10 paper cassette is loaded which is clearly incompatible with the document's requested paper type (see section 7.6, page 59).

Paper Margin (5.4)

Through the `papermargin` document control item (again, see section 3.3.3.1, page 21), a document can control the way the IMPRINT-10 lays a virtual page image on the paper. With `papermargin zero`, the default, a page image's left edge will coincide with the paper's left edge. With `papermargin old`, the default when an old-style Impress job is being interpreted (see section 7.4, page 57, and section 8.6, page 63), the page image's left edge will be about 1/8 of an inch to the left of the paper's left edge (corresponding to the IMPRINT-10 pre-product's margin setting). With `papermargin visible`, the page image's left edge will coincide with the LBP-10 belt margin, which is about 3/8 of an inch to the right of the paper's left edge (this belt margin is an area not covered by the laser due to the mechanics of LBP-10 paper-handling).

Paper Safety (5.5)

The IMPRINT-10 can print pages at full speed, preserving each page image until its paper incarnation is fully cleared from the printer, and re-printing it if LBP-10 problems develop, to insure that no pages are ever lost. A document requests or refuses this "paper safety insurance" with the document control item `jamresistance` (q.v., section 3.3.3.1, page 21). In a case where a page image is extremely large, occupying all or most all of page image memory, use of this paper safety option can cause a slowdown in printing, as the next page image cannot be created until the current, large page is completely clear of the LBP-10. And, at job boundaries, use of this option will cause a slowdown, as the IMPRINT-10 will wait for the last printed page of the document to get safely out before producing any job header page images (so that it can report any problems in the job header!); further, it will wait for the job header pages to completely clear before beginning the next job.

CHAPTER 6
Communications

\014

Introduction (6.1)

The previous chapters have described the way the IMPRINT-10 operates on its own without considering how the printer communicates with the host. Now we move a step backwards: this chapter discusses the link between the printer and the host computer. This link has three parts: the hardware interface, or the electronics that sends signals between the two machines, the software interface, or the programs that transmit the data, and the communications protocol, which defines the rules used by these.

The Task (6.1.1)

Communications has an obvious purpose: to get messages from place to another correctly without being damaged by extraneous interference or by well-meaning attempts to "adjust" the message. For our printer, the equipment handling the communications must be able to do this in *both* directions, since the printer will always need to send some messages back to the host. In the simplest case, these messages will only tell the host to stop sending data temporarily ("flow-control" information); in the most complicated cases, the printer's messages will provide the host with a detailed description of the printer's status, in addition to error-correcting information reporting problems the printer has found with the data it has received.

The second problem, attempts by the host to "adjust" the data it sends, is more sticky. Some hosts will insist on altering the data they send to the printer to make it fit some preconceived notion, probably incorrect, of what is actually attached to the other end of the wire. For example, some hosts insist on adding line-feeds after carriage returns, or on converting tab characters to a series of spaces because they have been designed to operate with some particular line printer or communications terminal. They may also be incapable of storing and transmitting the document as a byte-stream, treating it instead as a "record" with some control characters automatically included. The communications protocol must protect the data from this kind of meddling, which may be "built in" to the host.

As far as the IMPRINT-10 is concerned, each document is simply a finite sequence of (8 bit) bytes. Using this definition, we can summarize the function of the printer's communications system as follows: The communications system must send the printer an uninterrupted, untouched sequence of bytes in such a way that it knows when each document begins and when it ends. As simple as this task sounds, implementing it can be much more difficult.

Byte Stream Protocol (6.2)

The byte stream protocol is the simplest protocol that IMAGEN provides. It allows the printer to communicate with a host that can transmit seven or eight bits of unaltered information per byte. When using this protocol, the printer can only transmit two simple messages from the printer to the host: it can tell the host that the printer's buffer is almost full, requesting that the host stop transmitting briefly, and it can ask the host to resume sending data. Since this protocol has no ability to correct or detect errors, we do not reccomend that you use it over long or noisy data lines. This protocol can be used with either of the two byte-stream interfaces we supply for the IMPRINT-10, the Centronics (8-bit) parallel interface and the RS232C serial interface. (See section 8.5.1, page 62 for details on how to configure these interfaces.)

The byte stream protocol consists of these rules and definitions:

- *data width*: Data can be interpreted as either 7 bits or 8 bits "wide," depending on whether the host in question can send 8 bits of untouched data. For example, if a host insists on adding a parity bit to serial output data, then you would have to choose 7 bit data, and "quote" any data byte values greater than 127 (see below). If the host does not add a parity bit, you should send complete 8-bit data; since 8-bit data requires less quoting, your communications will be more efficient.
- *unprintables*: Whichever data width you use, this protocol allows you to ignore unprintable byte values (those less than 32 in value, or equal to 127) if the host insists on inserting unprintables into the output stream for some reason. For example, some hosts may "wrap" what they believe are long lines of output with a carriage-return and line-feed or "pad" output with nulls (0) or rubouts (127) at the end of a line. In this case, the byte stream protocol will have to be configured to ignore unprintable byte values. Note that being forced to ignore unprintables will reduce the bandwidth of the byte stream even more, as all such bytes will need to be quoted (see below). Also note that the notion of "printable" is an ASCII character set issue, but that the document bytes being sent don't necessarily have anything to do with ASCII; i.e., they could be arbitrary binary data.
- *quote character*: Because there is at least one reserved character, the end of document marker, and because documents are sequences of arbitrary 8-bit bytes, some "quoting" must be performed to get this end of document character through as ordinary data. Thus, we have a quote character, which must also itself be quoted. To get an arbitrary document byte *b* over the byte stream interface, the host can send the quote character, followed by two bytes representing the value *b* as a hexadecimal number; each of these bytes, high-order first, is a digit from 0 to 9 or from a to f (representing hexadecimal "nibble" values 10 to 15). The character used for quoting depends on your configuration and, specifically, on whether or not your host can transmit 7 or 8 untouched bits in each byte. The quote character to use in either case is given in section 8.5, page 62.
- *end of document character*: The byte stream protocols, like all others, require some indication of an end of document. In this case, a single character is assigned to mark the "end of document." As with the quote character, the character used for the end of a document depends on your configuration. Again, see section 8.5, page 62 to find out what charactr to use in any situation.

The IMPRINT-10 handles an 8-bit data byte appearing on either type of byte stream communications interface in the following fashion:

- if the data width is 7 bits, the most-significant bit of the 8-bit byte is set to zero, and this value is used below;

- if the data byte is the end of document character, then document input is ended, and the current byte is otherwise ignored;
- if unprintables are being ignored, and this byte value is less than 32 or equal to 127, then it is ignored;
- if the data byte is the quote character, then the following two data bytes are read and interpreted as the printable hexadecimal representation of the actual document byte, high-order nibble first, and the byte so represented is taken as a document input byte, in place of the three data bytes read (if either of the nibbles is not a hexadecimal digit, then it is treated as 0 and an error is counted);
- otherwise, the data byte is used directly as the document byte.

Therefore, a spooler designed to send information to the IMPRINT-10 from your host using the byte stream protocol must take the data to present, "quote" it properly (of course, proper quoting depends on the configuration of your system, as explained above and in section 8.5), and send it to the printer over your data line. If this is done properly, the host will not be able to interfere with communications to the printer; the protocol allows the printer to ignore any characters the host might want to insert.

Printer State (6.2.1)

Neither of the two byte-stream interfaces allow the printer to report its state to the host; however, they will use ordinary flow control (see next section) to guarantee that no data is lost when printer problems occur.

Hardware (6.2.2)

Neither of these interfaces provide any error detection or correction. Flow control is achieved in the parallel interface by virtue of hardware handshaking, or, in the serial interface, by either an XON/XOFF or a CTS-toggling protocol.

The XON/XOFF flow control protocol is as follows. When the IMPRINT-10 decides that it is getting low on input buffer space, it sends an ASCII XOFF character (control-S), to which the host should stop sending "as soon as it can" (128 characters is a nominal upper bound on how many characters should be sent by a host after it receives an XOFF). The host should only resume sending when it receives an XON character (control-Q) from the IMPRINT-10.

The CTS-toggling flow control protocol is as follows. Normally, the clear-to-send signal (CTS, pin 5) on the RS232C connector is "high," indicating that the IMPRINT-10 is ready to receive data. When it detects that its input buffers are nearly exhausted, it will make CTS go "low" to indicate that the host should stop sending data "as soon as it can," with the same tolerances as the XON/XOFF protocol above. The IMPRINT-10 will once again bring CTS "high" when it is ready to receive more data. (Note that to use this CTS-togging protocol, you will need to use the "A" connector for host communications, and thus will need to swap the normal host/console serial port assignments. See section 8.3, page 61, for the details.)

Packet Protocol (6.3)

In addition to these simple protocols, IMAGEN provides an error-detecting protocol for the IMPRINT-10. This protocol allows the printer and the host to communicate with each other, letting the printer report its mechanical status and acknowledge receiving data correctly. This protocol places some requirements on the host computer. It must be able to:

- provide an 8 bit full-duplex no parity data line operating at 19,200, 9600, 4800, or 1200 baud (a hardware requirement)
- receive full duplex data from the printer (a hardware requirement)
- receive asynchronous input from the printer while sending data (an operating system requirement)
- receive and send eight bit binary data without any modification (an operating system requirement)

Principles of operation (6.3.1)

With the previous protocols, the printer did little besides receive data from the host. It could only transmit the simplest information back to the host (i.e. whether or not its buffer is full). The packet protocol gives the printer a much more versatile way to send reports to the host. Since the printer can store a lot of data, it can receive a large "batch" of data from the host, check for errors, request that the host repeat any portions of the data that it has not received correctly, and later insert these portions into the document correctly. The protocol is a basically a set of rules describing how the printer and the host can communicate -- how the printer can tell the host what problems there have been in the data, and how the host can, at its own leisure, "repair" those problems.

With the packet protocol, we transmit information to and from the printer in chunks called packets, which are blocks of up to 128 bytes. There are two kinds of packets: *data* packets and *status* packets. A data packet contains, simply enough, data; it contains the instructions that tell the IMPRINT-10 how to print your document. In addition, each data packet also contains a *packet number* and some other information the protocol needs to operate properly. Status packets carry information about the job's progress between the host and the printer. The host sends status packets to request information from the printer or to tell it that a new job is beginning; the printer sends status packets to report its status, to acknowledge that it has received data packets correctly, and to report that it has received data which is incorrect.

Now we can summarize how the packet protocol works. After sending some data packets, the host sends a status packet asking the printer to report its state (this is called an RSVP packet). The printer has two possible responses. First, it can send a status packet acknowledging that it has received all packets through a certain number n correctly; a packet sending this message is called an "information" packet. Second, it can send a status packet stating that it has received all the packets up to and including n correctly, but that something is wrong with the next, $(n+1)$. This is called a "NAK" packet. In the latter case, it is the host's responsibility to transmit the bad packet again, which the printer will automatically insert at the correct place in the sequence. After it has retransmitted the bad data packet, the host can resume sending data packets where it left off, request more information with an RSVP packet, or retransmit everything following the bad packet. In this case, the printer will ignore any duplicate packets it receives.

The printer will also periodically transmit information packets of its own accord; it will not wait

indefinitely for a request from the host. The printer will automatically send a status packet if thirty-two data packets arrive from the host without any requests for information; the printer will also send an information packet if it has some message the host ought to know about immediately. Such messages report that the printer has received some bad packets, that it is out of storage and cannot receive any more data packets, or that the printer has stopped printing for some reason (e.g. it is out of toner or someone has opened the front door). In any case, the host must be able to respond appropriately -- by waiting before sending more data packets, or by printing a message telling you to fix the printer.

Both the host and the printer use one additional kind of status packet, called a "sync" packet, at the beginning of jobs. When the host sends this packet, it is telling the printer that the next data packet will begin a new job and will be number 0. The printer responds to a "sync" packet when it actually achieves syncronization; that is, when it is ready to receive a new job. The host can also send a sync packet in the middle of a job if, for some reason, it wants to abort the job and begin a new one. The printer will also use a sync packet if the host requests information before it has sent the first data packet in a new job (i.e. if the host sends a "sync" packet and then one or more "RSVP" packets before sending any data).

Packet structure (6.3.2)

Each packet, no matter what kind of packet it is, has the same format. Each begins with a "start packet" character, followed by two bytes giving the packet's length, followed by the information the packet is carrying, and ending with a four byte checksum followed by the "end packet" character. We can summarize this sequence as follows:

Start packet / Length / Information / Checksum / End packet

The two special characters, "start packet" and "end packet," are the same for both status and data packets. Similarly, the packet's length and the packet's checksum are encoded the same way for any packet. However, since data packets and status packets do significantly different things, the "information" sections differ significantly.

Except for the "start" character, the "end" character, and any special characters that may occur in the "information" section of a data packet, *all* the information included in any packet will be coded as printable ASCII characters (i.e. chaacters with codes between 33_{10} and 124_{10}). For example, instead of sending a sixteen bit binary checksum, we send the checksum as four upper-case ASCII characters representing the sixteen bit checksum in hexadecimal. While this is slightly less efficient, it guarantees that we will not have to worry about "start" and "end" characters occuring within the packet. In addition, this makes it less likely that the host will do something to damage the packet, like adding an automatic line-feed after the code 13_{10} (carriage return).

"start" and "end" characters *(6.3.2.1)*

The "start" character is the byte 176_8 (in hex, $7E_{16}$; decimal 126_{10}). The "end" character is the byte 12_8 (in hex, A_{16}; decimal 10_{10}). These two special characters may *only* occur at the beginnings and ends of packets; they are not allowed within packets. The coding for status packets has been designed so that this will not be a problem; no legitimate status packet will ever need to use either of these characters. In data packets, however, you may need to send these characters in the "Information" segment. To handle this case, we have developed quoting conventions that let you represent these values with other characters. These conventions are described in section 6.3.2.4.

Packet "length" *(6.3.2.2)*

We use two bytes to represent the length of any packet. These bytes are two ASCII characters that represent, in hexadecimal, the number of bytes in the packet. For example, if a packet is 26 bytes long (decimal), the "length" section of the packet will be the two ASCII bytes representing "1A."

Checksum *(6.3.2.3)*

Like the length, the checksum is also encoded as printable ASCII characters: it is given by four bytes representing, in hexadecimal, the checksum of the entire package. We use the following algorithm to produce the checksum for each packet:

```
checksum := 0;  --initialize checksum
--go over the packet byte by byte, computing a temporary checksum
for temp_index := 1 to length - 5 do --skipping the 4 checksum bytes
     checksum := rotateleft16(checksum) + packet[tempindex];
--now add the end-packet character to the checksum
checksum := rotateleft16(checksum) + packet [length];
--checksum now holds the final value for the checksum
```

In this routine, `rotateleft16(x)` is a function that takes `x` as a 16-bit value, rotates it one bit to the left, and returns the rotated value. Note that this is a 16-bit *rotation*, not a shift, and that the rotation *must* occur before the addition. `length` is the total length of the packet, in bytes. `Packet` is the packet itself; it is an array of `length` bytes, subscripted from 1 to `length`.

This algorithm ignores the four checksum bytes (`packet[length-4]` to `packet[length-1]`) in its computation; however, it does take into account both the "start packet" and "end packet" characters. After running this algorithm, `checksum` holds the value of the checksum, which must then be converted into four ASCII characters representing its value in hex and, finally, inserted into the packet.

Data packets *(6.3.2.4)*

The "Information" section of a data packet contains the following information:

Type / Packet number / Mark / Databyte_1 / . . . / Databyte_n

In this packet, *type* is the ASCII character "D." This byte tells tells the processor that this packet is a data packet, not a status packet. The *Packet number* is a two byte field that gives the packet's sequence number. Sequence numbers begin with 0, at the beginning of a job, and increase by one with each packet. We do this numbering modulo 128; the next packet after packet 127 will be packet 0. Only data packets have packet numbers; status packets aren't counted. The packet number must be converted into tw upper case ASCII characters giving its value in hex. For example, if a packet is the 15th data packet sent in a job, this field will contain the ASCII characters "0F." The next field, *Mark*, tells the processor whether or not the current packet is the last in a job. If it is, this byte should contain the ASCII character "B"; otherwise, if this packet occurs anywhere else in the job, this should contain the character "1."

The final group of bytes contains the data you want to send to the printer. There are `length - 12` data bytes in each packet. The bytes in this field may contain any values *except* 176_8, 12_8, and 175_8; these are the "start" and "end" characters which can only appear at the beginning or the end of a packet, and the special character used for quoting. If you need to include either of these values as data, we provide this "quoting" convention: Each quoted byte must be replaed by a sequence of two bytes. The first byte in the series is always 175_8 (or, in hex, $7D_{16}$; decimal 125_{10}). The second byte in the sequence depends on the value you want to send: for 176_8, use "A"; for 12_8, use "B." Finally, if you need to send the quote character (175_8), send the two byte sequence 175_8, "C."

Example: If you want to transmit the data bytes (represented in octal)

```
000, 001, 012, 010, 0175, 000, 022
```

you would have to transmit the slightly longer sequence

```
000, 001, 175, 102, 010, 175, 101, 000, 022
```

In the second sequence, the bytes 175, 102 now represent the illegal character 012, and 175, 103 represent the quoting character 175.

Any quoted sequence must be contained completely within a packet. You cannot separate the quote character from the letter following it. If the current packet cannot contain both both bytes, then you must put the quoted character in the next packet, or do something else to reduce the size of a packet.

Example: A packet ending with

(115 data bytes) / 175_8 / ''A'' /checksum / end packet

is illegal because it would require a packet 129 bytes long, after the checksum, length, etc. have been added. However, you cannot send the 128 byte packet ending with

(115 data bytes) / 175_8 / checksum /end packet

because the processor will "forget about" the quote character (175_8) before it sees the "A" at the beginning of the next packet. Instead, you must shorten the current packet to 127 bytes, placing the quoting character, together with the character "A" at the beginning of the next packet. This means that the beginning of the next packet will look like this:

start packet / length / ''D'' /''Number'' / ''1'' / 175_8 / ''A'' / etc.

Status packets *(6.3.2.5)*

While status packets have the same overall structure as data packets, they carry different information and therefore require a differently structured "information" section. This section contains the data reporting the printer's status. It has five fields, which are structured as follows:

Type / Flavor / Window / Acknowledge / Error type

- The field "Type" must contain the ASCII character "S," showing that this is a status packet.
- "Flavor" is also a single byte containing a lower-case ASCII charater. It can have one of four values:

  "i" signifies that this packet is an information packet; these packet report the state of the printer and acknowledge that the printer has received data correctly. This kind of packet will not be sent if there are outstanding errors -- not magnificent errors, but errors which have not been corrected yet.

  "r" signifies an RSVP packet. The host generates these packets to request that the printer send it information. The printer will respond with an information packet, a NAK packet, or a sync packet, depending on the context. In an RSVP packet, the "window," "acknowledge," and "error type" fields aren't used.

  "n" signifies a "NAK" packet ("NAK" stands for "not acknowledged"). The printer will generate NAK packets to report that it has received packets incorrectly, and that the host has not yet sent replacements.

  "s" signifies a "sync" packet. Both the host and the printer generate these packets to start new jobs. When the host sends a "sync" packet, it is telling the printer to begin a new job, beginning with packet number 0. When the printer sends a "sync" packet, it is telling the host that it is ready to begin a new job, and expects the next data packet to be packet number 0.

