NAPLPS: A New Standard for Text and Graphics

Part 1: Introduction, History, and Structure

A close look at an important and controversial new communications standard.

Jim Fleming
Unir Corporation
Suite 106
5987 East 71st St.
Indianapolis, IN 46220

William Frezza
Jerrold Division
General Instrument Corporation
2200 Byberry Rd.
Hatboro, PA 19040

Personal computers have a great deal in common. Several of them use the same microprocessor. Most have the same language in read-only memory (BASIC). And all use more or less the same keyboard. But there is a tremendous variation in the ways various computers handle graphics.

In order to mass-produce graphics software or to mass-distribute graphics information (as in videotex and teletext), a standard for graphics information is needed.

The North American Presentation-Level-Protocol Syntax (NAPLPS, or "nap-lips") is a method for encoding visual information in a standard and compact manner, which can then be exchanged among people using a variety of different computer systems. Like the well-established American Standard Code for Information Interchange (ASCII), NAPLPS is a set of rules and conventions describing how data bytes of information should be formatted, as well as a set of guidelines describing what should be displayed when properly formatted data bytes are received by a terminal.

Unlike ASCII, however, the major emphasis in NAPLPS is on the communication of information in a two-dimensional graphics format. Graphics and textual information can be represented in a variety of modes, colors, and styles. Facilities are also provided that allow a terminal user to interact with the two-dimensional visual display in an extremely free-form manner.

NAPLPS also includes a method for minimizing the amount of information that must be sent over communications lines. Techniques are provided that allow extensions to be added to NAPLPS at some future time without affecting existing features.

The basic concept of NAPLPS can be illustrated by the cartoon in figure 1 on page 204. It shows a robot artist being fed a stream of commands that are used to paint a picture. At the robot's disposal are pens of various colors, spray paints, character templates, and all the other items found in an art studio.

With various commands, we can direct the robot's arm to any area of the canvas we desire. We can instruct the robot to use any of several standard colors, or we can tell it to create a new color from the existing ones. When text is needed, the robot selects the proper-size template for the desired letters, grabs a can of spray paint, places the template on the canvas, and paints a character.

The goal of this system is that the beauty and complexity of a picture should be limited only by the imagination and skillfulness of the person (or program) creating the commands being fed to the robot.

About the Authors

Jim Fleming and William Frezza are members of the ANSI X3L2 Committee on Character Sets and Coding. Mr. Fleming is also working on Chemical Bank's Pronto home-banking project.