- The "window" field shows how much space remains in the printer's buffer for incoming packets. This field is four bytes long, and contains the ASCII representation of the number of bytes remaining in the buffer (in hexadecimal).

  Regardless of its length, each packet requires 124 bytes. When there is only enough space for one packet left, the host must stop sending data packets and wait for the printer to process some of the data. While it is waiting, the host can send RSVP packets as often as it likes to find out when it can begin sending data again. The printer will not allow you to fill the last 124 bytes in the buffer, and will not acknowledge any data packet you send when the buffer is too small.

  Remember that the printer does *not* respond to every data packet it receives; it is the host's responsibility to keep track of the space remaining in the buffer between status reports from the printer. If you fill the buffer, the printer *will* respond to any incoming data packet with a "NAK" packet, showing that it has not been able to accept the incoming material. If the host insists on sending data packets when the window is full, it will receive a barrage of "NAK" packets for the printer. With some hosts, this barrage may cause problems; with any host, it wastes time, so don't lose track of the window's size.

  This field will be ignored in any status packet that the host sends; the printer does not expect the host to buffer its data, and will not wait for the host if its buffer gets full. In packets sent by the host, fill this field with ASCII "FFFF."

- The "Acknowledge" field is two bytes long. It contains, in ASCII, the hexadecimal number of the last packet received correctly by the printer. Like the previous field, it makes no sense for the host to acknowledge the printer's packets; it only sends status packets, which aren't numbered. Therefore, the host should fill this field with ASCII "FF" in any packets it sends.

  Exactly how the host should interpret this field depends on what type of packet it is receiving. In a "sync" packet, this field can safely be ignored. In an "information" packet, this field confirms that the printer has received packets all the packets up to and including packet "n" correctly; therefore, the host can discard these packets, since they will not need to be retransmitted in the future. In a "NAK" packet, this field implies that the printer has received all the packets through n correctly, but that there is some error in packet $n+1$. Therefore, in a "NAK" packet this field explicitly requests that the host resend data packet $n+1$. The host can respond by sending a single packet, or by re-transmitting all the packets following the bad one. Which response to take depends on your taste in spooler design. For more information, see section 6.3.3.3.

- The "error type" field in a status packet is a single byte containing a printable ASCII character that reports the physical status of the printer -- whether or not it has sufficient supplies, and whether or not it is waiting for a job. Only "sync" and "information" packets use this field; in NAK packets, this byte will contain nothing of interest. When the host is sending a packet, it should again fill this field with ASCII "F."

  This character is encoded as six binary bits, to which we add 32_{10}, guaranteeing that the result will be a printable character. The six binary bits are encoded as follows:

000001_2	paper jam
000010_2	out of premix
000100_2	out of toner
001000_2	out of paper
010000_2	there is a job in progress (if this bit is zero, there is no job in progress, or the current job is being aborted.)

100000_2 there is a job being aborted (if this bit is set, the previous bit will be zero: no job in progress.)

The special code 001111_2 indicates that the printer is offline (i.e. turned off). Again, to decode this byte, subtract 32_{10} and look at what is left. For example, if this byte contains the ASCII character "&," in decimal 38_{10}, you would subtract 32, leaving 6_{10} or 110_2; this indicates that the printer is waiting for a new job, that is out of premix, and that it is out of toner. If this byte contains ASCII "H," which is decimal 72_{10}, you subtract 32, getting 40_{10}, which is 101000_2. This means that the current job is being aborted, and that the printer is out of paper.

Using the packet protocol (6.3.3)

When, and why, the printer generates status packets *(6.3.3.1)*

Throughout this discussion, we have noted many conditions which will make the printer generate a status packet. Here, we summarize this information. The printer will generate a status packet

- if it has received 32 data packets from the printer since the last status packet it has sent. Under these circumstances, it will send and "information" packet, if it has received all the data packets so far correctly, or an "NAK" packet showing the last packet correctly received if there have been errors in the transmission. This feature makes it difficult for the host to "ignore" the printer completely; the printer will insist on giving a report periodically, even if everything is going well.
- if the host generates any kind of packet when the printer's buffer is full. These will always be "NAK" packets, warning the host that the printer isn't listening. Note that this condition will probably create a barrage of "NAK" packets, since every incoming packet will force a response. This may create problems at the host.
- if "something happens" to the printer; i.e. someone has turned it on or off, opened or closed on of the access doors, or it has run out of paper, toner, or premix. These packets will be "sync" packets at the beginning of the job, or "information" packets in mid-job; they will *never* be "NAK" packets, even if there are outstanding errors, because they cannot transmit this information correctly. However, this won't cause any problems; an "information" packet will *never* mislead you by acknowledging incorrect data.
- in response to a host-generated "RSVP" packet. Under these circumstances, the printer will generate an "information" packet if there are no outstanding errors (i.e. errors in data packets that the host has not yet corrected), a "NAK" packet if there are packets that the host needs to retransmit, and a "sync" packet if the printer is awaiting the beginning of a new job. The printer will be waiting for the beginning of a new job if it has received a "sync" packet from the host, *or* if it has received a data packet with the mark set to "B," indicating that that packet was the last in the job.
- in response to a host-generated "sync" packet. The printer will always respond with another "sync" packet as soon as it it is ready to accept a new job -- that is, after resetting the packet number to 0 and finishing the current job. If a "sync" packet comes before the last packet in the job (a data packet marked "B"), the job will be aborted and the printer will be ready to accept a new job.
- whenever it receives a data packet with any sort of error (a checksum error, an incorrect packet length, etc.). In this case, the printer will respond with a "NAK" packet showing the last packet correctly received. Note that this applies only to packets with checksum errors, not packets sent out of order -- even if they're sent out of order by mistake.

Sending data packets *(6.3.3.2)*

When you are forming and transmitting data packets, you should obey two rules: First, don't transmit more than 64 packets at a time without replacing any bad packets. This guarantees that the packets you send will have unique numbers. Second, be careful not to fill the buffer for incoming data packets. This doesn't require you to bombard the printer with "RSVP" packets; you can keep track of the buffer's size by subtracting 124 bytes each time you transmit a packet. Since the printer will be emptying the buffer while you are filling it, your estimate of the buffer's size will always be somewhat "conservative" – which is fine. Remember that, regardless of their size,all data packets occupy 124 bytes in the buffer. When there are only 124 bytes left in the buffer, stop sending new data packets and wait for the buffer to empty.

Responding to status packets *(6.3.3.3)*

The host's reaction to various kinds of status packets are varied and often depend on the context. An "information" packet always means that the host can discard data packets with sequence numbers lower than or equal to the sequence reported in the "acknowledge" field. These packets have been received correctly, and therefore do not need to be kept around. In addition, a "NAK" packet implicitly acknowledges that all packets with numbers up to and including the one specified have been received correctly; again, you can discard these packets.

However, the primary function of a "NAK" packet is not to acknowledge data received correctly, but to report problems. A "NAK" packet means that there is a problem with the data packet numbered one *higher* than the sequence number reported, and possibly other errors with packets having higher number. The host can respond to this in two ways. Either it can retransmit the problematic packet by itself – which the printer will insert into the document correctly, even though it is out of order – or it can go back to the problematic packet and retransmit it along with everything that came later. Since the printer will discard any packets that duplicate packets already received correctly, this will not waste space in the buffer, nor will it cause duplication problems. In either case, the host should follow any replacement packets with an "RSVP" packet, giving the printer a chance to report any further problems. Bad packets need not be replaced immediately; if the printer gets to where it needs to print a bad packet in order to continue the job, it will wait until the packet has been replaced.

For some messages, the appropriate response for the host is to wait for a while and then send an "RSVP" packet to find out the printer's status. This is the appropriate response if the printer reports that its buffer has less than 124 bytes. If the printer reports that it needs some human intervention before it can continue printing (e.g. adding toner), you can continue to send it more data; but if the problem isn't corrected, the printer will eventually run out of space in the buffer. In this case, you should print a message telling someone to service the printer. Note that any reports about the toner level, etc. *must* come from "information" or "sync" packets; "NAK" packets cannot carry this kind of information. For recommended waiting times before getting a new report on the printer's status, see section 6.3.3.5.

Beginning and ending jobs *(6.3.3.4)*

Beginning a job is simple; the host sends a "sync" packet, waits for a returning "sync" packet from the printer, and then sends data packets, beginning with packet number 0.

It is also easy to end a job: send a data packet marked "B." This packet could conceivably be empty; for example, you could transmit all the data you need for the job, then follow it by a single "B" packet with no data. This packet essentially tells the printer the highest last packet number the job will use; sending a "B" packet does *not* prevent you from going back and re-sending other packets that the printer has not received correctly. When the printer has received the entire job correctly, it will be ready to begin the next.

Suggested delays *(6.3.3.5)*

If the printer is not ready to receive more data for some reason, the host should wait a few seconds then send an "RSVP" to find out whether or not things have changed yet. The ideal time to wait before "polling" the printer again depends on what is causing the delay. For each possible problem, we recommend these intervals between successive "RSVP" packets:

No room in the buffer

Poll the printer every second. The buffer empties very quickly. It also runs out of room fairly often, so you don't want to waste too much time waiting.

Printer needs service

Poll the printer every second. After the problem has been fixed, you will want to begin printing again immediately. Note that there is no reason to stop sending data packets if the printer needs service, since the printer can prepare pages for printing for a long time before running out of memory. Of course, if the problem with the printer isn't fixed, it will eventually run out of memory, forcing you to stop sending packets.

Printer off-line

Poll the printer every three to ten seconds if you either get an "off-line" message or the printer does not respond to an "RSVP."

Examples (6.3.4)

Here are two examples of packet protocols. The first example is the simplest protocol possible; the second has been derived from the protocol used by some of IMAGEN's spoolers. Both of these are *only* examples; they are not "right," or even necessarily "best."

A simple protocol *(6.3.4.1)*

In this protocol, the host sends an "RSVP" packet after every data packet and waits for a confirmation before proceeding to the next packet. While this is not at all efficient, it guarantees that the printer gets everything. The outline below summarizes this protocol:

(1) confirm that the printer is responding

(2) send a data packet to the printer

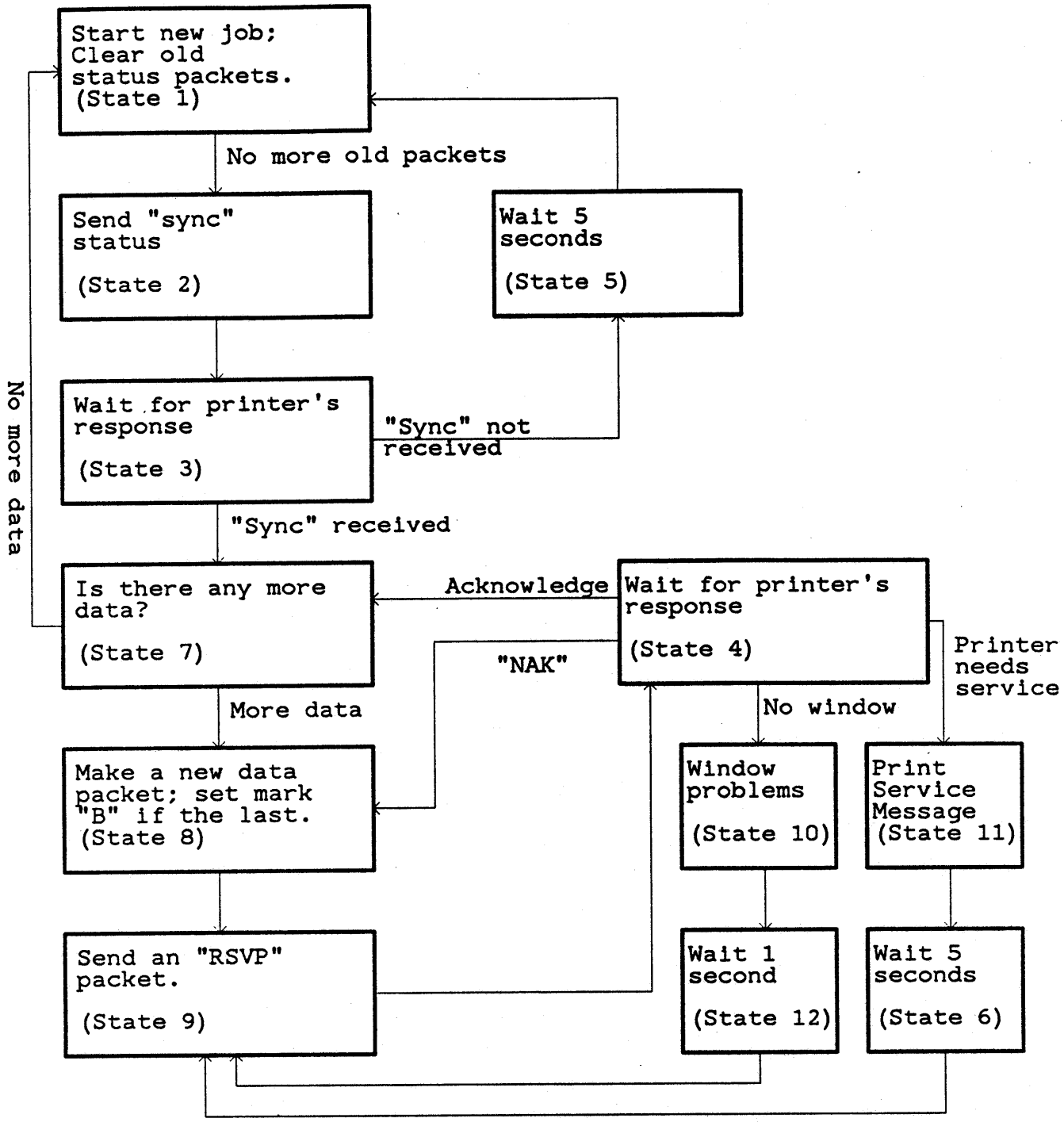
(3) send an "RSVP" packet to the printer

(4) wait for the response

(5) decode the status packet and respond accordingly by either waiting, resending the last data packet, or sending a new packet (step 2)

(6) clean up the job after the last packet

The following state diagram shows this protocol in greater detail:


Start new job; Clear old status packets. (State 1)
No more old packets
Send "sync" status (State 2)
Wait 5 seconds (State 5)
Wait for printer's response (State 3)
"Sync" not received
"Sync" received
No more data
Is there any more data? (State 7)
Acknowledge
Wait for printer's response (State 4)
"NAK"
Printer needs service
No window
More data
Make a new data packet; set mark "B" if the last. (State 8)
Window problems (State 10)
Print Service Message (State 11)
Send an "RSVP" packet. (State 9)
Wait 1 second (State 12)
Wait 5 seconds (State 6)

Simple packet protocol

A more complex protocol *(6.3.4.2)*

Here is a more complex example of how to use the packet protocol. It would run much faster than the previous example, because it takes advantage of the printer's buffer. As a consequence, it must keep track carefully of the buffer's size at any time. This spooler also adjusts the number of packets it will send at any time, so that it will not bombard the printer with data if the communications lines between the printer and the host are bad. This requires that the spooler keep track of the queue's maximum length, and that there be some specified "absolute" maximum length. In outline, the spooler looks like this:

```
Clear the queue of messages from the printer.
Send a "sync" packet and wait for the printer to return "sync."
        If the printer does not respond after a certain time, print a message.
        Then try again.
Initialize the file you want to send (add document control instructions to
the beginning, etc).

Main loop: Continue until there is only one data packet left.
        Send any current data packets (there will be none on the first
          iteration).
        Send an "RSVP" packet.
        Make enough data packets to fill the buffer, saving space for one packet.
        Packet-formation loop: Exit when buffer size is zero or there is
          only enough data left for one packet or the queue of data packets
          reaches its maximum length.
                Increment packet number by one; decrease buffer size by one.
                Make a data packet with mark set to "1" and add it to the
                  queue.
        End packet-formation loop.
        Inner loop: Check queued messages from printer. Exit the loop when all
          the messages have been processed.
                Test the checksum, length, etc. of the messge. Ignore it if it
                  has not been received correctly.
                Look at the next message to find what has been received correctly.
                Clear data packets received correctly from the queue.
                Update buffer size.
                If a "NAK" packet:
                        Add the bad packet to data packet queue.
                        To save time if there are communications problems,
                          cut the maximum length of the data packet queue in
                          half.
                If an "information" packet:
                        Figure out printer's status.
                        If the printer needs help, print a message, send an
                          "RSVP" packet, and wait.
                        If the maximum length of the data packet queue is not
                          its greatest possible value, increase it by one.
        End inner loop.
End outer loop.
```

Finishing the job: We need to make sure that the printer has gotten all the data correctly before sending the final packet.
Loop: Exit when the printer responds with an "information" packet.
    Send an "RSVP" packet. Wait for the printer's response.
    Test the checksum, length, etc. of the printer's response.
    If the response is a "NAK" packet, resend the bad packet.
End loop.

Send the final data packet, marking it "B" to signify that the job is done.
Loop: Send an "RSVP" packet to verify that this packet has been received correctly. Exit when the printer responds with an "information" or "sync" packet.
If the response is a "NAK" packet, resend the final packet.
End loop.

Clean up and terminate.

TCP/IP on Ethernet (6.4)

The IMPRINT-10 TCP/IP on Ethernet implementation fully implements the U.S. Federal Government packet communications standard TCP/IP [ref. TCP, IP] using Ethernet [ref. Ethernet], as the link-level communications medium, with IP packets encapsulated as Ethernet packet type 2048 [ref. Numbers].[4] The only current TCP option, maximum segment size, is implemented. All IP options are ignored, correctly. The ICMP [ref. IP] implementation supports the minimally sufficient set of ECHO and REDIRECT packet types; redirects based on type of service are not supported. Stale ICMP routing information is removed after an hour.

User Interface (6.4.1)

The interface provided to an IMPRINT-10 user by the TCP/IP communications option is a single-connection TCP/IP, available only on well-known port 23 (Telnet, [ref. Numbers]). If the IMPRINT-10 is busy when a host attempts to open a connection, the attempt will be rejected with a TCP RESET. Otherwise, the connection will be established. Once the connection is established, the host should send the document to be printed as a sequence of bytes (optionally preceded by a document header; see Chapter 3), closing the connection afterwards.

The IMPRINT-10 will dally for 5 seconds after either side closes the connection.

Currently, no data is sent to the host over an open connection.[5]

[4] Note that the Berkeley 4.1c/4.2 Unix trailing-IP-header packet type is *not* supported, but probably will be in the future, even though it is a violation of the encapsulation protocol.

[5] In the future, accounting data will be sent to the host over the connection once the host closes its side.

Address Resolution Protocol (6.4.2)

The IMPRINT-10 fully implements the HARP (Host Address Resolution Protocol) for Internet addresses and Ethernet hardware, currently a proposed standard [ref. HARP]. However, even if HARP is not supported on the Ethernet to which the IMPRINT-10 is attached, the IMPRINT-10 will be able to communicate with directly-connected hosts (by performing Internet-to-Ethernet mapping based on incoming packets).

Pre-installation Configuration (6.4.3)

Currently, the customer must supply IMAGEN with the Internet address of the IMPRINT-10, and with the Internet address of the default gateway on the same Ethernet as the IMPRINT-10. These addresses are stored in EPROM, and thus, in most cases, require a field visit to change.

References (6.4.4)

The relevant standard documents are:

TCP USC-ISI, *Department of Defense Standard Transmission Control Protocol.* Technical Report RFC 793, DARPA, January, 1980.

IP J. Postel, *Department of Defense Standard Internet Protocol.* Technical Report RFC 791, DARPA, September, 1981.

Ethernet
Xerox/Intel/DEC, *The Ethernet/A Local Area Network/Data Link Layer and Physical Layer Specifications.* September, 1980.

HARP
D. Plummer, *An Ethernet Address Resolution Protocol.* Technical Report RFC 826, MIT, November, 1982.

Numbers
J. Postel, J. Vernon, *Assigned Numbers,* Technical Report RFC 820, USC-ISI, January, 1983.

CHAPTER 7
Messages

Introduction (7.1)

This chapter describes the set of messages which the IMPRINT-10 can produce during document processing, and which are presented to the document's owner on the job header page(s). These messages are distinct from the set of console messages (see Chapter 4, page 25) which are relevant to system operation and control from the console.

Each document language interpreter can also produce messages relevant to documents written in that language; see the Chapter for the language in question, section "Messages."

See Chapter 3 for information about the document control options relevant to message processing, and section 3.3.3.2, page 22, in particular.

Job Header Pages (7.2)

After the IMPRINT-10 has completely finished processing a document, it will produce a set of *job header* pages if the document requests it, or if an error occurs during document processing.[6] This page or set of pages records, first, the given document control information, and, then, all messages produced during document processing.

To help sort paper in the LBP-10's output tray into separate documents, the job header pages are striped on the edges with two vertical gray bars for the entire length of the paper, making them easy to spot.

Document Control Items (7.2.1)

The document control information item pairs are printed on the last, or top-most, page of the job, in a highly visible font (see Chapter 3, page 19, for more information about document control). This makes this top-most page useful as a "banner page" for sorting documents into bins or groups, based on the owner's name, or whatever information the spooling host might supply. (If there are no document control items, because of problems with the document, then the single message **(No document control information)** will appear.)

Document control items pairs can either supply information to the system for use during document processing--such as the document language specification--or can simply be comments useful

[6] These pages come out in the "right" order, i.e., with the first job page printed last, or on top. In the case of a large number of header pages, the system may have to print these pages out of order.

to the operators and document consumers--such as the document's name, source, and owner. Since the IMPRINT-10 will not complain about a misspelled document control item, but will treat it as a comment, the job header page marks those items considered comments with surrounding brackets (eg, **[Name]**). These comment markers can help you find problems with the document control specification.

Message Format (7.2.2)

After the document control items, the IMPRINT-10 lists the job messages in a smaller, typewriter-style font. And, after the final job message, the system prints a a small gray square to signal the end of messages.

The first message listed, if there are any job messages, totals the number of message occurrences, mentioning any which are omitted due to lack of space, viz.:

`Number of job messages:` *# messages[,* `omitted:` *# omitted]*

Then, each message is listed, followed either by a set of lines detailing each occurrence of the message (what we will call *occurrence data*), if the document control option `messagedetail on` is used (or if the message always includes its occurrence details), or else by a count of occurrences, if message detail has not been requested; if there is no room to record some number of occurrences, then the number of omitted messages is also noted.

If the message is page-specific, the occurrence list will include the page number; if it is also copy-specific, the copy number will be listed. Page numbers begin with 1 for the document's first page image, up to $n+1$ for any partially-completed page image (where there are n completed page images), and copy numbers are dependent on the paper handling requested for the document (see sections 5.2, page 37 for more detail about page and copy numbering). Any reference to a partially-complete page will include the tag `(unprinted)`.

Note that because of potential page loss or duplication in the face of printer problems, depending on whether the job has elected `jamresistance` (see Chapters 5, page 37, and 3, page 19), there is no guaranteed correspondence between the physical sheets of paper in the output tray and the header's page and copy numbers; that is, the IMPRINT-10 can never know exactly which pages which made it into the output tray and which didn't. However, any `LBP-10 problems:` reprinting messages (see section 7.6, page 59) give you an idea of where the lost or duplicate pages will be.

Here is a sample job message list which illustrates some of the above:

```
Number of job messages: 41
Flushed leftover document bytes:
  22
Glyph off page: (40, 34 omitted)
  Page 3: [2300 20]
  Page 3: [2310 10]
  Page 3: [2533 21]
  Page 3: [2600 25]
  Page 6: [-2 100]
  Page 8: [100 5020]
```

The total number of messages is 41, where the third message includes 6 detailed occurrences on 3 different pages (3, 6, and 8), and 34 omitted occurrences (presumably due to lack of space (admittedly not very realistic)). The second message has only one occurrence (by nature), which will always be listed regardless of the `messagedetail` document control setting (again, by

nature). Note that the number of occurrences is not listed for the second message, as there is only one.

Some messages are fatal errors; they will appear prefixed with `Fatal error:` and document processing will stop sometime soon after their occurrence (there is some asynchrony involved). Some more serious error messages originate from the programming machine which supports the language emulators, called "IVM"; these messages will appear prefixed with `IVM Error:`, and are always fatal errors, as their occurrence will stop document processing immediately.

Communications Subsystem Messages (7.3)

`Host requesting synch; ending document input`
: The communications subsystem finds that the host is requesting re-synchronization before completely sending the previous document; thus, it is artificially closing off the current document before starting to accept the next.

Document Control Subsystem Messages (7.4)

The IMPRINT-10 document control interpreter subsystem can produce the following messages:

`Assuming old-style document structure and Impress language`
: The current document is entirely missing document control information, and the system is configured to assume that such a document is, in fact, an old-style Impress "job"; this provides document compatibility with the pre-product IMPRINT-10 software. The IMPRINT-10 further assumes the document control `papermargin old`.

`Illegal character in document control information` (fatal error)
: The character shown in any occurrence data (or, rather its value in decimal shown) is not permitted in document control information. (The most likely cause is unterminated document control information, which further causes document data to be interpreted as document control.)

`Item in document control information too long` (fatal error)
: The partial item shown in any occurrence data is the first part of an excessively long item; this error is fatal on the assumption that some document control information wasn't properly terminated, and the offending item includes some of the document body.

`Non-numeric value in document control information (ignored)`
: The value shown in any occurrence data, appearing as the value of a keyword taking a numeric argument, is not a decimal number (i.e., it is something other than a string of decimal digits optionally preceded by a minus sign).

`Not enough memory for document control information`
: There is not enough room to store the item displayed in any occurrence data. This may cause later problems if the item is crucial to successful document processing.

`Numeric value out of range in document control information (ignored)`
: The decimal value shown in any occurrence data is outside the range $[-2^{16}, +2^{16})$.

`Unexpected end of document in document control information` (fatal error)
: The document ended unexpectedly in the middle of document control information. (Mostly likely caused by improperly terminated document control information, such as an unclosed double-quote (`"`), or a missing final right parenthesis.)

`Unexpected item in document control information` (fatal error)
The item shown in any occurrence data doesn't make sense where it is. (The most common cause of this error is a missing comma, which would have separated a keyword value from the next keyword; in this case, the unexpected item shown would be the next keyword.)

`Unrecognized boolean value`
The value displayed in any occurrence data is being interpreted as a boolean value and is not valid (see the list of legal boolean values in section 3.3, page 20).

Translation Subsystem Messages (7.5)

The translation subsystem (the IMAGEN virtual machine, or IVM) can produce various messages, most of which are fatal errors. These messages are listed below, in alphabetic order.

`AL result overflow` and `Invalid operand (AL)`
Interpretation of the current document has encountered an arithmetic overflow in the IVM world. The usual cause of this problem is positioning so far off the page that the coordinates overflow the range $[-2^{14},+2^{14})$.

`Can't find language emulator` (fatal error)
The emulator for the language specified in the document control information wasn't found in the IMPRINT-10's file system. A likely cause for this error is a misspelling of the language document control keyword, or the use of an IMAGEN language which isn't present in your IMPRINT-10.

`Flushed leftover document bytes`
This message's detail tells how many document bytes (8-bit data) were leftover from document processing; this is often innocuous, if, for example, the language is Impress and the Impress EOF command occurs some number of bytes before the actual end of document. The number of flushed bytes is often a good clue to localizing the problem area when document processing was abandoned for some reason. Note that this count is the number of document bytes as seen by the translation subsystem, which can differ from the number of bytes sent by the host, if communicating the document involves data expansion or compression.

`Glyph table overflow`
There is no more room in the glyph table to store the glyph being defined, so the definition is ignored. This will often be followed by other errors if the glyph is later referenced.

`Internal state`
This message, which appears after an IVM fatal error, details the IVM internal state for the sake of IMAGEN Customer Support. It is otherwise uninteresting.

`Invalid glyph dimensions`
The height or width dimension (or both) in a glyph definition is negative or zero.

`No document language specified in control information` (fatal error)
Either the document has no `language` specifier, or because of a large amount of previous document control information, there is no room to store it. In either case, the document can not be processed, as the IMPRINT-10 has no clue as how to interpret its contents.

`Not enough memory for page; won't be printed`
The page image identified by any occurrence data is too large for the IMPRINT-10's internal page memory, and it can't be printed. Otherwise, document processing proceeds apace.

`Object area exhausted` (fatal error)
The IVM internal object area is full, and the current document operation needs more space.

Trap (fatal error)
Interpretation of the current document has resulted in an unhandled exception, named by the occurrence data. This is a rather drastic error, and indicates a language emulator problem, as exceptions are supposed to be handled in a more graceful manner.

Undefined document code (fatal error)
The document byte code (given in decimal in the occurrence data) is meaningless in the current document language.

Unexpected end of document (fatal error)
The document ends abruptly in the "middle" of something, such as a glyph definition.

VS oflow (fatal error)
The emulator for the current language has exceeded an internal limit (viz., for the value stack).

XS oflow (fatal error)
The emulator for the current language has exceeded an internal limit (viz., for the execution stack).

The following errors indicate something rather drastic is wrong with some component of the system, and should be reported to IMAGEN Customer Support immediately. They will not be documented further.

```
DeleteFamily not implemented
Frame Get/Put out of bounds
Get/Put out of bounds
Glyph decoding confused
Impossible situation
Invalid Macro file
Invalid message
Invalid operator
Invalid subspace
Macro file not found
Macro file too long
Macro file too short
VS uflow
XS uflow
```

Paper Handling Subsystem Messages (7.6)

(See Chapter 5, page 37, for more information about paper handling.)

Document's paper type doesn't match loaded paper cassette type
The paper type specified or defaulted by the document does not match the LBP-10 paper cassette just loaded (or present at the start of document processing). The occurrence data names the problematic paper cassette. Note that this is only a warning; the system will go ahead and use the cassette. Note that some cassette sizes correspond to multiple paper types, so this is not guaranteed to catch all paper type/cassette mismatches.

LBP-10 problems: re-printing
The LBP-10 has had some problems which could lead to paper loss, and the IMPRINT-10 is reprinting the page and copy named in any occurrence data.

Page queue overflow (fatal error)
The internal page image queue is full and a new page image has been produced by the trans-

lation subsystem (currently, the limit is 250 pages in the queue). This can only happen if the job specifies special paper handling, such as page reversal or multiple copies with collation.

`Printing all copies`
The IMPRINT-10 is out of memory for its page image queue, and must print all copies of pages, up through the page mentioned in any occurrence data.

`Unrecognized paper type (ignored)`
The paper type given as the value of the `paper` document control item, and shown in any occurrence data is not one of the values known to the IMPRINT-10. (See section 3.3.3.1, page 21, for more information about paper types.)

`Unrecognized papermargin value`
The value of the `papermargin` document control item, shown in any occurrence data, does not make any sense. (See section 3.3.3.1, page 21, for more information about paper margin values.)

Rasterization Subsystem Messages (7.7)

The rasterization subsystem is responsible for putting the final bits of the page images on paper, and can produce the following messages.

`Compiler error`
The translation subsystem has made a mistake in producing a page image; this is an internal error, and is rather serious. Please notify IMAGEN Customer Support if it occurs.

`Missed paper top or bottom`
Some problem occurred with paper handling in the LBP-10, and the IMPRINT-10 didn't see the paper top or bottom signals. This will usually be accompanied by other messages about LBP-10 problems.

`Missed scan lines`
The page was complex enough that the IMPRINT-10 missed the number of scan lines given in any occurrence data.

Job Subsystem Messages (7.8)

The job management subsystem can produce the following messages about operator intervention in document processing. (See Chapter 4, page 25, for more information about the console commands mentioned here.)

`Operator ended job`
The operator artificially ended document input from the console with the `End (current job)` command.

`Operator aborted job`
The operator aborted document processing from the console with the `Abort (current job)` command.

Message Subsystem Messages (7.9)

The IMPRINT-10 subsystem handling job messages can, itself, produce several messages:

`Job error limit exceeded` (fatal error)
The number of job errors has passed the `maxerrors` document control limit.

`jobMemory must be in range 1 to 10; using default`
The `jobmemory` value shown in any occurrence data is not in the specified range.

`Number of job messages:` *# messages/,* `omitted:` *# omitted/*
(described earlier, in section 7.2.2)

Messages from Language Emulators (7.10)

The IMPRINT-10 applications language emulators such as Impress, Printer, etc., can also produce error messages; these messages will appear in the format given in section 7.2.2. They are always page-specific, but never copy-specific, as they are only apropos the composition of a given page image. See the relevant language Chapters, section "Messages," for a list of messages the emulator can produce.

CHAPTER 8
Configuration

Introduction (8.1)

The IMPRINT-10 is mostly factory-configured, based on the communications interface ordered. However, there are several configuration choices that you can make at installation or later, and this chapter describes them.

Configuration Hardware (8.2)

Configuration choices are made by installing or removing jumpers on the IP68 CPU card. The configuration jumper set is the 16 leftmost pin pairs (paired vertically) of the 25 pairs at the top right of the IP68 board; they are *numbered from the left, starting at 1* with the board's component side facing you, and with the bus edge connectors pointing down. Be very careful to avoid shorting the other jumpers to the right, as hardware damage can result.

Normally, all jumpers are "out" (i.e., not jumpered), and this is the default configuration; it is designed to be sufficient for most needs. However, should you need to install some jumpers given the configuration information below, you can use the jumpering connectors shipped with your IMPRINT-10. We use the symbol • to denote an installed jumper, and o to denote an uninstalled jumper.

N.B. Don't use this guide to configure systems older than Version 1.7; instead, call IMAGEN Customer Support for help.

Serial Line Configuration (8.3)

The two RS423 serial lines standardly available on each IMPRINT-10 can be configured in several ways. (See section 4.7, page 35, for information about configuration reporting at system start-up.) Note: these RS432 lines may not work properly with some picky RS232C terminals and host serial interfaces.

The most fundamental jumper is jumper 4 (numbered from 1, starting at the left), which chooses the logical assignment of host and console ports to physical port. The top connector on the back of the IMPRINT-10 enclosure, called the "B" connector, is a normally the host serial port (if you are using a serial communications option) and is a DTE, i.e., it looks like a terminal to any host. The bottom connector on the back of the IMPRINT-10, called the "A" connector, is normally the console port, and is a DCE, i.e., it looks like a modem (which is correct for most terminals). Only the "A" connector has modem control signals, with DTR and CTS normally high; if you want to use the CTS-toggling flow control, for example, you will have to use this connector for the host port.

- o connector "A" is the console port, and connector "B" is the host port
- • connector "B" is the console port, and connector "A" is the host port

Jumpers 1 and 2 (again, numbered from 1, starting at the left) together describe the desired speed for the *logical* host port (connector "A" or "B", depending on Jumper 4).

- o o 9.6 kilobaud
- o • 4.8 kilobaud
- • o 19.2 kilobaud
- • • 1.2 kilobaud

Jumpers 6 and 7 together describe the desired speed for the (logical) console port (connector "A" or "B", depending on Jumper 4):

- o o 9.6 kilobaud
- o • 1.2 kilobaud
- • o 4.8 kilobaud
- • • .3 kilobaud

Console Terminal Configuration (8.4)

The IMPRINT-10 can use either a CRT or a hardcopy terminal as its console. On a CRT, the IMPRINT-10 assumes it can erase characters from the screen by outputting a backspace-space-backspace sequence, and it uses this to good effect. By default, the console is assumed to be a CRT, but if that is not the case, you can use Jumper 5 to inform the IMPRINT-10.

- o the console is a CRT (with the erasure property described above)
- • the console is not a CRT

Communications Configuration (8.5)

Jumpers 8, 9, 10 and 11 are communications configuration jumpers. Their use is dependent on the communications interface installed at the factory. (See Chapter 6 for more details.)

Byte Stream Protocol (8.5.1)

The byte stream protocols use the communications configuration jumpers as follows.

Jumper 8:

- o 7-bit data: the high order bit of each byte is taken as zero
- • 8-bit data: the 8-bit data byte is taken as given

Jumper 9:

- o unprintable data bytes (less than 32 or equal to 127) are ignored; the end of document character is > (62, octal 076), and the quote character is \ (92, octal 0134)
- • unprintable data bytes are taken as given; the end of document character is control-D (04), and the quote character is control-B (02)

Jumper 10 (serial only):

- XON/XOFF flow control
- CTS-toggle flow control

Jumper 11: (currently unused)

Packet Protocol (8.5.2)

The serial packet protocol interface uses no configuration jumpers.

Default Document Language Configuration (8.6)

[Not implemented yet.]

CHAPTER 9
Impress Language

Introduction (9.1)

This manual describes the IMPRINT-10 Impress language, a page layout language. The Impress language features:

- full orientation flexibility: Impress can orient the printed page in all four directions.
- full character placement freedom: A character can be placed anywhere on the page, in any orientation, with no restrictions
- compact representation: Placing adjacent characters on the page in the same font only requires one byte per character
- typesetting application support: Character, word and line advance are controllable in all four orientations, and set to the natural direction of text
- font flexibility: There is no pre-defined limit on the size or number of characters defined or used in a document
- resident fonts: Impress supports many resident fonts, each of which is useable in all four orientations
- "down-loaded" character bit images: In addition to characters in the resident fonts, documents can define and use their own characters.
- high-level graphics commands: Impress can draw polygon outlines with a textured pen and fill polygons with texture.
- forms layout support: Impress provides the ability to draw textured rectangles simply.
- scaleable graphics plane: The page can be magnified by a factor of two or four.
- low-level bit-raster graphics: Impress can print images directly from a bit-raster
- macro facility: Users can define up to 256 substitution macros

Be aware, as you read this chapter, that Impress contains some very powerful features for advanced applications, and thus it may seem overly complex at first glance. It *is* possible to use Impress in a very simple fashion, and section 9.14, page 91, provides some helpful suggestions.

Motivation (9.2)

Until recently the major difference between hand-writing and other forms of printing lay in the distinction between character formation and the creation of the final output. Written characters are created each time they are drawn, allowing an arbitrary selection of shapes and removing any mechanical distinction between text and graphics. In contrast, all traditional printing devices separated the task of creating a shape from the physical printing. A given piece of equipment would have only a limited selection of character shapes, and creation of a document required only that the desired shapes be identified sequentially. Because graphics could not be handled in this way, special equipment (chart recorders, x-y plotters) had to be developed specifically for that purpose.

Raster output devices, which divide the page into many dots and print by darkening these dots, make printing much more flexible, much more like writing. Each time a character is drawn, it is created anew and can have a different appearance. Furthermore, the need to distinguish between graphics and text disappears. Devices of this sort include dot matrix printers, laser printers, and some CRT terminals. But this flexibility comes at a price: it takes much more information to describe every dot on the page than to list the characters that appear on it. For example, a typical page of 500 words contains about 3000 characters, and could thus be described in the same number of bytes to a character device such as a line-printer. Even at the fairly low resolution of 100 dots, or pixels, per inch, the pixel image of the same page might contain 1,000,000 pixels, requiring about 125,000 bytes to describe.