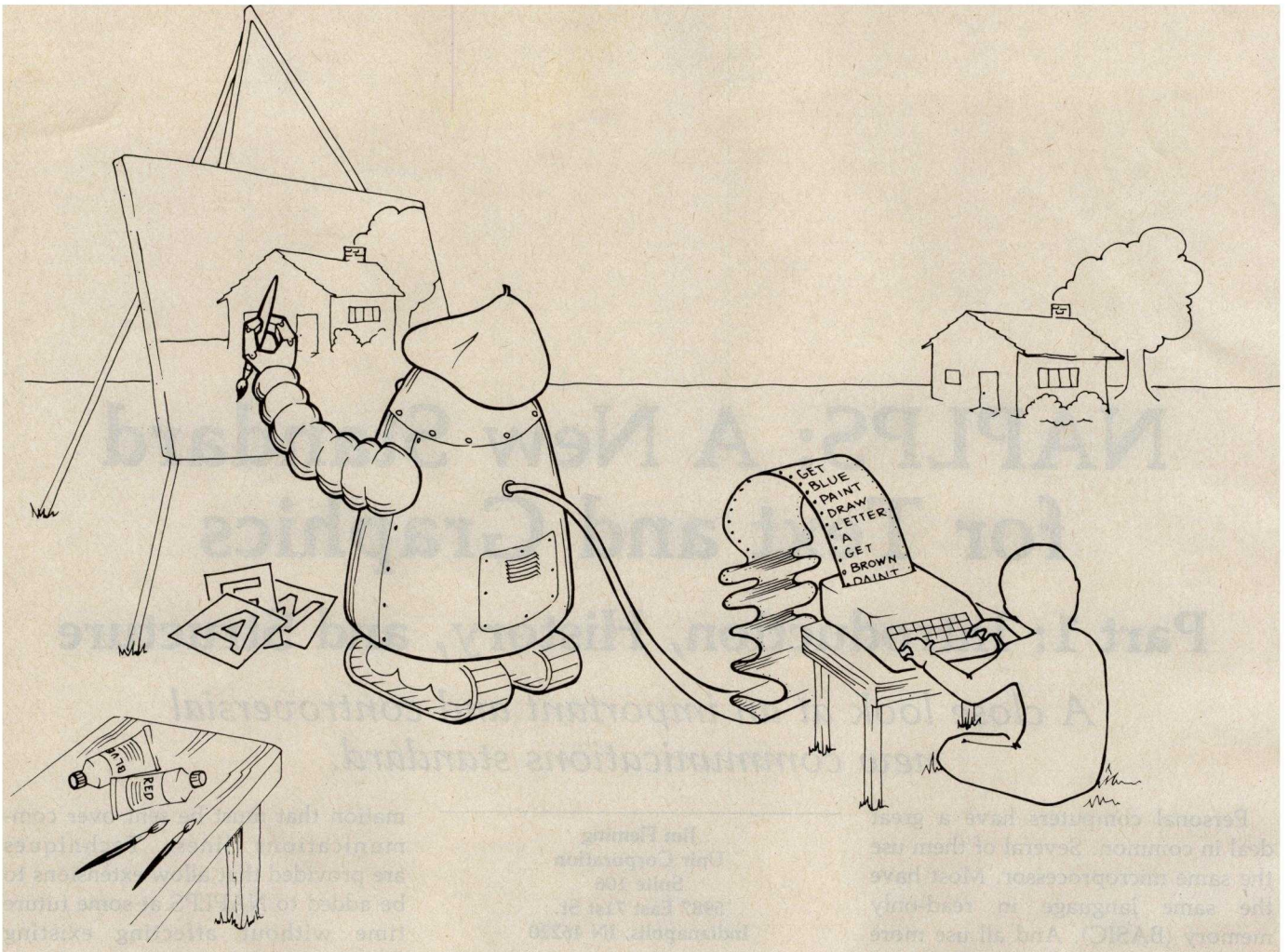


Figure 1: *A stylized representation of how the NAPLPS system works. The programmer or artist creates a list of graphics commands, e.g., "get red pen," "draw a circle." The robot (or NAPLPS decoder) then interprets these commands and uses various drawing instruments such as pens, brushes, rulers, and compasses to draw on the canvas (display screen). If a text character is specified, the robot uses an appropriate template for that character.*

This article is the first in a series of articles on NAPLPS. In this part, we give an overall perspective of NAPLPS, describing its history and background, as well as its structure and major features. In subsequent parts, we will cover the basic text and graphics features of NAPLPS from a bit and byte perspective, describe some of its more advanced features, and explore the future of NAPLPS with an emphasis on personal computers, local and regional area networks, and distributed processing.

History and Background

NAPLPS has its roots in videotex, a much-discussed system of large host computers and low-cost, user-friendly graphics terminals. Because of the large potential market for these terminals, many groups around the world have been designing such systems for use in homes, offices, and public areas. As shown in figure 2 on page 206, a basic videotex system consists of a host computer with a database of information, a communication network, and a terminal. The terminal users request information from the database, and the desired information is sent back to the terminal, where it is interpreted and displayed.

Unfortunately, all the experimental systems designed around the world used different coding schemes. As is the case with most languages, the various coding schemes had different strengths and weaknesses. Some were more efficient than others; some were more easily decoded by terminals; some preserved the "conceptual" content of the information; and some were tailored to particular hardware configurations.

At the time NAPLPS was developed, videotex coding schemes could be divided into two major groups. In one group were schemes that were similar to the approach used in the British Prestel system, which was the first videotex effort in the world. The other group of schemes is best represented by the Telidon system developed in Canada as an alternative to the Prestel system. As is the case with many developments in the computer field, being first does not imply being the best.

Table 1 on page 210 compares Prestel-like systems and Telidon-like systems. Without going into all the

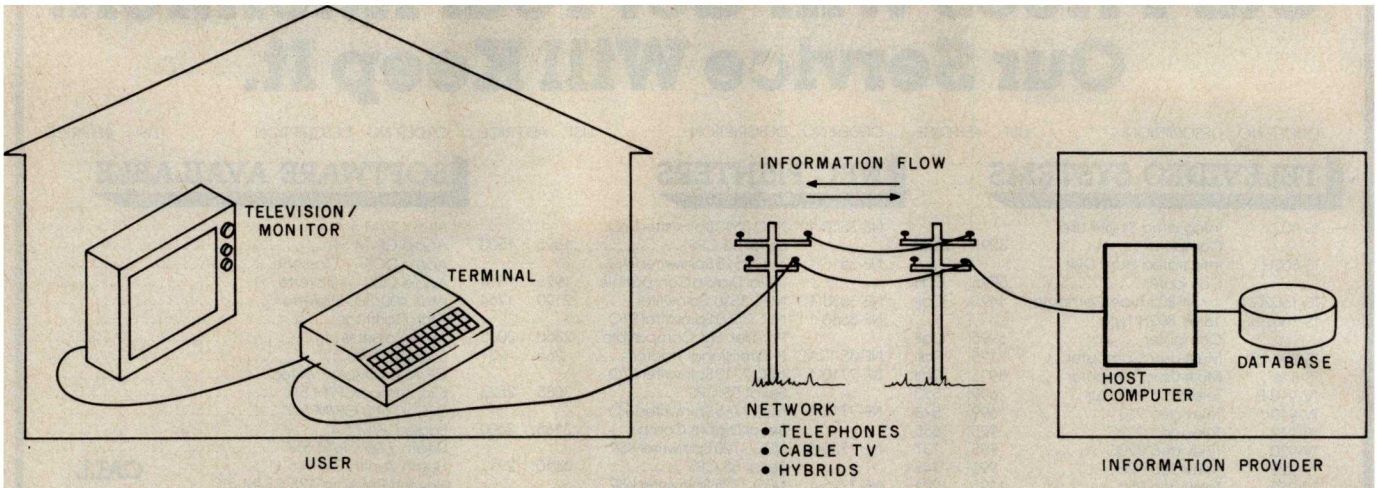


Figure 2: *A diagram of a typical videotex system. Videotex is defined here as two-way communication of textual and graphical information between a low-cost, user-friendly terminal and a large, central host computer. Communication can be by telephone lines, television cables, or a hybrid system using a broadcast television channel for information sent from the host computer and using telephone lines for information sent from the terminal.*

technical, emotional, and political history, suffice it to say that NAPLPS was designed using Telidon-like systems as a base.

In May 1981, AT&T created a bit of commotion by releasing documentation for a new Telidon-like scheme called PLP (Presentation-Level Protocol) at the Videotex '81 conference in Toronto. Since that time, continuous efforts have been underway in various standards groups to adopt PLP.

NAPLPS is a standard version of PLP that resulted from a joint effort by the American National Standards Institute (ANSI) and the Canadian Standards Association (CSA). Copies of the draft proposed NAPLPS standard (document #BSRX3.110-198X) can be obtained from CBEMA (Computer and Business Equipment Manufacturers Association, X3 Secretariat, Suite 500, 311 First St., NW, Washington, DC 20001).