A related problem encountered with electrographic raster printers is that they print an entire page at a time and therefore require that data be sent to them at a constant speed until the page is finished. This requires that the host dedicate itself to printing once it has begun, which is usually impractical and always inefficient. Alternatively, the printer could be built to buffer an entire page before printing; but until recently, it was hardly practical to build a printer with the megabyte or so of memory needed for quality printing.

To achieve some of the benefits of raster print mechanisms without bringing the world's computers to their knees, designers began building some intelligence into the printing unit. At first, printers were built which compromised their versatility for the sake of convenience. Most raster printers look like a traditional printer to their host; they have a fixed repertoire of characters, which can only be changed by some external means (by loading new floppy disks or changing EPROMS). Although it improves efficiency tremendously, this effectively returns printing to the age of the line printer. The results look better; but only a few different types of characters can be used at a time, and graphics and texts are once again hopelessly separate.

Impress was designed to compromise efficiency and flexibility as little as possible. It allows the host to describe the raster image of each page fully, but without wasting a lot of time or storage -- freeing your computer to do what you bought it for. Characters of any size and orientation can be loaded once and referenced thereafter with a single byte; and Impress's resident fonts go even further, eliminating the need to load all but a few special characters with each document. Furthermore, Impress allows you to describe graphic images quickly and compactly; and it allows you to mix graphics and text on the same page. Finally, Impress is simple; unlike some language for graphics or typesetting, Impress only has three dozen commands.

Units of Measure (9.3)

Currently, Impress uses units of measure which are equal to $1/240^{th}$ of an inch, and all references to distances are in these units. In the future, other units of measure will be supported by providing a command to specify the unit measure in absolute distances, allowing Impress documents to be generated in a printer-independent fashion.

Note that some features of Impress will always be printer-resolution dependent, such as downloaded character bit forms and bit-raster images.

The Forms of Impress (9.4)

To the Imprint 10, Impress commands are strings of bytes which the machine can read directly. This we call the "Encoded form" of Impress. Since it would be difficult for a user to read commands in this form, we have developed another form of Impress, called the "Publication form," in which easily readable mnemonics stand for the actual commands and their arguments. For example @SET-FAMILY 2@ represents the bytes 207 002 (this command is explained in section 9.8.8). We will always use the publication form to give examples, leaving the encoded form for the machine itself.

Publication Form (9.4.1)

The syntax of the publication form is simple. The "@" sign begins and ends every command in publication form, as it does in the example above. Operands to commands are given in decimal form. In addition, a space separates the command from its operands and, for commands with multiple operands, the operands from each other. To represent any printed characters, the publication form uses the characters themselves if their ASCII value falls between 33 and 126; e.g.

```
@SP@THIS @SP@ IS @SP@ A @SP@ TEXT
```

commands the system to print "THIS IS A TEXT" Since characters outside of this range have no ASCII equivalent, we will represent them by @LIT x@, where "LIT" means "Literal" and *x* is the character's value. Note that in publication form, as well as in encoded form, spaces appear as separate commands (@SP@).

The IMPRINT-10 does *not* accept documents written in publication form; this is only used for convenience in our descriptions. If, for some reason, you want to write documents directly in Impress, you may want to write a compiler that will translate publication form into encoded form, described below. However, such a compiler would be less useful than you might think at first; Impress was not designed as a primary language for document description, but as a mediator between the IMPRINT-10 and high level typesetting languages (e.g. *Troff* and TeX).

Encoded Form (9.4.2)

The Impress publication syntax used in this Chapter is completely independent of any particular encoded representation. Impress does have a standard encoded form, and it is this form that the IMPRINT-10 Impress language expects. This encoding may be described in a "top-down" fashion as:

- an Impress document is a sequence of zero or more Impress commands;
- an Impress command, in turn, is a sequence of one or more (8-bit) bytes, the command code

itself appearing as the first byte, followed by zero or more bytes of operands; the number of operands is either fixed or variable, depending on the command;

- an operand is a sequence of one or more bits, of either fixed or variable length, depending on its type, chosen from what we will call *operand types*:

bit	the operand is encoded as one bit; value range [0, 1];
bits(i)	the operand is encoded as an signed quantity in i bits, high order bit is sign bit; value range $[-2^{(i-1)}, 2^{(i-1)}-1]$;
ubits(i)	the operand is encoded as a unsigned quantity in i bits, high order bit first; value range $[0, 2^i-1]$;
byte	the operand is encoded as a signed, 8-bit byte quantity (the high order bit is the sign bit); value range [-128, 127];
ubyte	the operand is encoded as an unsigned, 8-bit byte quantity; value range [0, 255];
word	the operand is encoded as a signed, 16-bit quantity, restricted to the value range $[-2^{14}, 2^{14}-1]$;
string	the operand is encoded as a null-terminated sequence of bytes.

Note: the Impress encoding is "big-endian," that is, a high-order byte preceeds lower-order bytes in the case of multiple-byte values. And, the order of bits within a byte is given in a big-endian fashion, that is, most significant or high-order before less significant or low-order.

Description Form (9.4.3)

We use the following format to describe each command:

command: byte **NAME**
an-operand: type-1
another-operand: type-2
...

where NAME is the Impress command name mnemonic, *command* is the command's decimal value (a byte), *an-operand* is the first operand's name, of operand type type-1 (see previous section), *another-operand* is the second operand, of operand type type-2, etc. The type of each operand shows implicitly how it must be encoded. Bit or multi-bit operands, which occupy only part of a byte, are encoded from left-to-right (i.e. from the high-order bit to the low-order bit) within a byte. The first operand may have the form "0: bits(n)," which means that n 0 bits must follow the command byte before the operands are given. These zeros "fill out" the operands, so that they occupy an integral number of bytes. If a group of operands is bracketed, in the form

number: ubyte
/ *operand-x:* type-x *operand-y:* type-y /$_{number}$

these operands must occur *number* times in the order given; if, for example, the operand *number* has the value 5, five ordered pairs (operand x, operand y) must be given. In these cases, the repetition factor *number* will be an operand itself, as it is above. In some cases, the name of the command itself may be bracketed. This indicates that the command has been included for compatibility with Impress version 0 and ought to be avoided, since future versions may no longer support it.

A description of the command's function will follow the description of its syntax. Unless otherwise distinguished, all numbers in these descriptions will be given in decimal.

Document Structure Commands (9.5)

219: byte ENDPAGE

The ENDPAGE command declares the current page image complete, queues it for printing according to the document's requested paper HANDLING (see Chapter 5), and starts page layout on a fresh page. When you start a new page, no state variables (variables like pen size, current family, which are set once and remain the same until changed) will be changed. This includes the current *h, v* position. Therefore, after every page you will need to insert some command to bring the *h, v* position to its proper place on the new page, be that the upper-left-hand corner or somewhere else.

255: byte EOF

The EOF command marks the end of the Impress document. It is entirely optional: the physical end of document is sufficient. Note that any Impress commands between the last ENDPAGE command and the EOF command (or physical end of document) will be processed but not printed, and that any data after the EOF command but before the physical end of document will be completely ignored (with a warning message; see section 7.5, page 58).

(Also see section 9.6, page 69, and section 9.15, page 92, for information about the PAGE command. The PAGE command was logically a document structure command in Impress version 0.)

Coordinate System Setting and Positioning Commands (9.6)

Impress uses two coordinate systems: a "physical" coordinate system which keeps track of absolute movements on the physical page which will be printed, and a "logical" coordinate system which the user can manipulate to suit his needs. We label the axes of the physical coordinate system *x* and *y*. The origin of these axes is in the upper left-hand corner of the page; *x* increases to the right, and *y* increases from top to bottom. The maximum values of *x* and *y* depend on the size of paper being used, but within this limitation, the system is independent of the actual paper size.[7]

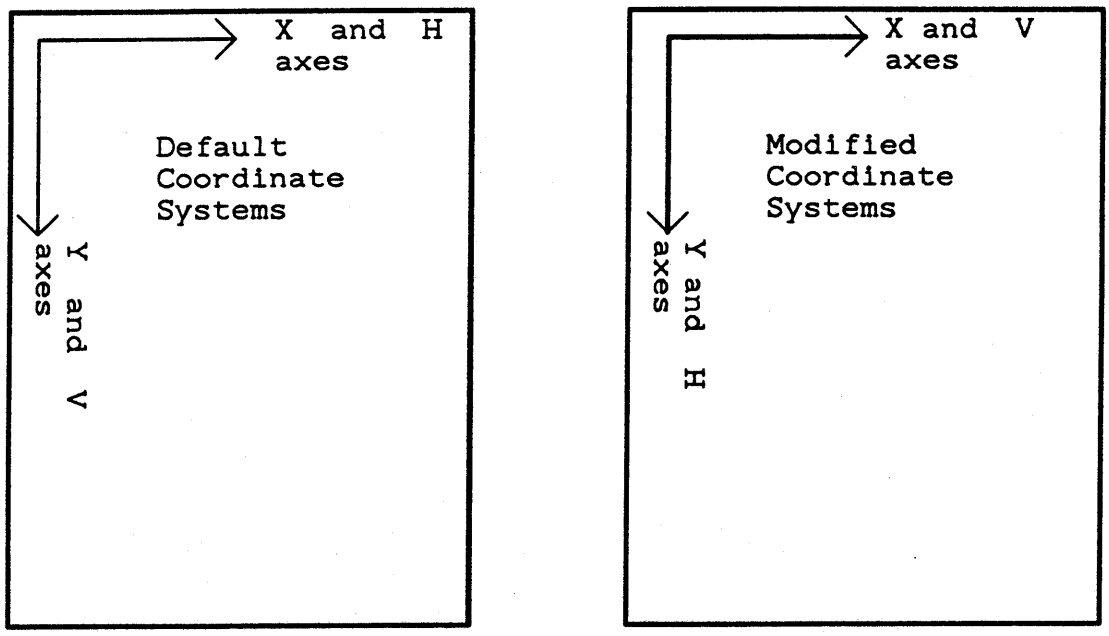
The *logical* coordinate system is another pair of axes, called *h* and *v*, which the user can manipulate for any particular application. *h* and *v* stand for "horizontal" and "vertical," and initially agree with the *x* and *y* axes of the physical system. When changed, they will no longer be "horizontal" or "vertical" in any absolute sense, but will reflect your definitions of horizontal and vertical for any particular application.

A few simple rules govern the orientation of these axes. They must be 90 degrees apart (orthogonal), they must be parallel to the edges of the page, and they are always "connected"-- that is, the coordinates of adjacent points differ by one. Within these limitations,

[7] See section 5.3, page 38, for more information about changing page image dimensions.

the user can change the orientation of these axes in two ways: by rotating the logical coordinate system and by changing the angle between the *h* and *v* axes. When rotating the logical coordinate system, the amount of rotation is measured clockwise from the physical *x* axis to the logical *h* axis. Since the axes must be parallel to the page edges, there are only four possible rotations: 0, 90, 180, and 270 degrees. Likewise, the angle between the *h* and *v* axes (measured clockwise from *h* to *v*) can only have two values: positive or negative 90 degrees. Changing this parameter "flips" the *v* axis around the *h* axis.

- Example: To draw a graph which stretches across the page the "long" way ("landscape" orientation), you would want to rotate the *h* axis 90 degrees, making it coincide with the physical *y* axis. Then you would "flip" the *v* axis by setting the angle between *h* and *v* to -90 degrees. This will make it coincide with the physical *x* axis. This transform would simplify the work needed to translate your data points to locations on the physical page. These operations give the result below:



Besides changing the orientation of the logical coordinate system, we can also move its origin. This can be done in two ways. First: the origin can be placed at the "upper-left-hand corner" of the current *logical* page. If you have not changed the relationship between the *h* and *v* axes (i.e. it is still positive 90), this will be the corner in which all *h* and *v* coordinates on the page are positive. If you *have* changed this relationship, the top left corner will be the corner in which all *h* coordinates are positive and all *v* coordinates are negative. Note that this does *not* correspond to the upper-left-hand corner of the physical page, but reflects your definition of what "upper-left-hand" means. Second: the origin can be placed at the current physical page location. This

allows the origin to move to the interior of the page. This is particularly useful if you want to insert figures into a text; you can make the figure in the most convenient way, prefacing it with a command to set the origin to the current physical page location. This will let a printing program place the figure wherever it needs to be inserted.

The following commands control the logical coordinate system:

SET_HV_SYSTEM

205: byte
0: bit *origin:* ubits(2) *axes:* ubits(2) *orientation:* ubits(3)

The SET_HV_SYSTEM command selects the logical coordinate system used to lay out your pages. The operand *axes* sets the angle between *h* and *v*, and the operand *relation* governs the rotation of the logical coordinate system, and *origin* controls the position of the new system's origin. These operands set state variables, which remain in effect until changed explicitly. Since the default values for *axes*, *orientation* and *origin* set the *h* and *v* axes equivalent to *x* and *y*, documents need not contain an explicit SET_HV_SYSTEM command.

The document can define the new logical coordinate system either with respect to the physical (*x*-*y*) system or with respect to the current logical system. This allows you to rotate an entire page using many different coordinate systems with a single command at the beginning, provided that all the previous coordinate systems were defined relatively -- that is, with respect to the previous logical system. Of course, any SET_HV_SYSTEM commands that define new coordinates with respect to the physical coordinate system would have to be changed explicitly.

When this command is used to change coordinate systems, the new origin is set based on the value of *origin*:

0, 1 no change (leave the origin where it is);

2 set the origin to the "top left-hand corner" of the page in the new *logical* orientation;

3 set the origin to the current physical page position.

The new orientation's logical coordinate system axes relation is set according to the value of *axes*:

0 no change;

1 invert the relationship between the *h* and *v* axes (if it was positive, make it negative, and vice-versa);

2 set the relation between *h* and *v* to positive 90 degrees (clockwise);

3 set the relation between *h* and *v* to negative 90 degrees (counter-clockwise).

The new orientation (the positive angle between the new logical *h* axis and the physical *x* axis) is set according to the value of *orientation*:

0 0 degrees from the current *h* axis (i.e., no change)

1 90 degrees from the current *h* axis

2 180 degrees from the current *h* axis

3 270 degrees from the current *h* axis

4 0 degrees from the physical *x* axis

5 90 degrees from the physical *x* axis

6 180 degrees from the physical *x* axis

7 270 degrees from the physical *x* axis

This command does not change the physical position of the printer's "pen"; it only changes the coordinate system in effect. If you want to move to the new origin after the SET_HV_SYSTEM, you'll have to move explicity, e.g., with SET_ABS_H and SET_ABS_V commands (see below).

135: byte **SET_ABS_H**
new-h: word

The SET_ABS_H command sets the *h* position to *new-h* (an absolute move).

137: byte **SET_ABS_V**
new-v: word

The SET_ABS_V command sets the *v* position to *new-v* (an absolute move).

136: byte **SET_REL_H**
delta-h: word

The SET_REL_H command adds *delta-h* to *h* (a relative move).

138: byte **SET_REL_V**
delta-v: word

The SET_REL_V command adds *delta-v* to *v* (a relative move).

213: byte **[PAGE]**

The PAGE command sets both *h* and *v* to zero (i.e., positions to the current coordinate system origin; see section 9.15, page 92).

195: byte **[SET_HPOS]**
h: bits(15) *abs-or-rel:* bit

The SET_HPOS command performs either a

```
@SET_ABS_H h@
```

or a

```
@SET_REL_H h@
```

as *abs-or-rel* is 0 or 1, respectively. This command emulates the Impress version 0 SET-XPOS command in the default logical coordinate system.

196: byte [SET_VPOS]
v: bits(15) *abs-or-rel:* bit

The SET_VPOS command performs either a

```
@SET_ABS_V v@
```

or a

```
@SET_REL_V v@
```

as *abs-or-rel* is 0 or 1, respectively. This command emulates the Impress version 0 SET-YPOS command in the default logical coordinate system.

Text Positioning Commands (9.7)

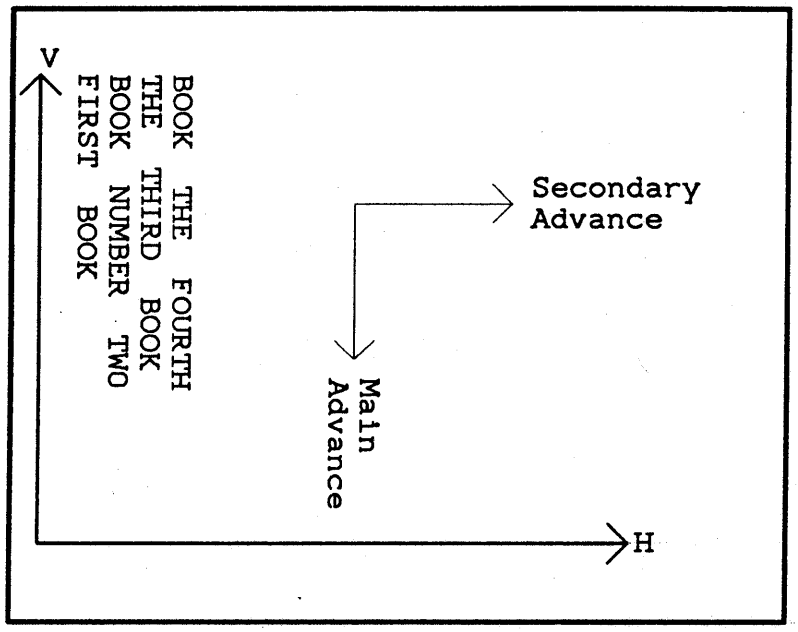
Any text has a built-in order. For example, we read English from left to right, from the top of the page to the bottom, and from the front of the book to the back. Other languages are written differently; Chinese is written from top to bottom in columns which are ordered from right to left. Everyone's life will be easier if text can be handled in the order in which it is written; imagine the difficulty of trying to type Chinese on a typewriter with an English mechanism. It is simplest and most efficient to store and print texts in their own natural order. Therefore, Impress lets you alter the directions in which texts are placed on the page. In addition to its obvious uses for foreign language text, this facility makes it much easier to draw graphs and charts, since labels can be placed in any direction.

We need two concepts to describe the order in which texts are presented. The *main advance direction* of a text is the direction of the lines on which it is written; it is the direction your eye follows in reading a single line. In English, the main advance is from left to right; in Chinese, it is from top to bottom. The *secondary advance direction* gives the direction in which lines advance. English lines move down the page, so the *secondary advance direction* lies along the physical y axis. For Chinese, the secondary advance would be from right to left, or along the x axis in a negative direction. Impress requires these directions to be perpendicular to each other and to parallel the horizontal and vertical axes. Within these restrictions, you can manipulate the advance directions much as you can manipulate the logical axes.

While it is easy to confuse the logical coordinate system and the advance directions, the two concepts differ significantly. The logical coordinate system determines the layout of the page, while the advance directions determine how the characters are placed within that layout. Consequently, while the advance directions determine where to place the next character on the page, they do not effect the meaning of any pair of coordinates. Any pair of coordinates will always be interpreted in terms of the current h and v axes, no matter what advance directions you have chosen. Therefore, when characters are not being printed (as in graphics), the "advance directions" have no effect. Both together control the orientation in which Impress will print a character. Characters will appear so they can be read normally (i.e. "rightside up") along the main advance direction; but this direction is defined in terms of the logical coordinate system. Therefore, the angle at which any character will be placed on the physical page is the sum of the logical coordinate system's rotation and the additional rotation, relative to the logical coordinates, given to the main advance direction. When used together, the coordinate setting and text positioning commands allow many different types of pages to be printed with ease.

- Example– Printing a list of book titles as they would appear on a library shelf: First we want to shift the coordinate system to give a "landscape" orientation, that is, to turn the

page so that is short and wide. To do this, see the previous example. Now, thinking about a library shelf, we realize that titles are written from top to bottom; this is the *main advance direction*. The individual titles proceed according to call-number order, from left to right, giving us the *secondary advance direction*. Therefore, we set the *main advance* direction to be 90 degrees from the *h* axis and the *secondary advance direction* to be -90 degrees from the primary. After making these definitions, we can lay the text out simply by listing the titles in the proper order, separated by the proper "end of line" commands (CRLF).



Library books

Finally, since the spaces between words remain relatively stable (word spacing varies from line to line, and line spacing varies for different fonts), Impress provides commands to set these spaces conveniently.

206: byte
0: ubits(5) *main:* ubits(2) *secondary:* bit

SET_ADV_DIRS

The SET_ADV_DIRS command sets the main advance direction according to *main*:

0 0 degrees from the current *h* axis

1 90 degrees from the current *h* axis

2 180 degrees from the current *h* axis

3 270 degrees from the current *h* axis

and sets the secondary advance direction, relative to the main advance direction (considered as a vector), according to *secondary*:

0 positive 90 degrees (clockwise) from the main advance direction

1 negative 90 degrees (counter-clockwise) from the main advance direction

By default, the main and secondary advance directions are those correct for the latin-derivative languages, namely, 0, so you needn't use this command for ordinary text.

133: byte MMOVE
delta-m: word

The MMOVE command displaces the pen's the current (*h*, *v*) position by *delta-m* in the main advance direction; if *delta-m* is negative, then the the current position is displaced in the direction opposite the main advance direction. By default, when no main advance direction has been chosen, the MMOVE command simply changes the current *h* position by adding in *delta-m*. Under these circumstances, this command is the same as SET_REL_H. It might help to think of the main advance direction as a unit vector which we multiply by *delta-m* and then add to the current position vector, (*h*,*v*).

134: byte SMOVE
delta-s: word

The SMOVE command displaces the current (*h*, *v*) position by *delta-s* in the secondary advance direction; if *delta-s* is negative, then the current position is displaced in the direction opposite the secondary advance direction. By default, when no secondary advance direction has been chosen, the SMOVE command simply changes the current *v* position by adding in *delta-m*. Under these circumstances, this command is the same as SET_REL_V. (As with MMOVE, it may be helpful to imagine such a move, in the general case, as adding the secondary advance unit vector, multiplied by *delta-s*, to the current position considered as a vector (*h*, *v*).)

210: byte SET_SP
space-size: word

Most document-production systems attempt to space words evenly on the line. It is usually not possible to do this and simultaneously keep a given line width; with a fixed-width space, the results will usually have the "ragged edge" we expect from a typewriter. However, if we use two space sizes differing by one unit, it will be possible to justify the right hand margin. Furthermore, the difference between the two space sizes will be unnoticeable. This command sets the current inter-word space size (a state variable) to *space-size*, permitting the next two single-byte commands to perform inter-word spacing by the current space size or by the current space size plus one.

128: byte SP

The SP command performs an inter-word space by moving in the main advance direction (the direction of text) by the current space size amount. By default, when no special advance directions are in force, this command is equivalent to adding the space size to *h*.

129: byte SP1

The SP1 command performs an alternate inter-word space, by moving in the main advance direction (the direction of text) by the current space size amount, plus one. By default, when no special advance directions are in force, this command is equivalent to adding the space size, plus one, to *h*.

131: byte MPLUS

The MPLUS command adjusts the current position by one in the main advance direction (the direction of text). By default, when no special advance directions are set, this command is equivalent to adding one to *h*. This command, along with the MMINUS command, below, can be useful when translating from a nearly-infinite-precision coordinate system (such as used internally by many document production systems) to the less resolute coordinates of Impress.

132: byte MMINUS

The MMINUS command adjusts the current position by one in the opposite of the main advance direction. By default, when no special advance directions are set, this command is equivalent to subtracting one from *h*.

197: byte CRLF

The CRLF command (reminiscent of "carriage-return line-feed") provides the "end of line" feature of Impress, along with the SET_BOL and SET_IL, below, which set the beginning-of-line margin and inter-line space amount, respectively. In the simplest case, when no special advance directions are in force, CRLF will simply set *h* to the beginning-of-line margin value, and add the inter-line space amount to *v*.

In more detail, the CRLF command changes the current position to the next line of text by moving to the beginning-of-line margin in the main advance direction, and by moving "down" by the inter-line space setting in the secondary advance direction. More exactly, if the main advance direction is parallel to the *h* axis, then CRLF will set *h* to the beginning-of-line margin value, and will add the inter-line space amount to *v*; if the main advance direction is parallel to the *v* axis, then CRLF will set *v* to the beginning-of-line margin value, and wil add the inter-line space amount to *h*.

209: byte **SET_BOL**
line-begin: word

The SET_BOL command sets the current beginning-of-line margin (a state variable) to the value *line-begin*. This value will be used no matter were the margin is; although the location of the margin will depend on the advance directions and may change in the course of a document, Impress will use the same margin width until you alter it with this command. The default value for *line-begin* is 0, so each document will probably need to set this variable explicitly. See the CRLF command description above for information on how this value is used.

208: byte **SET_IL**
inter-line: word

The SET_IL command sets the current inter-line space (a state variable) to the value *inter-line*. As with the *space-size* and *line-begin*, the default value for *inter-line* is zero. See the CRLF command description above for information on how this value is used.

130: byte **[M]**
delta-h: byte 130: byte

The M command adds *delta-h* to *h*, effectively moving a limited amount in the direction of text; this command has been superseded by the MMOVE command (see above).

Text-Printing Commands (9.8)

Glyphs (9.8.1)

A *glyph* is any mark (characters, shapes, etc.) that can be placed on the page repeatedly. This distinguishes a glyph from a shapes that can be printed but which are not stored in memory. Each glyph is printed on paper according to a rectangle of bits called *pixels* (for "picture elements"); if a bit in this rectangle, called the glyph's *pixel image*, is one, the corresponding rectangle on the physical page will be darkened whenever this glyph is printed. Therefore, we will call the pixels with value 1 "black" or "on," and those with value 0 "off" or white."

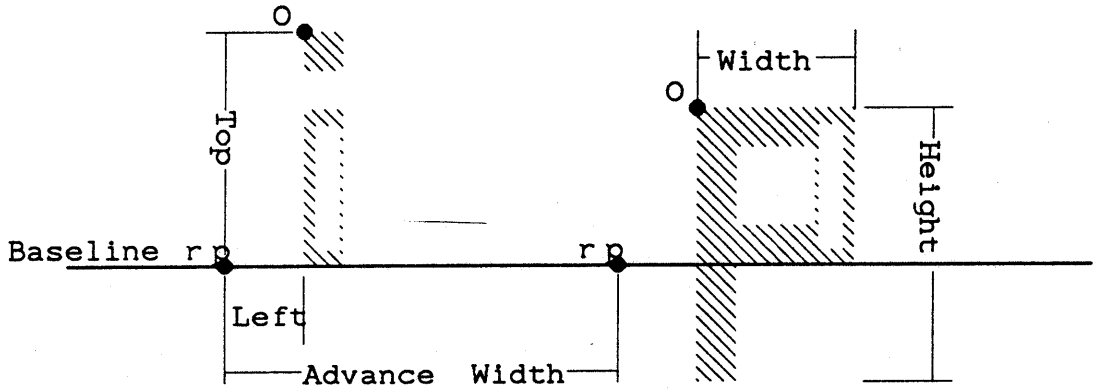
Now, any glyph, no matter how small, may occupy a lot of space on the page -- even in a 72-point font, a period will be a relatively small dot surrounded by a lot of white space. Obviously, saving a lot of white pixels wastes storage. To save storage, Impress defines each glyph with the smallest possible rectanglular pixel image containing all the glyph's black pixels. For example, the pixel image of a period only covers the small square containing the dot itself. As a consequence, all glyph definitions must include information about where to place the image; we will need to know where to put the dot so it will be a period, not some other kind of mark. Each glyph therefore has a *reference point*, which is the logical position on the page where the glyph "begins," and each definition specifies the properties *left* and *right offset* (explained below) which determine where to place the glyph's image relative to the reference point. Reference points always lie on the "baseline," or the line on which you are printing. In addition to the glyph's position, its definition must also tell Impress how far to advance before printing the next glyph.

Both the location of the reference point and the advance width are independent of the size of the glyph's pixel image.

From the above, an Impress glyph must have the following properties:

- *height*: the height in pixels of the glyph's pixel image;
- *width*: the width in pixels of the glyph's pixel image;
- *top offset*: considering the pixel image as a possessing an x, y coordinate system with origin at the upper-left-hand corner of the image (y increasing downwards), the top offset gives the y coordinate of the reference point (may be negative or positive);
- *left offset*: similarly, the left offset gives the x coordinate of the reference point (may be negative or positive);
- *advance width*: the distance to advance the current position after printing the glyph (unsigned, in the main advance direction; see section 9.7).

In this digram showing the properties of two glyphs, the *reference point* is labelled `rp`, the *top offset* is labelled `top`, the *left offset* is labelled `left`, and the origin of the glyph's pixel image is labelled `o`. The shaded area shows the size of the pixel image. Note how much smaller this is than the area the glyph effectively covers: the entire space between its reference point and the next.



Impress allows both host-resident and printer-resident glyphs. The former are defined by the user and must be "down-loaded" with each job; the latter are a permanent part of the Imprint-10 system.

Fonts (9.8.2)

For our purposes, a *symbol* is simply a number between 0 and 65735 which stands for an abstract character like "lower-case 'a'." By itself, a symbol merely names a character abstractly, saying nothing about how it will look on paper. An IMAGEN *font* is a collection of glyphs with a consistent size and style used to print symbols on paper; it defines one possible way to print a set of symbols. Do not confuse the concepts of *glyph* and *symbol*; a particular glyph always looks the same, but the appearance of a symbol will differ tremendously from one font to the next. Just as the letter "a" remains the same regardless of your handwriting, symbol 97 (lower-case "a" in

ASCII) remains the same no matter what font you use to print it. Impress uses a very general definition of a font which allows each font to have many (65736) symbols.

Rotation (9.8.3)

Earlier (see section 9.7, page 75) we introduced the possibility of printing text in different orientation on the page. However, each glyph can only appear in one absolute orientation. This requires that each font provide four glyphs for each symbol: one for each orientation it can have on the page. All four glyphs will look similar, except that they will be "turned" in different directions. *Rotation* determines which of these glyphs should be used in any situation. Glyphs with rotations of 0 will be used to print normally oriented text; glyphs with rotations of 90 will be used to print text moving down the page, and so on.

Whenever any symbol is used, Impress computes the proper value of *rotation* from the orientation of the logical coordinate system and the main advance direction. Therefore, while every glyph definition must include two bits to describe the glyph's rotation, you never need to specify rotation when printing.

Family and Member (9.8.4)

Since fonts are large collections of symbols, and since Impress supports many fonts, selecting a particular glyph for printing directly would require many bytes: two bits for the rotation, seven for the symbol number, and as many bytes as necessary to name the font. However, we can reduce the number of bits needed considerably by noticing a few things. First, printed glyphs usually occur from the same rotation; we would rarely want to change the rotation more than a few times on a page, and often won't need to change it at all. Therefore, Impress only needs to change the value of *rotation* when you shift the coordinate system or the main advance direction. Second, printed glyphs tend to come from the same font or, at most, from a few fonts. Therefore, since we can define the set of glyphs which will tend to occur together, we can reduce the average amount of information needed to print a glyph to just over one byte each.

The notions of *family* and *member* put these observations into practice. A *family* is a user-defined collection of characters, or "shapes," that occur together. By "character," we mean all possible rotations of a specific shape; thus, a family collects all the glyphs needed to print any shape we might want in any direction, providing that these glyphs have been defined (either by an explicit definition or a resident font). A family may be as simple as all of the characters in a particular font; but families can be much more complex, depending on your application. Families can have up to 128 characters, each of which we call a *member*. We refer to each character by its member-number, *m*.

Finally, then: To print text, an Impress document defines a set of families either with explicit glyph definitions (using the SGLY command) or by grouping glyphs already defined by the resident fonts (using the CREATE_FAMILY_TABLE and CREATE_MAP commands). The SET_ADV_DIRS and SET_HV_SYSTEM commands allow Impress to compute which rotation to use; and MEMBER commands (command bytes 0 to 127) select the proper glyph from the current family and rotation.

Glyph Identifier (9.8.5)

Any glyph definition must include some way of identifying the glyph so that it can be used later. Therefore, you must define each glyph with a unique *glyph identifier*, which is a sixteen bit quantity specifying the glyph's rotation (2 bits), family (7 bits) and member number (7 bits). Generally, you will only need to specify the entire glyph identifier when defining, or "down-loading" glyphs; when printing, Impress supplies the rotation and family from the context (see above). In most cases, Impress assumes that glyph identifiers remain the same throughout a document. However, they are completely under your control; glyphs can be deleted and their identifiers reassigned with the FORCE_GLY_DELETE command (see below).

Down-loaded Glyphs (9.8.6)

The following commands allow you to define glyphs explicitly within the body of a document. This is called "down-loading" glyphs. Glyphs defined this way *do not* belong to any IMAGEN font; they are identified only be their family and member numbers. The resolution possible for these glyphs depends on the resolution of the printer.

199: byte **BGLY**
rotation: ubits(2) *family:* ubits(7) *member:* ubits(7)
advance-width: word *width:* word *left-offset:* word
height: word *top-offset:* word
mask (see below)

The BGLY command defines the down-loaded glyph identified by

<*rotation, family, member*>.

Its advance width is given by *advance-width*, its reference point by *top-offset* and *left-offset* together, and its pixel mask size by *width* and *height*. *Mask* is a string of bytes that defines the glyph's pixel image. The bytes in *mask* are grouped in *height* rows; each row contains enough bytes to describe *width* pixels. Therefore, the number of bits in each row *width* rounded up to the nearest multiple of 8. For example, if *width* is 13, each row will have 16 bits or 2 bytes. The unused bits in each row are on the right, and will be ignored.

A row describes one row of pixels in the glyph's image from left to right horizontally (in the direction of the physical x axis). The rows are then stacked vertically from top to bottom (the physical y direction). Bits encoded one signify darkened, or black, pixels. Since a mask defines the glyph's image in terms of the physical coordinate system, each mask has an absolute rotation. Therefore, two identically-shaped glyphs from different rotations will have completely different masks and different values for *height*, *width*, *top-offset*, and *left-offset*.

198: byte **[SGLY]**
rotation: ubits(2) *family:* ubits(7) *member:* ubits(7)
advance: byte *width:* byte *left-offset:* byte
height: byte *top-offset:* byte
mask

The SGLY command is an obsolete version of the BGLY command.

Resident Glyphs (9.8.7)

In addition to glyphs defined by the particular document (down-loaded glyphs), Imprint 10 provides many resident glyphs permanently stored in a proprietary encoding. Each resident glyph is identified as a particular symbol in a particular font. Unlike down-loaded glyphs, they cannot be accessed directly, but must be referenced indirectly through *member maps* and *family tables*. These map the family and member names into font and symbol names. Since any resident font will provide its characters in all four rotations, and since Impress will keep track of the current rotation by itself, we can ignore the *rotation* operand in this discussion.

A member map takes a "member name" (a number between 0 and 127) and returns a "symbol name" (another number, possibly the same). To define this map, we use a set of triples

<starting member, starting symbol, count>.

Each triple linearly maps the members from "starting member" to "starting member +count -1" into the symbols from "starting symbol" to "starting symbol + count -1." For example, the triple <23,34,3> maps 23 into 34, 24 into 35, and 25 into 36. In other words, this map assigns symbol 34 to member 23; a reference to member 23 means "use symbol 34 if possible," and so on. The exact meaning of "symbol 34" will depend on what font is in use, and that symbol 34 may not even exist in a particular font. In addition, several members may be mapped into the same symbol and, surprisingly, any member may be given several symbol values. The simplest member map is the single triple <0,0,128>, which sets the symbol names equal to the member names. More complicated maps would have many triples.

You can define up to 127 different member maps, each of which has its own name, a number between 1 and 127. The map <0,0,128> is named "0" by the system and is always available for simple use.

A family table comprises a set of associations between member maps and fonts. To describe these associations, we use a set of pairs

<member map *m*, resident font *fn*>

This means "use member map *m* with font *fn* if possible." The simplest family table would have only one entry, <0, *fn*>; it would use font *fn* and the simple resident map 0 to print any member. Again, more complex family tables have many entries; and, within one table, several fonts may be associated with a particular member map. For example, <7, first font>, <7, second font> forms a perfectly legal table, although it associates two fonts with the same map. The significance of this will be explained later. Like the member maps, the family tables have names, ranging from 0 to 95.

So far, the member maps and family tables have been described abstractly, as pairs or triples of names. They function as follows: The user selects a particular family table. This is a "state variable"; this family table will remain in use until you select another. With this family table, Impress will interpret a given member *m*, representing some character, as follows:

First, Impress tries to find a down-loaded glyph described by the current member, family, and rotation; explicitly defined glyphs will be used wherever possible, taking precedence over those in the resident fonts. If there is none, it tries to find a glyph in the resident fonts through the family tables and member maps. To do so, Impress takes the first <map,font> pair from the family table and tries to find member *m* in this map. If this member does not appear in that map, it proceeds to the next pair. If it does appear, Impress takes the first symbol value for *m* in that map and looks that symbol up in the corresponding font, where it may or may not appear; not every symbol is defined in every font. If that symbol appears in the font, Impress prints it; if it

does *not* appear, Impress goes to the next value for *m* in the same map and tries again. Note that, with this procedure, the *order* of the triples in the member map is crucial; Impress tries different values for *symbol* in the order they're given. If impress does not find a character in *font* for any possible value of *symbol*, it will proceed to the next <map,font> pair in the family table. Now we can make sense of the duplicate assignment <7, first font >, <7, second font>; it means "if first font doesn't work with map 7, try second font." If Impress reaches the end of the family table without finding anything to print, it will print a special mark and give an error message (see section 9.16, page 92).

Given this procedure, it should be clear that the order of elements in both the family table and the member maps is extremely important. If you make complicated tables or maps, be sure the elements are ordered properly, listing symbols and fonts in the order in which you want them to be used.

Example:

current family table: <1,"roman">, <2,"gothic">
member map 1: <22,33,1> <27, 96,3>
member map 2: <20, 100, 30> <27, 999, 1>

Given member 27 to print:

Impress first tries map 1; it then finds symbol value 96 and looks up 96 in the "roman" font. If the symbol is there, it prints it and goes to the next member. If not, it goes to member map 2. The first symbol value for 27 in this map is 107, so Impress looks up 107 in the "gothic" font and prints it if it is there. If the "gothic" font has no character 107, Impress then proceeds to the second value for 27 in map 2, which is 999, and tries to print it with the same font. If there is still no character, Impress prints an `undefined glyph` error message, since there are no more maps to try.