This series of articles will provide an overview of the features of NAPLPS. The specific details and examples presented in these articles are not meant to form a complete NAPLPS specification. Anyone interested in doing development work using NAPLPS should obtain a copy of the ANSI document.

Layered Protocols

Modern communication systems are designed in a *layered* or *modular* manner to help prevent extensive system redesign when parts of a system are changed. Layering achieves many of the advantages found in good structured system design. By isolating functions in various layers, we can proceed to standardize and imple-

Text continued on page 210

Characteristic	Prestel-like Systems	Telidon-like Systems	Comment
Video-display hardware dependence	Very much	Very little	This is the main advantage of NAPLPS. It is not based on special circuits or architectures.
Image complexity	Poor	Excellent	There is no comparison. It would be like trying to compare 8-mm home movies to 35-mm theater films.
Easily decoded by terminals	Yes	No	Prestel wins this one. Unfortunately, most things in life that are easy are not worth much.
Requires microprocessor terminal	No	Yes	Many thought this was an advantage and an objective worth achieving. Maybe they don't know how to program microprocessors.
Works with printers, plotters, etc.	No	Yes	While some were asking "Why?," others were saying "Why not?"
Memory intensive	No	Yes	Prestel wins again. Now that 16K bits are cheaper than 4K, this hardly seems a victory.
Preserves "conceptual" content information	No	Yes	Most are still trying to figure out what this means and why it is useful.
Can be extended for years	No	Yes	This certainly can be disputed. Time will be the judge.
Sensitive to errors in the communication channel	Less	More	A valid point but hardly an issue for a level-6 protocol.
Cost	Low	????	The true bottom line in some people's books. But how much did a personal computer cost 10 years ago?

Table 1: *A comparison of two types of graphics encoding systems for use in videotex applications: Prestel-like systems and Telidon-like systems. NAPLPS is one of the latter.*

ment a system for one layer without regard to details of other layers. Because layering is an abstract and sometimes confusing topic, we will use a simple example of communication between two people to illustrate the concept.

As shown in figure 3 on page 212, when two people converse, their basic goal is to communicate ideas to each other with as much understanding as possible. We shall regard these ideas themselves as the first level or layer of communication. This level, which may be considered the highest or most abstract, will be called the *conceptual* level.

In order for people to communicate these ideas, they must choose a language—say, English—as a set of rules for *presenting* the ideas. And with English come all the rules concerning grammar, sentence structure, and so on. We shall include English as part of a second level of communication that we shall call the *logical* level. The ideas from the upper level would have to be expressed in this logical level before a transfer could take place between the two people.

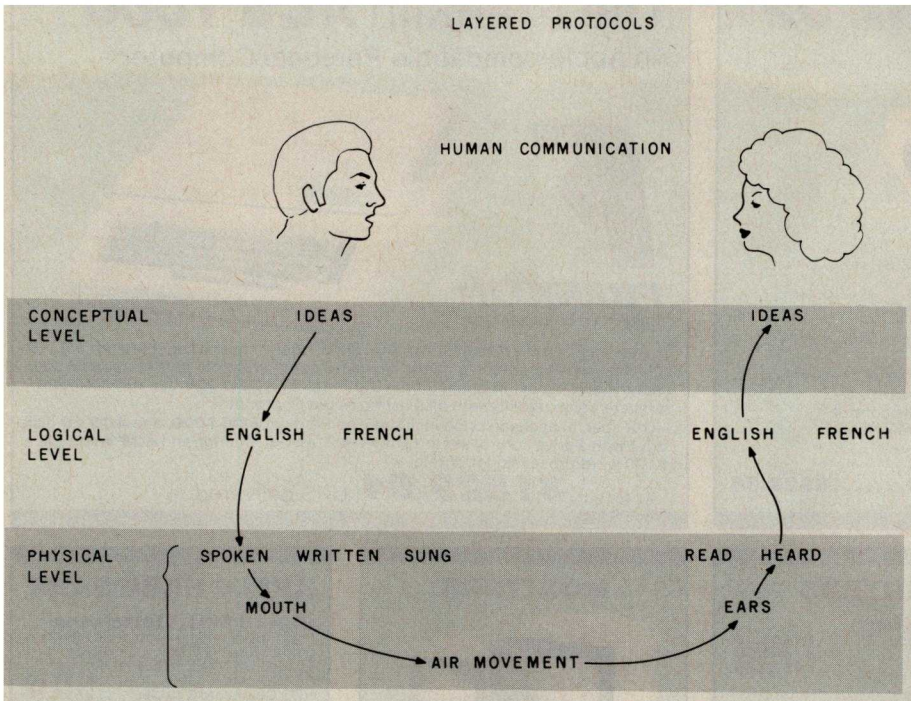


Figure 3: *A diagram showing how communication can be divided into a series of layered protocols. Here, the example of communication is a simple conversation in English between two people. The conceptual level comprises the actual ideas to be communicated. The logical level comprises the language in which the ideas are to be expressed. The physical level comprises the physical phenomena that are used to convey the English words. In the case of speech, this involves movements of the mouth, air, and the listener's eardrums.*