The family tables allow a document to change between several different fonts easily. A typical document would probably want to use several different sizes of roman, boldface, and italic fonts in the same way. This can be done with one member map and several simple family tables associating this map to a particular font. Changing the family table would effectively switch fonts.

222: byte CREATE_MAP

map-name: ubyte *triples:* ubyte
[starting-member: byte *starting-symbol:* word *count:* ubyte *]$_{triples}$*

The CREATE_MAP command defines the member map named *map-name* (an ordinal in the range [1, 127]) to consist of the given *triples* triples, each constructed as

<starting-member, starting-symbol, count>.

(The map named 0 is reserved; see above.) Re-defining member maps can give unpredictable results.

221: byte **CREATE_FAMILY_TABLE**
family: ubyte *pairs:* ubyte
[*map-name:* ubyte *font-name:* string]$_{pairs}$

The CREATE_FAMILY_TABLE creates the family table for the family named by *family* (which must be in the range [0, 95]), with the given *pairs* pairs, each constructed as

<*map-name, font-name*>.

Size must be positive. Note that the *font-name* pair elements are strings, and are thus null-terminated. Re-defining family tables can give unpredictable results.

Printing Glyphs (9.8.8)

207: byte **SET_FAMILY**
family: ubyte

The SET_FAMILY command sets the current family to *family*, which must lie in the range [0, 95]. The current family setting is used by the MEMBER commands, below, and is a state variable (see section 9.12, page 89). The families named 96-127 are reserved by Impress.

0-127: byte **MEMBER**

Any Impress command code between 0 and 127 is a MEMBER command. Using the current family and rotation, these commands print the corresponding member at the current position, then advances the printer to the next position according to the main advance direction and the glyph's advance width. The process Impress uses to locate the proper glyph has been described above. Remember that down-loaded glyphs take precedence over resident glyphs, regardless of whatever family tables are in effect. (See section 9.7 for information about motion in the main advance direction.)

Cacheing Glyphs (9.8.9)

Since the IMPRINT-10 features dynamic memory allocation, it is rarely necessary to "cache" glyphs. The availability of resident fonts also makes it unlikely that you will need to manage memory directly; these glyphs are automatically generated from permanent storage and deleted as necessary. However, some applications using many different kinds of glyphs or particularly large glyphs (such as a font manual) may require you to delete glyphs in order to reclaim space in memory. The following commands allow you to cache glyphs:

200: byte **DELG**
rotation: ubits(2) *family:* ubits(7) *member:* ubits(7)

The DELG command deletes the glyph identified by the operand triple

<*rotation, family, member*>.

Glyphs deleted with DELG are only marked as disposable, and not actually deleted. This only makes a difference when, for example, a document deletes a glyph and later redefines it. Since Impress assumes that glyph identifiers are unique, any redefinition will be identical to the previous definition. If this uniqueness does not hold for some reason (e.g., a document wants to use more than 96 families at a time), then the following command can be useful.

240: byte **FORCE_GLY_DELETE**

The FORCE_GLY_DELETE causes all glyphs marked for deletion (excepting any glyphs used in the current page) to disappear forever, thus freeing their glyph identifiers for re-use. This command has a side-effect on any page images held for printing (e.g., in the presence of a collated multiple-copy document): they must be all printed at the time of the FORCE_GLY_DELETE, so use it with care. (See section 5.1, page 37, for more information about multiple copies, collation, and reversal.)

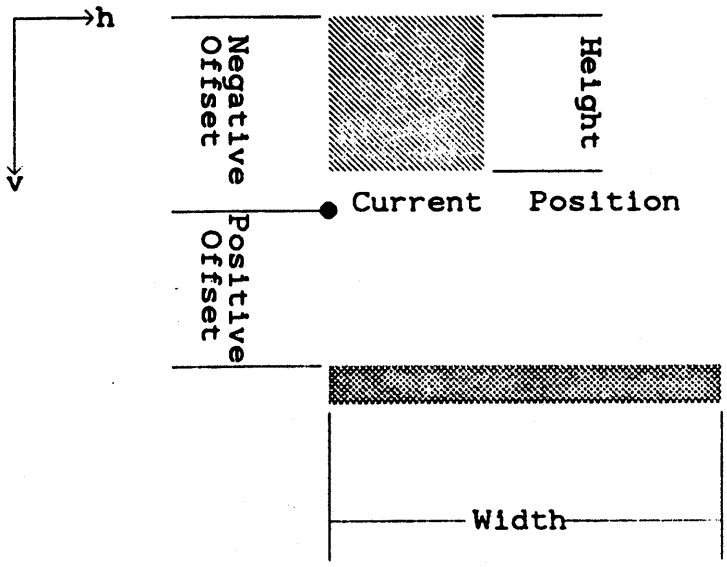
Text Rule Commands (9.9)

Impress has commands to print text "rules" (typesetting jargon for black lines or rectangles whose sides parallel the paper edges).

193: byte **BRULE**
width: word *height:* word *top-offset:* word

The BRULE command prints a rectangle with height *height* and width *width*. *Height* is measured along the *v* axis, *width* along the *h*. *Top-offset* controls the rectangle's position; it moves the rectangle from the current position in the *v* direction. If you have not changed the *h* and *v* axes, a positive *top-offset* will make the rule appear "below" the current position. Correspondingly, a negative *top-offset* will put the rule "above" the current position. In a sense, this is true in any coordinate system; just remember that "above" and "below" will be interpreted according to the current *h-v* system. In any coordinate system, *top-offset* specifies the distance between the current position and the "top" of the rule; remember that the "top" of the rule is the side with the smallest *v* coordinate, whatever set of axes you are using. All rules begin at the current *h* position and extend in the positive *h* direction. In an unmodified coordinate system, they will extend to the right.

If you are using a texture (see the SET_TEXTURE command, section 9.10, below), Impress will print the rectangle using this texture. If not, the rectangle will be printed in black. The following diagram shows two rules with different sizes, textures, and offsets:



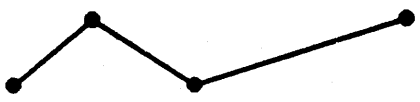
192: byte [SRULE]
width: byte *height:* byte *top-offset:* byte

The SRULE command, now obsolete, is the same as the BRULE commands, but its operands are capable of less range.

Graphics Commands (9.10)

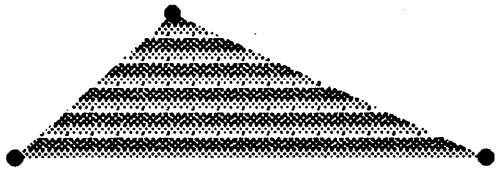
Impress can print graphic data in addition to texts. There are both "high level" and "low level" commands; the first describe and print polygons or their borders easily, while the second will produce any arbitrary shape from a bit map. The following definitions are central to the Impress graphics facility:

- the *page*: the Impress virtual page image being generated, considered as a rectangular array of pixels;
- the current *path*: a connected series of lines defined by their endpoints. Note that n line segments will have n+1 endpoints. Impress only allows one path to be active, or "current," at a time; once you have defined a path, you must finish working with it before defining a new one. Here is a 3 segment path:



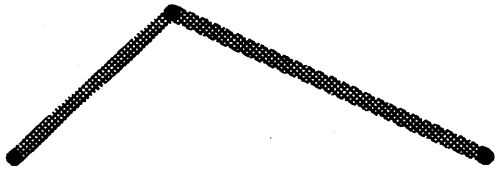
Example 1

- the current *texture*: a square glyph used to cover (tile) the texture page (a state variable);
- the *texture page*: an "imaginary" page tiled (covered) completely with a given "texture," (i.e. covered with a single character). The texture page matches the pixels of the actual page image one for one. It is used to "shade in" portions of your page with textures other than black.
- the *pen size*: the width of the line Impress draws when it is connecting points. The pen size can range from 0 to 20 units (its current size is a state variable);
- the *interior mask*: the set of pixels *inside* the polygon determined by the current path. The path need not be complete to define a polygon; if the path specified is not closed, Impress will connect its beginning and end. Example 2 shows the interior mask of a triangle.



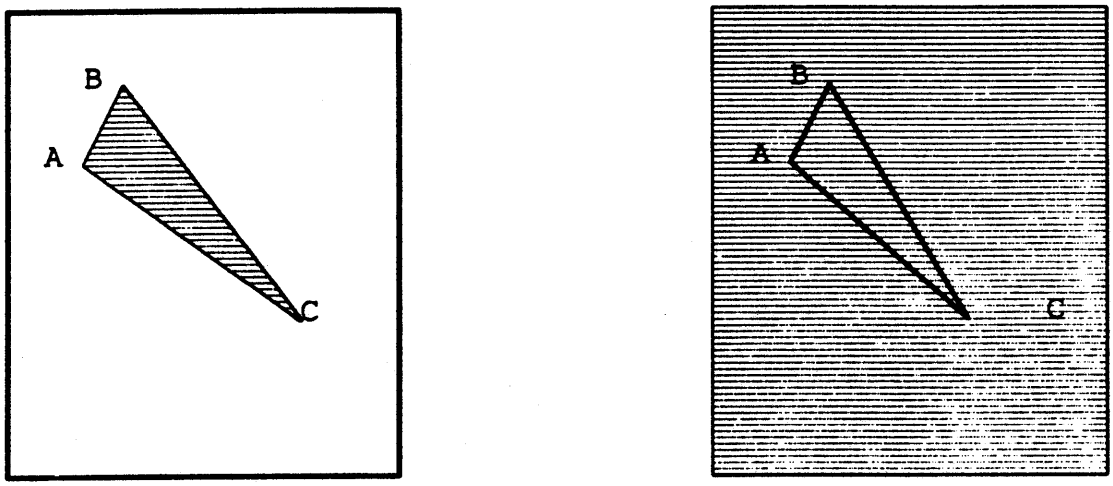
Example 2

- the *border mask*: the "line" connecting the current path as drawn with the current pen. Since the width of the pen is not zero, this "line" is really a polygon and, as such, has an interior. Since it has an interior, it can be "shaded" with the current texture. The next figure shows the border mask of the triangle from example 2:



Example 3

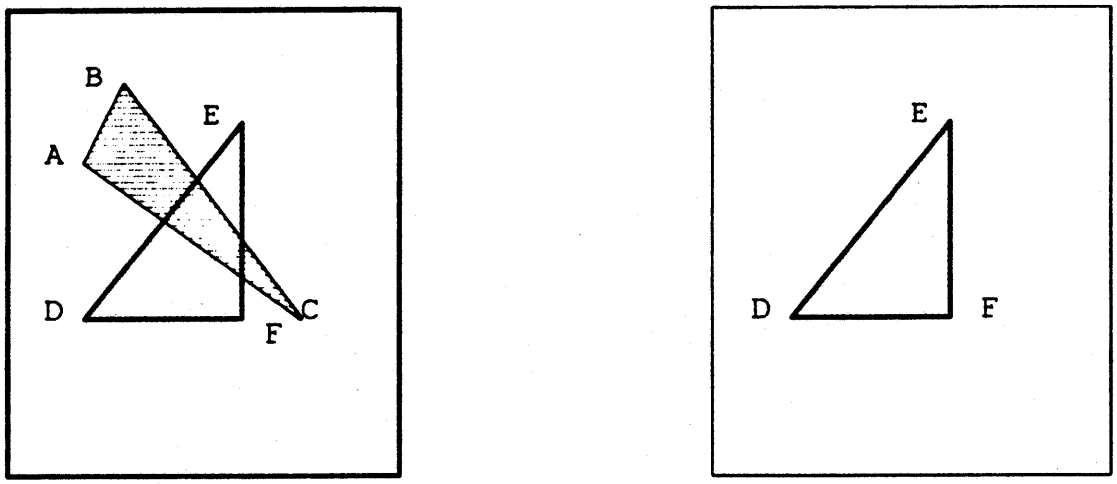
- a *graphic operation*: The four graphic operations determine how to fill in any region.
    - Operation 0 ("white"): Marks the current page with the color white. This won't be visible unless you are writing over something already present.
    - Operation 3 ("opaque"): Marks the page with whatever is in the corresponding part of the texture page; remember that the texture page matches the current page one to one. Thus, to fill in a triangle, Operation 3 copies whatever is in the corresponding triangle on the texture page onto the current page.
        - Example: The interior mask of triangle ABC has been filled from the texture page using operation 3. The right-hand block represents the texture page and shows how the triangle is "cut out" and then "pasted" onto the current page.



Example 4

- Operation 7 ("or"): Marks the current page by performing an "or" operation between the current page and the texture page. If the area you are shading is empty, the result will be the same as operation 3; but if it isn't, this operation will mark the page *without* erasing what is already there. This allows shapes to overlap each other.

  - Example: Operation 7 has been used to fill in the interior mask of the triangle DEF. Note that the pre-existing triangle has remained, even though it penetrates the new triangle. Again, the right-hand block represents the texture page.



Example 5

o Operation 15 ("black"): Marks the current page with black.

230: byte
vertex-count: word
[*h:* word *v:* word]$_{vertex\text{-}count}$

CREATE_PATH

Defines the "path" given by connecting the points (h,v) in the logical coordinate system. The two commands DRAW-PATH and FILL-PATH (below) will either draw these segments or fill in thearea inside them. Without using these commands, nothing will be printed. To define (and later print) a single point, set *vertex-count* to 1 and give one pair of coordinates. When drawn, this will appear as a dot the size of the current pen.

Since only one path can be current at a time, be careful to print out your old path before defining a new one; Impress will "forget" the old path as soon as it receives a new CREATE_PATH command.

231: byte
0: ubits(2) *family:* ubits(7) *member:* ubits(7)

SET_TEXTURE

Sets the current texture, a state variable, to the glyph identified by

<*current rotation, family, member*>,

where the glyph identification process is the same as used to find glyphs for printing (see section 9.8.8, page 85). The special case of *family* 0, *member* 0 always means a completely black texture, regardless of the actual glyph thus referenced. Currently, the width and height of any texture glyph must be 32.

203: byte
rotation: ubits(2) *family:* ubits(7) *member:* ubits(7)
pattern: ubits(32) *on:* ubyte *off:* ubyte *shift:* ubyte
operation: ubyte

MAKE_TEXTURE

The MAKE_TEXTURE command offers a simple way to down-load texture glyphs. It defines a glyphs of size 32 by 32 identified by

<rotation, family, member>

The string of bits *pattern* gives the first row of this glyph; the pattern

0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1

which we represent

...O...O...O...O...O...O...O...O

generates a first row in which every fourth pixel is darkened. The rest of the glyph is generated as follows: *Pattern* is repeated unchanged for *on* rows, the followed by *off* blank rows. Next, the pattern is shifted to the right *shift* spaces, repeated *on* times again and then followed by *off* rows. Note that a shift of 31 to the right is the same as a shift of 1 to the left. Thus *shift* never

needs to be negative. This process continues until the glyph is 32 pixels high, giving the correct size glyph for texture operation.

The *operation* variable lets you use the graphics operations explained earlier to define glyphs in repeated steps. Just as operation "7" allows shapes to be superimposed with the logical "or" operation, operation "7" here allows two or more texture definitions to be placed "on top of" each other. The other operations work similarly, but are less useful. This makes it easy to create a "cross-hatched" texture: first define a pattern of lines slanting to the right; then, using the same rotation, family, and member, define a pattern of lines slanting to the left. Example:

```
0 1 1 OOO..... 1 0  1 7
0 1 1 .....OOO 1 0 31 7
```

creates a simple cross-hatched texture by superimposing two groups of texture lines.

232: byte SET_PEN

diameter: byte

The pen's diameter, a state variable, is set to *diameter*, which must lie in the range [1, 20].

234: byte DRAW_PATH

operation-type: byte

The DRAW_PATH command draws the current path on the page with the graphic operation named by *operation type* (for example, operation 15 will draw the path as a black line). Any of the four operations may be used.

233: byte FILL_PATH

operation-type: byte

The FILL_PATH command colors the interior mask of the current path with the graphic operation named (for example, operation 3 will shade the interior with the current texture). If the current path is not closed, Impress will close it by connecting its endpoints. Note: The path being filled must be *simple*, that is, it must not cross itself. (A current implementation limitation makes it significantly faster to define paths used by FILL_PATH in a "clockwise" direction; that is, with the interior of the polygon to your right as you head "forward.")

235: byte BITMAP

operation-type: byte *hsize:* ubyte *vsize:* ubyte *bits:* see below

The BITMAP command takes the pixel rectangle given by *bits* (described below), aligns its upper-left-hand corner with the current (h, v) position, rounded down to the nearest multiple of 32 in each coordinate, and performs the graphic operation dictated by *operation-type* on the portion of the page image covered by the adjusted *bits* rectangle, using the *bits* rectangle itself in place of the imaginary texture page. Only the "opaque" and "or" graphic operations make sense with the BITMAP command.

The pixel rectangle *bits* is encoded as *vsize* vertically-stacked rows (top to bottom) of *hsize* horizontally-placed (left to right) "patches", each patch 32 vertically-stacked rows of 32 horizontally-placed bits. Each patch is thus 128 bytes, and the entire rectangle occupies

$$128 \cdot hsize \cdot vsize$$

bytes.

In simple terms, the BITMAP command supplies a rectangle of bits which are placed at the current position on the page, either replacing what's there, or combining with it. NOTE that the BITMAP command obeys the *h, v* logical coordinate system, so that the bit-map will be rotated accordingly. Also note that page magnification (see section 9.11, below) is often useful with the BITMAP command, when it is used to print a high-resolution screen image; in this case, a doubling of the screen image results in reasonable dimensions for screens of approximately 1K by 1K pixels.

Here is an example of these commands which draws a triangle, filling it opaquely with the gray texture glyph given by family 1, member 20, and partially outlining it in black with a pen of diameter 10. (This example assumes that the texture glyph has already been defined in some fashion.)

```
@CREATE_PATH 3  100 100  100 200  200 200@
@SET_TEXTURE 1 20@
@FILL_PATH 3@
@SET_PEN 10@
@DRAW_PATH 15@
```

Note that the path is not closed (the first and last points don't coincide); thus, the DRAW_PATH will not draw the third side, resulting in a black "L"-shaped outline around the grey triangle.

Currently, the text and graphics "planes" are distinct, and are logically "or"ed together for printing; i.e., no text commands can affect the graphics plane, and no graphics commands can affect the text plane. This is a restriction, and will it be lifted in future versions of Impress: there will only be one plane for both text and graphics, and graphic operations such as FILL_PATH will be able to affect the text in the page image.

Magnification Command (9.11)

236: byte
power: byte

SET_MAGNIFICATION

The SET_MAGNIFICATION command sets the magnification for the current page to 2^{power}, where *power* must be 0, 1, or 2, currently, requesting magnifications of 1, 2 and 4, respectively. Subsequent attempts to set the current page's magnification are ignored. The default magnification for a page is 1.

When a page is printed, it is magnified appropriately, *fixed at the physical page origin [0, 0].* For example, if a page has a magnification of 2, only the upper-left-hand quarter of the virtual page image will be visible. Thus, setting a page magnification is effectively reducing the virtual page size by that same factor.

Currently, only the Impress graphics plane is magnified. This restriction will be lifted in future versions of Impress.