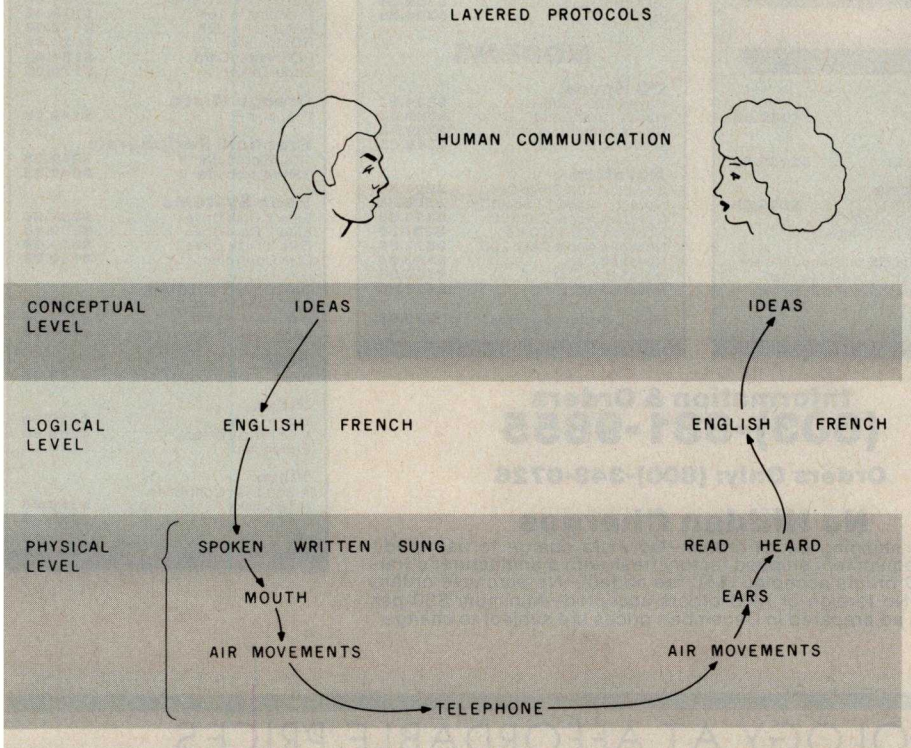


Figure 4: *If the conversation in figure 3 were conducted over a telephone, we could interpret this as a change of the physical layer. The advantage of layered protocols is that one layer can be changed without affecting the other layers. Although the physical level here has been changed, the logical level—English—is unaffected.*

Once English is chosen, a mechanism is needed to physically transfer the logical representations of the conceptual ideas from one person to another. This will be done on the *physical* level. In human communication, several choices exist. The most obvious is speech. When we speak, a set of physical tools is used. The English constructs from the logical level are converted to movements of the diaphragm, tongue, and mouth, which result in the movement of air. The vibrating air is detected by the other person's ears (if she is listening) and is transferred into bone and muscle movements. The second person must decode these movements, recreate the English, and conceptualize the idea.

This example can also be used to illustrate why layering is useful in preventing complete system redesign when changes are made. It can even be used to show how standard layers can be mixed and matched as the needs of a system change.

Suppose that the two people are separated by a large distance and that a telephone must be used so that they can talk to one another. The lowest level (the physical level) is the only area affected. As shown in figure 4, the telephone and the telephone network are used to transport the sounds from one location to another. The logical English constructs can remain the same and the ideas can be communicated.

If French or German is substituted at the logic level, no changes need to be made to the physical level. The conceptual level may or may not be affected, depending on how adept the languages are in representing certain ideas. For example, when learning a second language, one usually runs into the case where an instructor says, "That idea really can't be translated into this language."

As mentioned before, layering is done to prevent expensive system redesign when parts of a complex communication system are changed. Imagine how inconvenient it would have been if everyone had had to learn a new language when the telephone was invented. Or imagine how expensive it would be if a dif-

ferent telephone system were needed to speak different foreign languages.

Data-communication systems have likewise been divided into various layers. A seven-level model promoted by the International Organization for Standardization (ISO) is typically used. A complete description of the model is beyond the scope of this article. In general terms, however, this seven-layer model, like our simple example, runs from the more abstract layers at the top (level 7) to the physical layers at the bottom (level 1). Most of the work in standardizing data-communication protocols has heretofore been done at the lower, physical levels.

NAPLPS is a standard for the sixth level, commonly called the *presentation level*, of the seven-level model. In our example of human communication, NAPLPS is similar to the logical (English, French, and German) level. NAPLPS has been designed to allow a large variety of information to be encoded in a manner that preserves the conceptual content of the information. NAPLPS codes can be physically transported between computer systems via modems and data links, floppy disks, magnetic tapes, and other common mechanisms.

Code-Extension Techniques

The coding of NAPLPS begins with bits and bytes. The 8-bit byte can be used to represent 256 unique patterns or code points. At first glance, the 256 codes might seem to be a large enough set, especially if only letters, digits, and control information must be encoded. But in order to encode graphics coordinates, colors, graphics drawing commands, and advanced control information, more than 256 codes are needed. The obvious solution is to group bytes together sequentially to form an extremely large set of commands. This is similar to what occurs in English where the 26 letters of the alphabet are grouped to form words.

In NAPLPS, 96-code sets can be swapped in and out of a large 256-code table.

Grouping of bytes is commonly called *code extension*. Many code-extension techniques use the ASCII Escape character (ESC, hexadecimal 1B, decimal 27) as an indicator that the next character has a special meaning. Many times, the next character indicates that more characters follow. (An example of this type of code extension is the typical multicharacter Escape sequence for the cursor-positioning sequence supported by many terminals.)

This approach to code extension is fine for a small number of extensions, but tends to become a hodgepodge of inconsistent code sequences when a large number of extensions are defined.

NAPLPS has been designed with an extremely general code-extension structure that is independent of the specific "meanings" of the codes, and is based on an ISO recommendation (ISO 2022.2).

Keep in mind that up to this point we have been talking about codes as 8-bit binary numbers in the decimal range 0 to 255. No meaning has been placed on the codes. Because of the widespread use of ASCII, many people assume that a capital "C" must *always* be coded as a decimal 67, as it is in ASCII. The assumption is also made that the value 67 cannot be used to code anything but capital Cs. In order to fully understand NAPLPS, you must first realize that the relationship that exists between the capital C and 67 is by *convention* and not due to some physical limitation of computers or an act of God. Furthermore, you must realize that the decimal value 67 (or any code) can be given other meanings in other contexts as long as an indication is given as to which context is currently in effect.

The basic strategy underlying code extension in NAPLPS is to take a large table of codes (128 or 256) and divide it into smaller sets of codes that can be "swapped" in and out of the large table. The small code sets

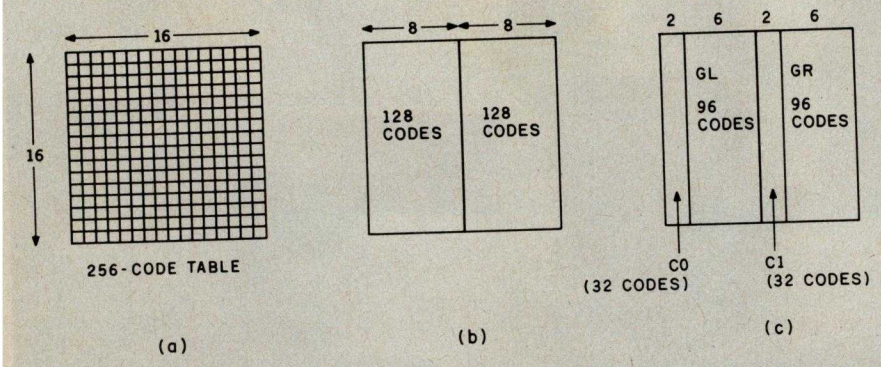


Figure 5: *With an 8-bit code, 256 combinations are possible. These can be represented on a 16 by 16 table (a). For convenience, this large table can be divided into two 128-code tables (b). Each of these 128-code tables can then be further subdivided into a 32-code table and a 96-code table (c).*

can include codes with similar characteristics. The sets can have standard names, and a standard mechanism can be established to control the swapping. New sets can be added as long as a unique name is chosen. Because a standard mechanism would already be in place to handle the swapping, the new code set could be added without affecting other sets.

Up to now, we have been talking mainly about an 8-bit code. Actually, two code-extension techniques are supported in NAPLPS: 7-bit and 8-bit. The 7-bit extension technique is used in systems where only 7 data bits can be passed through the lower, physical levels of communication (levels 1 through 5). The eighth bit is often reserved for parity so that errors can be detected. In a seven-level system, error control is usually performed at level 2. Because NAPLPS is a level-6 protocol, the error-control bits have already been handled prior to the data's reaching level 6.

The 8-bit code-extension technique is used when all 8 data bits are available for NAPLPS information. This is the method that is used in systems where the low-level protocols can support 8 bits. It will also be used when files containing NAPLPS are exchanged between users via disks and tapes. Because of the eventual widespread use of the 8-bit code-extension technique, it is the one that will be described in this article.

With 8 data bits, the 256 codes or patterns can be grouped in the form of a table with 16 rows and 16 columns ($16 \times 16 = 256$), as shown in figure 5a.

The 16 by 16 table can be divided into two sets of 128 codes, as shown in figure 5b. These two sets can each be partitioned into sets of 32 and 96 codes ($32 + 96 = 128$), as shown in figure 5c. The 32 codes will occupy two columns of the original 16 by 16