State Saving and Restoring (9.12)

At any point, the system has a certain *state* described by a number of *state-variables*. The effect of many IMPRESS commands depends on the value thesevariables have at any given time. The complete list of state variables, and the commands that set their value, is:

pen size:	SET_PEN
texture:	SET_TEXTURE
interword space size:	SET_SP
inter-line space size:	SET_IL
left margin:	SET_BOL
family:	SET_FAMILY
h v position:	SET_ABS_H, SET_ABS_V, etc.
advance directions:	SET_ADV_DIRS
Origin of the *h v* system:	SET_HV_SYSTEM

If you could save these variables, you could stop at any point in the document, do something else, and return to what you were doing before without troule. Impress provides three commands to do this: the PUSH command saves the current state in a "stack," the POP command restore the state at the top of the stack, and the SET_PUSH_MASK lets you define which state variables you want saved whenever you use the PUSH command. For example, if you were doing a plot with the origin in the upper left hand corner but suddenly wanted to move the origin to the center for a few operations, you could PUSH the current state (origin in the corner), move the origin to the center and do whatever is needed, and then POP the state which automatically moves the origin back to the corner where it was before. Note that, unless you used the SET_PUSH_MASK command, this would also return you to your previous *h v* position.

SET_PUSH_MASK

214: byte
0: bits(7)
pen-and-texture: bit
interword-space: bit
interline-space: bit
beginning-of-line: bit
family: bit
hv-position: bit
advance-directions:
origin: bit
orientation: bit

SET_PUSH_MASK sets the push mask from its operands. SET_PUSH_MASK defines which variables the PUSH command will save. Except for the first operand, which controls both the pen size and the graphics texture, each operand controls a single state variable; therefore, there are nine operands, corresponding to the ten variables listed above. If an oerand is "1," the corresponding variable will be saved when you PUSH the state. If an operand is zero, its value will

not be saved and will be lost if at any point you change it. In most cases, you will want to save the entire state of the printer (i.e. all ten variables). Therefore, the push-mask is is set to save everything at the beginning of each document, and will not change unless you alter it explicitly by using this command.

211: byte **PUSH**

The PUSH command saves the state variables as dictated by the current push mask, which, in turn, is manipulated by the SET_PUSH_MASK command, above.

212: byte **POP**

The POP command restores the state variables saved by the most recent unmatched PUSH command. Note that the variables restored are exactly those given by the push mask at the time the matching PUSH command was encountered.

Macros (9.13)

Impress provides a simple but complete macro facility. Macros are named by an ordinal in the range [0, 255]. A macro is a parameterless sequence of *complete* Impress commands. When the macro is invoked, the sequence of commands is treated as ordinary document input.

242: byte **DEFINE_MACRO**
macro-name: ubyte *body-length:* word
[body-byte: byte *]*$_{body\text{-}length}$

The DEFINE_MACRO command defines the macro named *macro-name*, with body the sequence of *body-bytes*, of length *body-length*. This body is entirely uninterpreted at definition time; it can contain anything, but it should contain a complete sequence of Impress commands. Any errant Impress commands in its body will be detected at EXECUTE_MACRO time.

243: byte **EXECUTE_MACRO**
macro-name: ubyte

The EXECUTE_MACRO command treats the body of the macro named by *macro-name* as ordinary Impress document input, resuming document input at the point immediately after the EXECUTE_MACRO, once finished with the macro body.

Even though macros have no parameters, the reasonably large macro name space permits a simple parameter scheme: the macro names 0 to 32 (say) could be reserved by an Impress document for parameters (which would be macros themselves). Then, for example, the "caller" of a macro with two arguments would define macros named 0 and 1 before calling, and the called macro would execute the macros named 0 and 1 to effectively "pick up" its parameters.

Simple Impress (9.14)

To compose simple documents with Impress, you need use only a little of the full command set. For example, to make a simple "line-printer listing," you can do the following:

- set the current family to 2 (chosen arbitrarily):
  ```
  @SET_FAMILY 2@
  ```
- map family 2 to the resident font COUR12 (a typewriter-style, fixed-width font, with nominal advance width of 23):
  ```
  @CREATE_FAMILY_TABLE 2 1 0 COUR12 @
  ```
  (this uses the built-in member map 0 to map all members directly to the ASCII symbols of COUR12)
- set the beginning-of-line to 240 (leaving an inch on the left margin)
  ```
  @SET_BOL 240@
  ```
- set the inter-line spacing to 24 (10 lines to the inch):
  ```
  @SET_IL 24@
  ```
- set the inter-word spacing to the nominal advance width of our font:
  ```
  @SET_SP 23@
  ```
- for each page of text to be printed, do the following:
  - position to (240, 240) (leaving an inch for the left and top margins):
    ```
    @SET_ABS_H 240@
    @SET_ABS_V 240@
    ```
  - output words of text with ordinary ASCII characters (considered by Impress to be MEMBER commands, of course), separated by SP commands, lines separated by CRLF commands:
    ```
    This @SP@ is @SP@ a @SP@ first @SP@ line. @CRLF@
    This @SP@ is @SP@ the @SP@ second. @CRLF@
      . . .
    ```
  - finally, end the page image with an ENDPAGE command:
    ```
    @ENDPAGE@
    ```

Changes from Version 0 (9.15)

The current Impress, version 1, is a compatible extension of Impress version 0, described in the *IMPRINT-10 System Handbook* dated June, 1982. Beyond the extensions, there are several fundamental changes from version 0:

- The old Impress "job header" structure has been replaced by the more general IMPRINT-10 document control language (see Chapter 3), leaving only the actual body of the Impress document relevant to this chapter. (Impress is now only one of several languages supported by the IMPRINT-10.)
- There are no longer any "pre-page" and "post-page" contexts; all commands are licit at any point in a document.
- The memory allocation issues are entirely different, as the IMPRINT-10 system itself is completely new internally. In particular, there are no pre-allocated areas for glyph, input, or compressed page image storage. Rather, the glyph table and page image storage compete with each other for space.

Messages (9.16)

Impress can generate the error messages below during document processing. The components of messages listed here in brackets are the occurrence detail data; for more information about messages, see Chapter 7, section 7.2.2, page 56, in particular. Note that any occurrence detail will be prefixed with the number of the page containing the error. The standard recovery action is to ignore any errant command, unless otherwise specified.

`Font file not found` [*fontname*]
A font file given as a sub-operand to a CREATE_FAMILY_TABLE command is not present in the IMPRINT-10's file system.

`Glyph off page` [*x y*]
The document is attempting to print a glyph at (physical position) (*x*, *y*), but this position is outside the page image.

`Invalid family` [CREATE_FAMILY_TABLE]
The *family* operand to CREATE_FAMILY_TABLE is outside the range [0, 127].

`Invalid map name` [CREATE_MAP]
The map name (*map-name* operand) given to a CREATE_MAP command is outside the range [0, 127].

`Invalid path` [FILL_PATH]
The current path given to FILL_PATH crosses itself (i.e., is not a simple polygon).

`Invalid texture dimensions` [*rotation family member*]
The texture glyph identified by

<*rotation, family, member*>

does not have the proper dimensions for a texture (currently, height and width must be 32).

`Not enough memory` [CREATE_MAP]
Impress doesn't have enough memory to store the member map being defined by a CREATE_MAP command.

`Not enough memory` [DRAW_PATH]
Impress doesn't have enough memory to complete a DRAW_PATH command.

`Not enough memory` [FILL_PATH]
Impress doesn't have enough memory to complete a FILL_PATH command.

`Invalid size` [CREATE_PATH]
The number of vertices (*vertex-count* operand) in a CREATE_PATH command is negative.

`Not enough memory` [DEFINE_MACRO]
Impress doesn't have enough memory to store the macro being defined in a DEFINE_MACRO command.

`Not enough memory` [CREATE_PATH]
Impress doesn't have enough memory to store the path being defined in a CREATE_PATH command.

`Not enough memory for glyph` [*rotation family member*]
The glyph identified by

<*rotation, family, member*>

is being defined either directly (BGLY) or indirectly (via a family table), and there is not enough memory to store the glyph definition, so it is being ignored. Note that `Undefined glyph` messages will probably show up later because of this ignored definition.

`Path off page [x y]`
The document is attempting to draw or fill a path at (physical position) (x, y), but this position is outside the page image.

`Rule off page [x y]`
The document is attempting to print a text rule at (physical position) (x, y), but this position is outside the page image.

`Undefined glyph` [*rotation family member*]
The document is attempting to print the glyph identified by

<*rotation, family, member*>

but this glyph is undefined: it has not been defined by an BGLY command, *family* has not been mapped by a CREATE_FAMILY_TABLE, or *family* has been mapped, but none of the member maps lead to a real glyph. Impress marks the point in error with a special undefined glyph symbol, a "U" inside a box.

`Undefined texture` [*rotation family member*]
Same as `Undefined glyph` (see above), but for a texture glyph. The texture "black" is used in place of the missing texture glyph.

Document Control (9.17)

Impress takes no document control information directly. However, the document control items `paper`, `paperwidth`, and `paperheight` affect Impress's idea of the page image size (see section 9.6, page 71, and section 3.3.3.1, page 21). And, the `messagedetail` document control item turns error message detail on or off (see section 3.3.3.2, page 22).

CHAPTER 10
Printer Language

Specification (10.1)

The Printer document language provides an emulation of a generic character printer, using the 7-bit USASCII character set.[8] It prints all 96 of ASCII's graphics and obeys all of its standard formatting controls. A Printer document can declare its logical page sizes, margins, and "look" (outlines and line rules), as well as how many logical forms it wants per physical page.

In more detail, the Printer language supports the following set of graphics (decimal byte values ranging from 33 to 126, in order):

```
   !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /
0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?
@  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
P  Q  R  S  T  U  V  W  X  Y  Z  [  \  ]  ^  _
`  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o
p  q  r  s  t  u  v  w  x  y  z  {  |  }  ~
```

It also supports the following set of controls, given with their decimal values and effects:

BS (8) backspace one character position (ignored in first position of line)

HT (9) (horizontal tab) space over to the next tab stop (every 8 print positions)

LF (10) (line feed) advance to the beginning of the next line, or to the beginning of the next form if the current form is vertically full (see below)

FF (12) (form feed) advance to the next form

CR (13) (carriage return) return to the first position of the current line

SP (32) (space) advance one position, leaving it blank

All other controls, along with the character with value 127 (the ASCII DEL), are ignored.

Any line of graphics longer than the form width will be truncated at the right margin, with a warning message (see section 10.4, below).

[8] the high order bit of each document byte is ignored

Document Control (10.2)

The document control language items specific to the Printer language control the form (logical page) placement, size and appearance. They are as follows, using the same notation as in section 3.3 (page 20):

`formlength` *int*
: declares the maximum number of lines per form (default `formlength 60`, maximum 66)

`formsperpage` *int*
: declares the number of logical pages per physical page, currently either 1 or 2 (default `formsperpage 1`)

`formwidth` *int*
: declares the maximum number of printing positions per form line (default `formwidth 80`, maximum 132)

`leftmargin` *int*
: declares the number of left-most printing positions to be left blank on each line (default `leftmargin 0`, must be less than `formwidth`)

`outlines` *bool*
: declares whether form outlines are used (default `outlines off`)

`rules` *bool*
: declares whether form rules (one every two lines) are used (default `rules off`)

`topmargin` *int*
: declares the number of blank lines left at the top of each form (default `topmargin 0`, must be less than `formlength`)

Practica (10.3)

You may note that there is no explicit way to fully specify a given physical page layout; this is intentional. The document requests what it wants in terms of form heights, widths and number of forms per physical page, and the Printer emulator will choose the layout which makes the most sense under these constraints and under its own constraints.

Currently, the physical layouts are chosen from amoung these four:

- one form per physical page, "portrait" orientation, 12-point typewriter style font: when `formsperpage` is 1 and `formwidth` is less than or equal to 80
- one form per physical page, "landscape" orientation, 10-point typewriter style font: when `formsperpage` is 1 and `formwidth` is greater than 80
- two forms per physical page, one beside the other, "landscape" orientation, 8-point typewriter style font: when `formsperpage` is 2, and `formwidth` is less than or equal to 80
- two forms per physical page, one above the other, "portrait" orientation, 7-point typewriter style font: when `formsperpage` is 2 and `formwidth` is greater than 80

Messages (10.4)

The following messages may be produced by the Printer language emulator. See section 7.2.2, page 56 for more information about emulator messages and their format.

`Invalid formlength value (66 assumed)`
: The given `formlength` value is either not a decimal number or is outside the permitted range (see the `formlength` description in section 10.2, above)

`Invalid formsperpage value (2 assumed)`
: The given `formsperpage` value is either not a decimal number or is not either 1 or 2, currently (see the `formsperpage` description in section 10.2, above)

`Invalid formwidth value (132 assumed)`
: The given `formwidth` value is either not a decimal number or is outside the permitted range (see the `formwidth` description in section 10.2, above)

`Invalid leftmargin value (zero assumed)`
: A given `leftmargin` value is either not a decimal number or is outside the permitted range (see the `leftmargin` description in section 10.2, above)

`Invalid topmargin value (zero assumed)`
: A given `topmargin` value is either not a decimal number or is outside the permitted range (see the `topmargin` description in section 10.2, above)

`Truncated line`
: A line is too long for the document's declared `formwidth` (and given `leftmargin`): the excess portion of the line is merely truncated. Any instance data gives the character ordinal at which the line was truncated.

CHAPTER 11
Daisy Language

Introduction (11.1)

The Daisy language allows the Imprint-10 to emulate a Diablo 1640 daisy-wheel printer. This language makes the printer "look like" the Diablo as much as possible, duplicating its commands and abilities so that it can be used without extensively modifying your existing software. The Daisy language allows:

- fixed width characters
- variable letter and line spacing
- font changes mid-text
- simple subscripts and superscripts
- margin-setting on all four sides of the page
- forward and reverse printing

Since Daisy emulates a line-oriented printer, its "movements" are all in terms of lines and spaces. It has no coordinate system beyond this, nor does it have any provision for absolute moves on the screen; i.e. you can't say "go to a particular spot and begin printing." To get from one spot on the page to another, you must use spaces, carriage returns and line feeds, as on a typewriter.

Definitions (11.1.1)

To keep track of character spacing, Daisy uses Diablo's notions of Horizontal and Vertical Motion indices. The *Horizontal Motion Index* (HMI) is similar to the *Advance Width* in Impress: it specifies the distance between characters in units of 1/120 inch. The most common values for HMI are 10 and 12, corresponding to 10 or 12 pitch type (Elite or Pica). The *Vertical Motion Index* (VMI) is analagous to the inter-line spacing of Impress: it specifies the distance between lines in units of 1/48 inch and has a default value of 8 (6 lines per inch).

Encoding (11.2)

Daisy commands are encoded as seven-bit ASCII characters. Each command occupies from one to three bytes, with the exception of `Assign-font`, explained below. Text is transmitted simply as ASCII.

Commands (11.3)

This section explains Daisy's commands and their functions. The command code for each command is given by its equivalent in ASCII. In these descriptions, "Esc" means "escape," "Ctrl" means "control," and "Return" means "carriage-return." For clarity, we have enclosed control characters (which look like dual characters but only occupy one byte) in angle brackets: <Ctrl X>. For example, Esc return P means that this command (clear all margins) has the three-byte code corresponding to the ASCII character string "Escape, carriage return, capital P." In decimal, these three bytes are 27 13 80; similarly, Esc <Ctrl=> has a two byte code, 27 30. Daisy will ignore Diable 1640 commands that it has not implemented.

Formatting commands (11.3.1)

These commands set the page margins and the variables VMI and HMI (vertical and horizontal motion indices) to control line and letter spacing. Daisy sets margins at the current position; that is, whenever a command to set a margin occurs, the margin will placed at the printer's current position on the page. This makes it easy to move the margins further in; to move them out, you will need to execute a command to clear the margins (either Esc C or Esc return P, below). These commands set the margins to 0; that is, they set the margins to the edge of the paper. After clearing the margins, use a carriage return or a form feed (bringing you either to the beginning of a new line or the top of a new page) and use spaces or line-feeds to move in to the correct space and set a new margin.

Here are Daisy's margin and formatting commands:

Esc 9	Set the left margin at the current position.
Esc 0	Set the right margin at the current position.
Esc T	Set the top margin at the current position.
Esc L	Set the bottom margin at the current position.
Esc C	Clear the top and bottom margins. Set the top and bottom margins to the top and bottom of the page.
Esc Return P	Clear all margins. This command sets all margins equal to the page boundaries.
Esc <Ctrl ->	Set HMI to the value given in the next byte.
Esc <Ctrl =>	Set VMI to the value given in the next byte.

Text commands (11.3.2)

This group of commands does the actual printing; they print text, do line feeds, change fonts, etc.

Line feed (<Ctrl J>)
: Advance to the next line. This command moves the printer down the page by the value of VMI, but keeps the same horizontal position. It does not include an implicit carriage return. To move to the beginning of a new line, you need separate Carriage Return and Line Feed commands. If you attempt to cross the bottom margin with a line-feed, Daisy will automatically print the current page and begin the next; i.e. it will perform an automatic form-feed.

Esc Line-feed (Esc <Ctrl J>)
: Move up the page to the previous line; this is called a "negative half-line-feed."

Esc U	Half line feed (i.e move down the page by half the value of VMI); useful for subscripts
Esc D	Negative half line feed (i.e. move up the page by half the value of VMI); useful for superscripts
Form feed (<Ctrl L>)	Prints the previous page and advances to the next. After advancing to the next page, you will be at the top left-hand margin.
Return	Carriage return; moves to the beginning of the current line.
Tab (<Ctrl I>)	Move forward to the next tab stop. Tab stops are automatically set every eight positions across the page; this conforms with the meaning of a tab for most editors.
Esc 5	Forward print on; print text in the forward direction.
Esc 6	Backward print on; print text backwards. This command lets Daisy work with systems where the printer prints from left to right then returns by printing backwards from right to left. When printing backwards, text must be sent to Daisy with its order reversed; if Backward Print is on, the string "sdrawkcaB" will print the word "Backwards" correctly.
Space	Advance forward one space (or, if backward print is on, move backward one space. Spaces are always in the right direction for printing a normal text).
Backspace	Move backward one space (forward if backward print is on)

Each ASCII code for a printable character is a command in its own right; it means "print this character in the current position and advance to the next." If Daisy reaches the end of a line before it finds a carriage return, it will print a black square at the margin. This is the equivalent to letting the characters pile up at the end of a line on a typewriter. Daisy will not perform an automatic carriage return, and it has no equivalent to a typewriter's "margin release" key. Note that if you are using the Backward Print feature, this can occur at either margin.

Daisy enhancements (11.3.3)

The following two commands allow you to change fonts in the middle of a Daisy text. Since this is analogous to physically changing the print wheels, these commands have no equivalent to any of the command codes available on a Diablo printer. Both of these commands have a more complex structure than the other Daisy commands; one or more arguments follows the command code. The descriptions below give the command code, followed by the arguments, and then provide descriptions of what these arguments mean.

Assign-font command:

Esc + (*d*: byte) (*name*: string)

Two arguments, *d* and *name* follow the two-byte command code, Esc +. The first argument *d* is an ASCII character between 0 and 9 (i.e. with a decimal value between 48 and 57). The second argument, *name*, is a sequence of characters ending with a line feed (ASCII 10) that names some resident font in the Imprint-10 system. This command assigns the font *name* to the value *d* for use by the `change-font` command described below. It performs no immediate action. You cannot reassign fonts; each of the ten possible values of *d* may only be used once in any document.