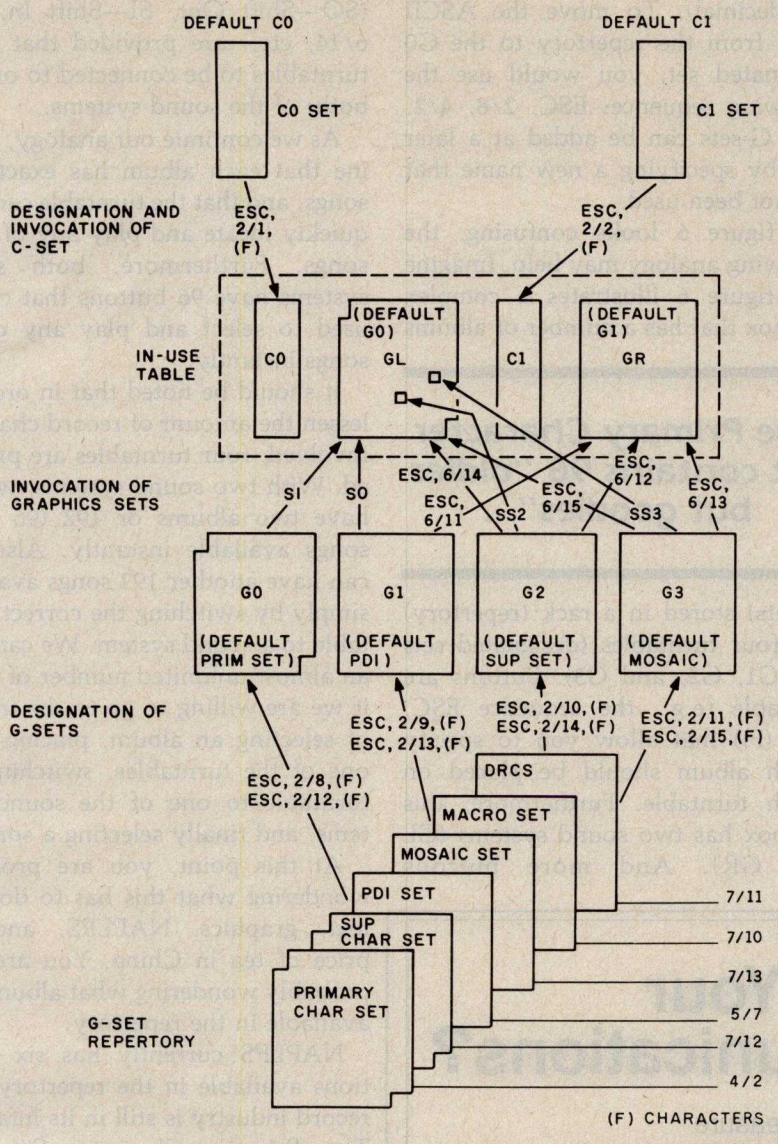


Figure 6: *A diagram showing the NAPLPS code-extension technique in an 8-bit environment. By swapping various 96-character graphics sets into and out of the graphics areas of the "in use" table, we can access a large number of characters or commands. Four graphics sets (or G-sets) are selected from the G-set repertory and placed in designated sets (G0 through G3). Then, two of these designated sets are placed in the graphics areas (GR and GL) of the 256-code "in use" table. Various code sequences (e.g., ESC 6/14) or control codes (e.g., SI) are used to swap the G-sets. The notation "6/14" represents the number 6E in hexadecimal. "(F)" refers to a single-code name of a particular G-set.*

table; the 96-code set will require six columns.

As you can see, the large 256-character table has now been divided into four smaller regions. These regions (or sets) allow us to group codes of similar use into tables of manageable size. The two small tables are called *control sets* or C-sets; the two large tables, *graphics sets* or G-sets.

As we mentioned before, a mechanism has been designed to allow a variety of code sets to be swapped into and out of these four areas of the large table. Currently, however, code-set swapping is done only with the large 96-character G-sets. Although a mechanism exists for swapping the small areas (C-sets), it is not being used at this time.

Before a G-set is swapped into one of the large areas, it must be selected from a repertory and placed into one of four designated sets. Two of these

designated sets are then placed into GL and GR, the two large areas in figure 5c. Codes are then interpreted based on the current G-sets that are in use in the large table.

Figure 6 illustrates this mechanism for the 8-bit code-extension technique. The arrows and labels indicate special code sequences that are used to cause the swapping. Most of these code sequences begin with the Escape character. The notation "6/14" used in figure 6 is an alternate way of specifying a code with a specific bit pattern. On a 16 by 16 table, 6/14 represents the bit pattern that refers to column 6 and row 14 of the table. In hexadecimal, 6/14 would be 6E; in decimal, (6 × 16) + 14 = 110.

To move a G-set from the repertory to one of the designated sets, a three-character sequence is used. The third character in the sequence (represented by "(F)" in figure 6) is the "name" of the G-set. Each G-set has a unique name that is specified in the NAPLPS standard. For example, the name of the ASCII G-set is 4/2 (42 in hexadecimal). To move the ASCII G-set from the repertory to the G0 designated set, you would use the following sequence: ESC, 2/8, 4/2. New G-sets can be added at a later date by specifying a new name that has not been used.

If figure 6 looks confusing, the following analogy may help. Imagine that figure 6 illustrates a complex jukebox that has a number of albums (G-sets) stored in a rack (repertory) and four turntables (designated sets G0, G1, G2, and G3). Buttons are available (e.g., the sequence ESC, 2/8, (F)) that allow you to specify which album should be placed on which turntable. Furthermore, this jukebox has two sound systems (GL and GR). And more buttons (SO—Shift Out, SI—Shift In, ESC 6/14, etc.) are provided that allow turntables to be connected to one (or both) of the sound systems.

The Primary Character Set contains 96 "oldies but goodies"...

As we continue our analogy, imagine that each album has exactly 96 songs, and that the turntable can very quickly locate and play any of these songs. Furthermore, both sound systems have 96 buttons that can be used to select and play any of the songs instantly.

It should be noted that in order to lessen the amount of record changing involved, four turntables are provided. With two sound systems, we can have two albums or 192 (96 × 2) songs available instantly. Also, we can have another 192 songs available simply by switching the correct turntable to a sound system. We can play an almost unlimited number of songs if we are willing to go to the trouble of selecting an album, placing it on one of the turntables, switching the turntable to one of the sound systems, and finally selecting a song.

At this point, you are probably wondering what this has to do with text, graphics, NAPLPS, and the price of tea in China. You are also probably wondering what albums are available in the repertory.

NAPLPS currently has six selections available in the repertory (this record industry is still in its infancy). The *Primary Character Set*, also known as ASCII, is full of 96 oldies but goodies like 0, 1, 2, . . . A, B, C, and x, y, z, etc. The *Supplementary Character Set* is full of 96 new and old international favorites, most of which are rarely played in the U.S. These include α and β. The *Picture-Description Instructions* album (PDIs) contains selections like "Line," "Arc," and "Draw Me a Polygon." Some of the hottest hits going are on this album. The *Mosaics* album is full of some very old songs that all sound the same. It is seldom played except by people over 40. The *Macro* album contains songs that cause other songs to be played. (You get a lot for your quarter here.) The *Dynamically Redefinable Character Set* album (DRCS) is initially blank. It can be used to mix existing songs together to form new songs. (Yes, on this juke-

Text continued on page 224

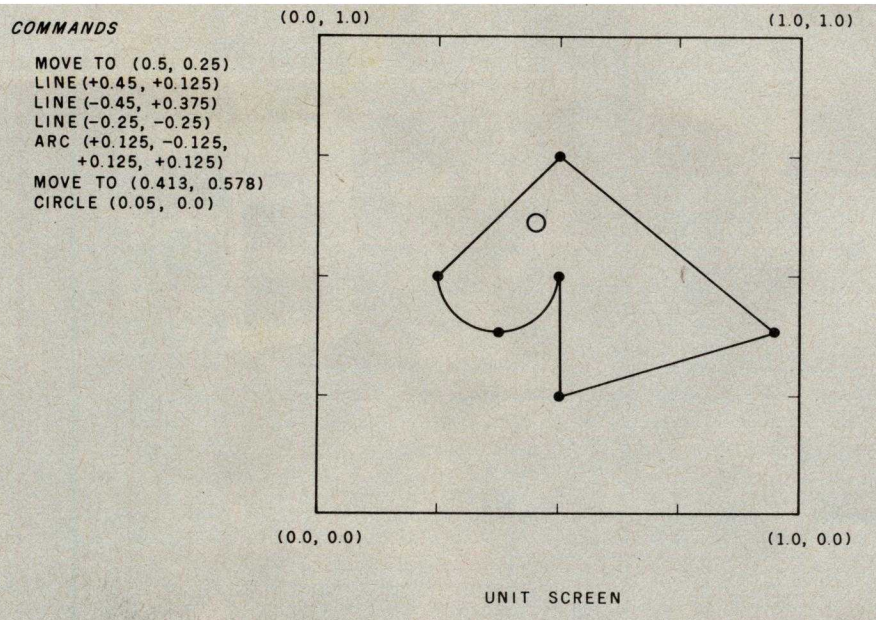


Figure 7: *The unit screen of NAPLPS. All coordinates are represented as fractions between 0.0 and 1.0. The figure on the screen was drawn with the commands listed on the left. The advantage of this coordinate scheme is that it can be easily implemented on display screens of various resolutions and sizes.*

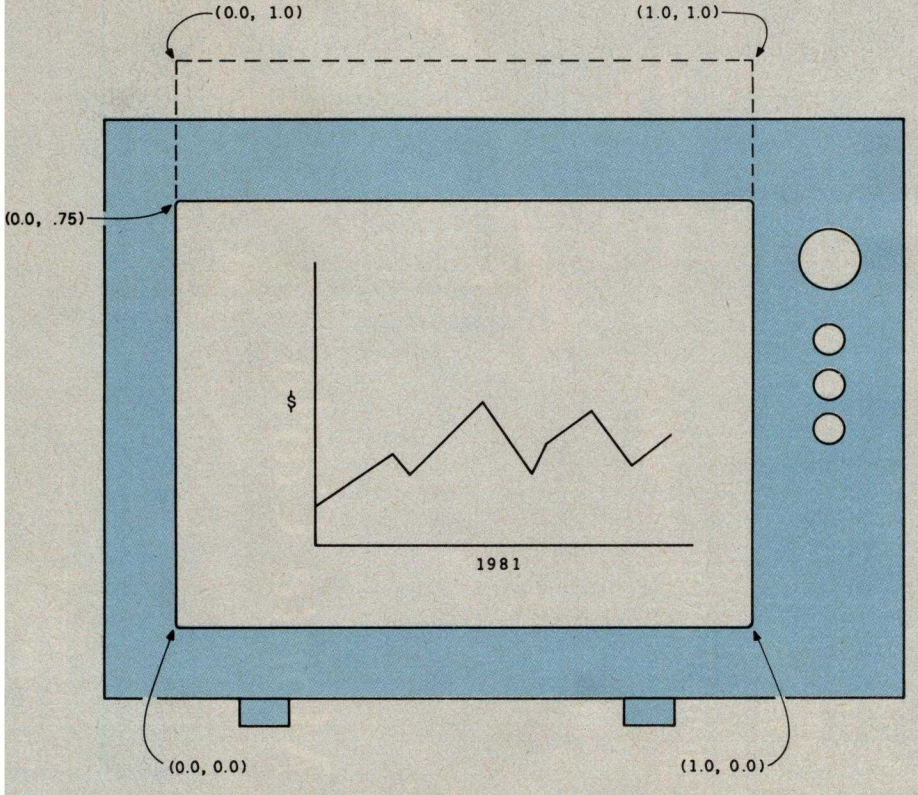


Figure 8: *The unit screen is square, but most display screens are rectangular. The convention that has been adopted is to represent on the display screen only the lower 75 percent of the unit screen. That is, any point with a y coordinate greater than 0.75 will not be seen.*

box you can record as well as play.)

As mentioned before, default selections have been set up so that no swapping commands are needed in many applications. As shown in figure 6, the ASCII character set is the default for the G0 designated set, and whatever is in G0 is the default for GL. Therefore, the codes in GL (decimal 32 to 127) will be mapped to the ASCII character set as the default condition. (Isn't it amazing how the simplicity of the present can be represented as a subset of the complexity of the future?)

The default for the G1 designated set is the PDI set, and G1 in turn is the default for GR. This arrangement allows text and graphics to be used without any swapping.

The default for G2 is the Supplementary Graphics Set, and the default for G3 is the Mosaic Set. We believe that the Macros and DRCS should have been the defaults. When you devise a standard, however, sometimes a little "default diplomacy" is necessary.

The entire NAPLPS code-extension structure is designed to support future growth in an organized manner. As can be seen, it provides a means of increasing the number of codes far beyond the 256 codes we would have had if there were no code-extension techniques. The overhead has been kept to a minimum while maintaining compatibility with existing ASCII systems.

The Unit Screen and Coordinate System

Now that we have plenty of room for character sets and commands, we can get down to the real purpose of NAPLPS—creating pictures.

In NAPLPS, pictures are drawn on a *unit screen*. As shown in figure 7, the unit screen is a square area of unknown resolution and size. The lower left corner of the screen has x-y coordinates equal to (0.0, 0.0); the upper right-hand corner of the screen has x-y coordinates of (1.0, 1.0).