Change-font command:

Esc F (*d*: byte)

A single argument, *d*, follows the two-byte command code, Esc F. *d* can be any ASCII character between 0 and 9 plus the character ":" (i.e. any character with a decimal value between 48 and 58). If *d* is a character between 0 and 9, this command will change the font to whatever font has been assigned the value *d* (see the `assign-font` command, above). If *d* is a colon (":"), this command changes the font to whatever was specified in the Document Control statement.

Examples: The command string `Esc + 2 COUR12 Line-feed` assigns value 2 to the Courier 12 font. After this you have made this definition, the string `Esc F 2` will make Daisy switch from whatever font is in use to Courier 12. At any time, the command string `Esc F :` will force Daisy to use the font specified in the document header.

Unimplemented commands (11.3.4)

The Diablo printer's commands for automatic text justification, graphics, boldface printing, and shadow printing have not been implemented; most typesetting programs have better ways of doing all these things (like changing fonts for boldface instead of overstriking). Daisy will ignore these commands.

Document Control (11.4)

When you are using Daisy, the Document Control Language will let you specify the initial font and initial values for HMI and VMI. The three relevant keywords for the document header are:

`font` *fontname*

This item lets you select the initial font for Daisy; Daisy will use this font unless you specify something else explicitly in the body of your document. *Fontname* must be a string of characters naming one of the Imprint-10's resident fonts. The default font is Courier 12 (`COUR12`), a fixed-width typewriter style font.

`HMI` *number*

This item lets you specify the initial value of HMI that Daisy will use. As explained in section 11.1.1, HMI determines the space size and the space between characters in 120th of an inch. For example, `HMI 20` makes the space size and the advance width for each character $20/120^{ths}$ inch (1/6 inch), thus printing 6 characters per inch. The default value for HMI is 12 (10 characters per inch).

`VMI` *number*

This item lets you specify the inital value of VMI. As explained in section 11.1.1, VMI determines the spacing between lines; it is measured in units of 1/48 inch. For example, the item `VMI 24` would make the space between lines 1/2 inch; subscripts and superscripts, which appear a half space above or below the line, would then be 1/4 inch. The default value for VMI is 8 (6 lines per inch).

A complete document header specifying all of these would look like this:

```
@document (language daisy, VMI 13, font cour12, HMI 42)
```

For more information on the document header, see chapter 4.

Messages (11.5)

If you do something wrong, Daisy will print the following error messages on the job header page:

`Invalid HMI value (12 assumed)`
: There is a document control statement which attempts to assign HMI a non-numeric value. Daisy has ignored this assignment and assumed the default value of 12. Note that this error can only occur in the document control statements; HMI assignments within the document itself are *always* legal, whether or not they make sense.

`Invalid VMI value (8 asumed)`
: There is a document control statement which attempts to assign VMI a non-numeric value. Daisy has ignored this assignment and assigned VMI the default value, 8. Again, this error can only occur in a document control statement.

`Invalid font declaration ignored`
: This message means that the *d* argument in the `assign-font` command is an ASCII character that does not fall within the range 0 to 9 (48-57 in decimal). Daisy has ignored this font declaration. This will probably lead to `Invalid font change` messages later. See section 11.3.3.

`Font redeclaration ignored`
: You have tried to assign a font to a *d* value that has already been used; once made, an assignment cannot be changed. Daisy has ignored the new assignment.

`Font not found (`*fontname*`)`
: The font *fontname* is not one of Imagen's resident fonts.

`Invalid font change`
: This message signifies a problem with a `change-font` command. There are two possibilities: First, you may have given an illegal value for the argument *d*; i.e., *d* is not a number between 0 and 9 or the special character ":" Second, if *d* is a legal value, you may not have assigned it to any font. In this case, there may be an incorrect font declaration earlier that has been ignored. In either case, Daisy will continue to use the old font and ignore the command.

CHAPTER 12
Tektronix Language

Introduction (12.1)

The Tektronix language allows you to use the Imprint-10 to emulate a Tektronix 4014 (or 4014-1) terminal. Documents using this language will be printed as they would appear if they were displayed on these terminals. Features we provide include:

- fixed-width characters in four different sizes
- "same-page" graphics and text
- automatic carriage return line-feed option

Of course, certain things can't be done; the Imprint-10 is only a printer, not an interactive terminal, and therefore several facilities Tektronix provides (like interactive graphics) can not be implemented. Features we cannot provide will be listed in section 12.3.3.

Modes (12.2)

The Tektronix terminal has two different operating modes: one for printing text, the other for printing graphics. The Tektronix language simulates both of these modes. Each of these modes has its own basic "rules," which will be explained below.

Alpha mode (12.2.1)

In the Alpha mode, the emulator will print regular alpha-numeric text. It does not really have a coordinate system; rather, it organizes pages like a typewriter does, according to lines and spaces. Therefore, the alpha mode does not allow any absolute movements on the page; you can only move around on the page with different combinations of spaces, backspaces, returns, and linefeeds. These pages start at the upper-left-hand corner of the page when it is held the *long* way[9]; printing proceeds from left to right across the page (the long way). After printing a page, the emulator will automatically begin the next page in the upper-left corner. We call the spot in this corner where printing begins "home."

Tektronix will provide automatic carriage returns and line feeds under some conditions. Any attempt to print a character past the page's right magin will make Tektronix advance to a new line and print that character on this line. If you attempt to backspace past the page's left margin, Tektronix will "wrap back" to the right-hand end of the same line. This holds

[9] In previous chapters, we have called this "landscape" orientation

true which ever margin is in effect; remember that the left margin may, under some circumstances, be in the middle of the page. Furthermore, a document control option allows you to tie the carriage return and linefeed commands together.

Whenever there is a carriage return, the printer will return to the left margin. In Tektronix, the margins are fixed; you cannot change the margin widths. However, the language provides two possible left margins; one is at the left side of the page, underneath the "home" position, and the other is in center. In most circumstances, Tektronix will use the margin on the left side. It will use the center margin if you attempt to move past the bottom of the page with a line-feed. Note that this includes line-feeds generated automatically, either by trying to put more characters on one line than it will hold, or by tying the line-feed and carriage return commands together with the `cr-is-crlf` option (see section 12.4). Any attempt to cross the bottom of the page will make Tektronix change to the other margin; if you cross the page bottom once, forcing Tektronix to select the second margin , and then cross the bottom margin again, Tektronix will return to the first margin at the far left side of the page. At this point, you may well be writing over material already there. This will probably happen only if you make some mistake; however, since Tektronix never prints pages automatically, it is easy to imagine this kind of error happening.

In the Alpha mode, you can select between four different character sizes: 8, 10, 12, and 14 points. The emulator automatically selects the proper character and line spacing to print these characters correctly. As a consequence, changing fonts will make it difficult to return to any particular location on the page after you have left it. For example, if you print an 8 point "x," space forward 4 spaces, change to 12 point type, do four backspaces, and print an 8 point "z," the result will be "zx," *not* "xz." The size of a space has changed partway through the text; the 12 point backspaces do not match the 8 point spaces.

Unlike most languages, the "backspace" in Tektronix does not erase what was previously there; backspacing lets you type over characters previously there, but does not let you delete them.

Graphics Mode (12.2.2)

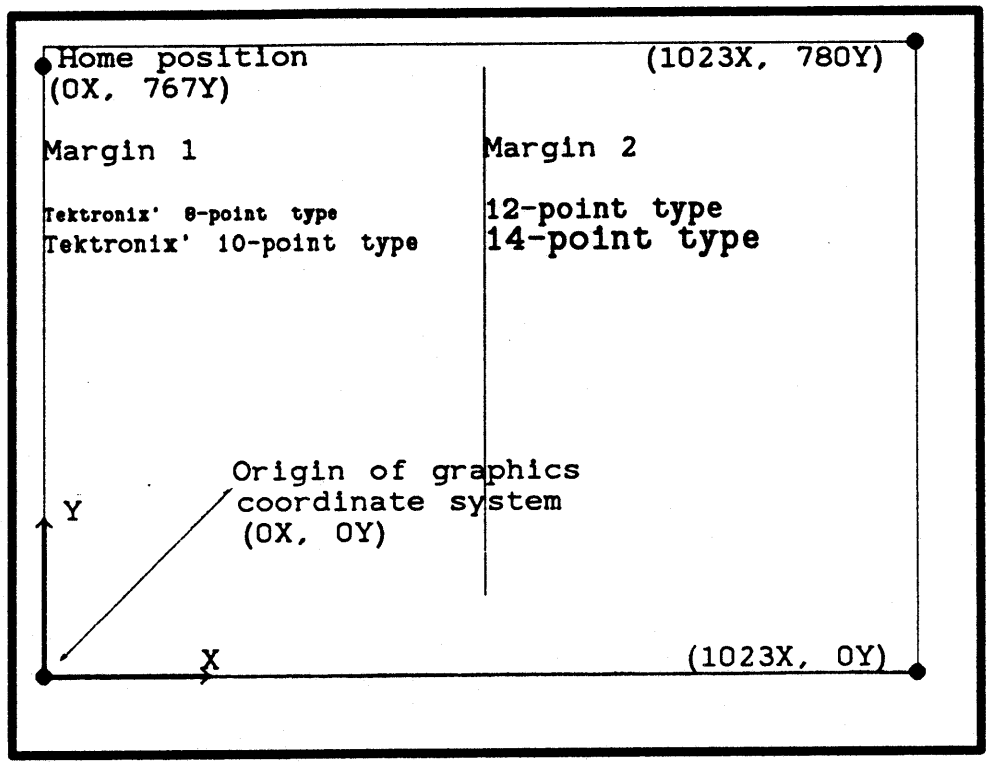
The Graphics mode is used, simply enough, to print graphics. Again, this language uses the paper in "landscape" orientation, i.e. short and wide. This mode organizes the page in terms of an x-y coordinate system with its origin in the lower left hand corner of the page. The x axis proceeds fron left to right across the page, with x increasing from 0 to 1023. The y axis proceeds from the bottom of the page to the top, the y ranging between 0 and 780. In Tektronics coordinates, the location of the alpha mode's "home" position is (0,767). In this system, 1 unit equals $1/120^{th}$ inch; that is, one Tektronix unit equals two Imprint-10 pixels. Consequently, if resolution is important to you, you would be better off using the Impress language for graphics.

In the graphics mode, you can print vectors, or lines, at arbitrary angles across the page. You specify a chain of vectors by their endpoints; three coordinate pairs, for example, specify two vectors (conceivably only one), which would be printed as two connected lines. If you only specify one point, nothing will happen; there are no single point vectors in Tektronix. Should you want to draw a single point, give its coordinate twice and print the vector. This will print a single point -- or, for Tektronix' sake, a vector with length zero. More advanced features, like bit-map graphics, are not possible.

Tektronix assumes that you will not want to draw graphics for more than one page at a time. Therefore, whenever you print a page in the Graphics mode, the system will automatically return to the Alpha mode, positioning you in the "Home" position at the top of a new page.

Summary (12.2.3)

This diagram shows how a Tektronix page is organized, along with the four fonts available.



The Tektronix page

Commands (12.3)

The Alpha mode and the Graphics mode both have their own command sets; the same command codes appear in both sets, but their meanings differ. At the beginning of any page, and at the beginning of any document, Tektronix will automatically be in the Alpha mode, no matter what mode it was in previously. Therefore, don't be careless and "forget" what mode you are using at any time; be aware that mode changes may occur without some explicit command on your part.

In the following descriptions, we give commands by their ASCII equivalents. Most of these commands are "control" codes, corresponding to some alpha-numeric key used with the control key. A table at the end of this chapter will show you what codes correspond to what keys. Some terminals may already have the ASCII control codes printed on the keyboard, above the customary alpha-numeric symbols. Several commands occupy two bytes; these all begin with the "escape" key, which we have abbreviated "Esc." On some terminals, it may have another name, like "altmode."

Alpha mode commands (12.3.1)

In the Alpha mode, two things can happen: you can print text, and you can change to the graphics mode. The following list shows the commands available in the Alpha mode and their function:

Esc 8	Use the largest characters available. These are in 14 point fixed-width type. With this font, a line hold 74 characters, and a page holds 35 lines.
Esc 9	Use the 12 point font. This allows 81 characters per line, 38 lines per page. (Note: This is the same size as Pica type)
Esc :	Use the 10 point font. 121 characters per line, 58 lines per page. (Note: This is the same size as Elite type)
Esc ;	Use the 8 point font. 133 characters per line, 64 lines per page.
Esc FF	Print the current page and begin a new one, starting at the "Home" position (the upper left-hand corner).
Esc Carriage return	Ignore carriage returns from this point on. Any further carriage returns will have no effect[10]. The command BEL (see below) will turn this feature off.
Esc LF	Move horizontally to the left margin. This sequence is a *carriage return* not a line feed; however, it differs from a regular carriage return in that it *always* works, whether or not you are in the "ignore carriage return" mode. The `cr-is-crlf` option also ties this command to a line-feed, just like a regular carriage return.
BEL	Disable the "ignore carriage return" condition.
BS	Backspace; move one space in the reverse direction. Backspacing past any left-hand margin will place you on the far right side of the page.
HT (Tab)	Move one space to the right. Moving past the right margin will place you at the beginning of the next line, generating an automatic carriagereturn-line feed pair.
LF (Line feed)	Move one line down the page. Moving past the bottom margin will place you at the *top* of the same page and change the margin to the one not in use.
VT	Reverse line-feed; move one line up the page.
CR	Move to the current left margin without changing to the line (Note: the `cr-is-crlf` option can be used to couple this with an automatic line feed).
GS	Enter the Graphics mode; begin accepting graphics commands.
SP	Move one space to the right.

In the Alpha mode, any ASCII code corresponding to a printable character will print that character in the current position. Tektronix will ignore non-printing characters, unless they are commands. This includes the "Escape" character, which will be ignored unless it is part of a command.

[10] We are not sure why anyone would want to do this; but this is a feature of the 4014

Graphics commands (12.3.2)

Here are the commands used in the Graphics mode. Some have the same function as in the Alpha mode.

Esc CR	Same as in Alpha mode.
Esc FF	Print the current page, enter the Alpha mode, and begin a new page. The next page will begin in the "Home" position.
BEL	Same as in Alpha mode.
CR	Enter the Alpha mode and perform a carriage-return; that is, begin on the next line after the end of the current vector.
GS	Draw the current vector on the page. The next point you give will be treated as the first point of a new vector, not an additional point on the old one. This command essentially tells Tektronix to start drawing a new line.
US	Enter the Alpha mode without changing location; printing will begin at the end of the last vector. If you use this cleverly, it will allow you to move to absolute locations on the page.

Specifying the location of a point in Tektronix takes four bytes; each coordinate occupies two bytes. Not all four bytes need to be sent every time; if one of the coordinates does not change, some of these bytes can be eliminated.

In Tektronix, we use bytes with values between 32 and 127 to specify coordinates. Bytes between 32 and 63 specify the high portion of either an x or a y coordinate; bytes between 64 and 95 specify the low portion of an x coordinate; and bytes between 96 and 127 specify the low portion of a y coordinate. Essentially, this means the following: An ASCII character occupies 7 bits of an 8 bit byte; therefore, we ignore the first bit (bit number 0) of any byte. We use the next three bits (bits 1 and 2) to determine what to do with the byte. If these bits are both 0, then the byte must be one of the command codes. If these bits are (in order) 0 1, then the byte must have a value between 32 and 63; we will use it as the first part of a y coordinate. If these bits are 1 0, then the byte has a value between 64 and 95, and we will use it as the "low" part of the x coordinate. Finally, if both these bits are one, the byte must be between 96 and 127; consequently, we will use it as the "low" part of a y coordinate.

This leaves the five bits at the end of each byte to specify a location. Putting two of these five bit chunks together gives you a complete 10 bit coordinate that can specify a point anywhere on the Tektronix page. Of course, the "high" part of the coordinate comes first, followed by the "low" part. However, even though each byte specifies its function in a pair of coordinates, the four bytes giving any coordinate pair cannot be sent in an arbitrary order. They must be sent in this order:

High y, Low y, High x, Low x

If you do not need to change all these bytes from one point to the next, you need not send the entire sequence. For example, if you want to draw a vector parallel to one edge of the page, only one coordinate (and hence two bytes) will change from one point to the next. This feature helps if you are doing a graph that requires a lot of small shifts in position, since you would only have to change the "low" half of your coordinates. However, you cannot simply send the bytes that change. The following table shows you what bytes must be sent under any set of conditions:

If High y changes: High y, Low x
If Low y changes: Low y, Low x
If High x changes: Low y, High x, and Low x
If Low x changes: Low x

These shortened coordinates can be combined; for example, if High y and Low y both change, but not x, you can send High y, Low y and Low x. Note that you must *always* send the Low x part of the coordinate, no matter what else changes. In addition, note that if High x changes, you have to send more than you might expect; the language also requires that you send the Low y byte. This prevents any ambiguity about the High x byte, which could also represent High y.

Unimplemented commands (12.3.3)

There are two groups of Tektronics 4014 commands that we have not implemented. The first group includes all the commands that do not "make sense" on our system; these are the "write-through" commands, the commands for the GIN mode and, in general, any commands that use the Tektronix 4014 interactively.

The second group of unimplemented commands are possible on the Imprint-10, but have not been included in the present version of the Tektronix language. Future versions may include them. These features are listed below, in the order in which they are likely to appear:

- "defocused mode" commands: at present, all lines are only one pixel wide.
- the APL character set: at present, we provide only the ACII character set.
- "enhanced graphics" commands: currently, we provide only $1/120^{th}$ inch resolution in Tektronix, not the higher resolution (4096 x 4096 point) page possible with the "enhanced graphics" option. However, Tektronix enhanced graphics is "downward compatible," i.e. high resolution graphics can be displayed on a terminal without the enhanced graphics package. Similarly, high resolution graphics can be printed on the IMPRINT-10 without problems, even though we do not provide their 4096 x 4096 point screen.
- point, incremental, and special point commands: Ignored in the current version. Since they are ignored, later commands will behave correctly.
- vector style commands: Again, ignored; currently, we only use the unbroken style.

Document Control (12.4)

There are two document control items specific to the Tektronics language. They both allow the emulator to simulate features which you can select by making wiring changes on the 4014 terminal. They are

`ls-is-crlf`
: If this document control variable is *on*, a linefeed will automatically produce a carriage-return line-feed combinatin. If it is *off*, a line-feed will simply be a line-feed. The default is `cr-is-crlf off`; to turn this feature on, you must specify `cr-is-crlf on` in your document header.

`cr-is-crlf`
: If this document control variable is *on*, a carriage return will produce a carriage-return line-feed combination. If it is *off*, a carriage return will be a carriage return by itself. As with the previous item, the default if `cr-is-crlf off`.

Error messages (12.5)

At this time, there is only one error message in Tektronix. It is

`Vector off page [`*x y*`]`

and it means that you are trying to print a vector with parts that lie outside the boundary of the page. *x* and *y* are the coordinates of the point that is causing the problem. All possible *x* coordinates will be on the page; but the same is not true for *y*, which has a maximum value of 780.

Control codes (12.6)

This table shows the control codes Tektronix uses, their keyboard equivalents, their decimal value, and their octal value.

Code	Name	Keyboard equivalent	Decimal value	Octal value
BEL	Bell	Ctrl g	7	7
BS	Backspace	Backspace or Ctrl h	8	10
HT	Tab	Ctrl i	9	11
VT		Ctrl k	11	13
FF	Formfeed	Ctrl l	12	14
GS	Graph	Ctrl M	29	35
US	Alpha	Ctrl O	31	37

Note: When you use the table above, be careful to distinguish between upper and lower case control characters.