The name "unit screen" is derived from the fact that all coordinates in the unit screen have an x and y component between 0.0 and 1.0. In NAPLPS, all coordinates and dis-

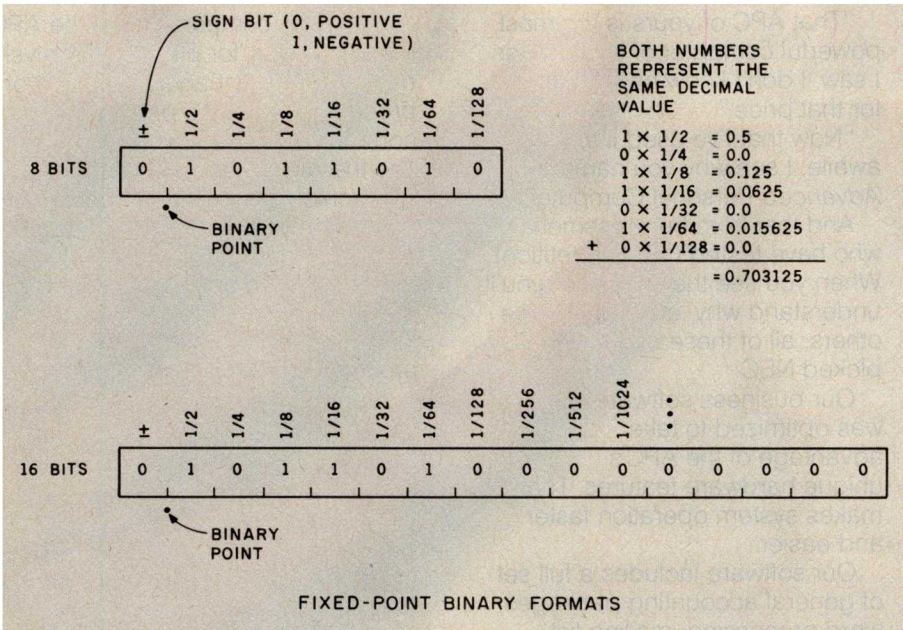


Figure 9: *NAPLPS coordinates are formatted as "fixed-point binary numbers." The 8- and 16-bit numbers given here represent the same decimal number, 0.703125.*

tances are specified thus in subunits relative to the unit screen. The advantage of specifying the coordinates in this manner is that the pictures will be independent of any particular hardware configuration. Another advantage is that objects in pictures will remain in the same relative position with respect to each other even though the resolution of the physical display may be increased.

In order that pictures may be seen, the unit coordinates must be *mapped* to a physical display. The only requirement imposed (under normal conditions) when making this mapping is that the *squareness* (commonly called *aspect ratio*) of the unit screen should be preserved. Unfortunately, when the unit screen is mapped to the rectangular screen of a television set, some of the unit screen cannot be seen. This is shown in figure 8. The convention that has been adopted is that only the lower 75 percent of the unit screen will be visible on the physical screen. Thus, any point with a y coordinate greater than 0.75 (it is usually closer to 0.78) will not be displayed on a television screen.

This technique of mapping points on the unit screen to the physical screen is called one-to-one mapping. In the future, additional mapping techniques may be added to NAPLPS that will allow the unit screen to be scaled, rotated, and mapped to the physical screen in a variety of ways. These capabilities will be added at the same time that three-dimensional features are defined.

Now that we know that all coordinates must be between 0.0 and 1.0, a problem arises: How do we represent these coordinates? Floating-point representations could be used. But this would make it difficult for integer-oriented microprocessors to handle the coordinates. Instead of a floating-point format, a *fixed-point binary* (not binary-coded decimal or BCD) format was chosen. This format is the same as a typical integer format, except the *binary point* is assumed to be on the left between the sign bit and the data bits. Figure 9 illustrates the formats for 8- and 16-bit systems.

The important thing to note about this format is that, unlike integers, as more bits of precision are added, they are added on the *right* instead of the left. Also, the values of the binary places work from the left to the right. The value of the bit position immediately to the right of the binary point is 1/2. The next bit position to the right is worth 1/4. The next ones are worth 1/8, 1/16, 1/32, etc.

The decimal value of a number is determined in a manner similar to integers. A number such as 0.1011010000000 represents a positive number (the sign bit of 0) equal to 1/2 + 1/8 + 1/16 + 1/64 or 0.703125, which of course is less than 1.0. An infinite number of zeros is assumed on the right of the number, just as with decimal numbers that are less than 1. Of course, the number will never equal 1.0 no matter how many 1s are placed on the right. (If you do not believe it, try figuring out what the fixed-point binary number 0.111111111111111 is in decimal.)

When coordinates are encoded in NAPLPS, each byte can contain 6 bits of data. (The other 2 bits will be accounted for later.) The standard two-dimensional format is shown on the left of figure 10 (page 227). On the right side of figure 10 is a three-dimensional format. Some three-dimensional capability is supported by NAPLPS today, but many more three-dimensional options will be available in the future. In that case, coordinates are specified in a unit cube rather than a unit screen.

In the two-dimensional format, the 6 data bits are used for 3 bits of x and 3 bits of y. Obviously, multiple bytes are needed if high-precision coordinates are used. As shown in figure 11, as each new byte is added to a coordinate specification, the x and y components each obtain 3 more bits of precision. The least significant bits are obtained after the most significant bits. A terminal may choose to throw away some of the least significant bits if more bits are sent than are needed for the resolution of that particular terminal.

When most people are first exposed to this method of coordinate encoding, their first reaction is that it will be too complex for a simple microprocessor to handle. On the contrary, there is a very easy way to handle this encoding technique: just ignore the binary point and the fractional concepts and treat the bits as integers.

To do this, you must first choose an adequate integer size for internal representations. On 16-bit microprocessors, 16 bits are commonly used. If signed 16-bit numbers are used, a grid

can be set up that ranges from −32,768 to 32,767 in both the x and y directions (see figure 12). The display screen or unit screen would occupy the first quadrant. The unit screen would then be 32,768 by 32,768, which is far more resolution than almost all graphics devices have today.

In this 16-bit internal form, an integer such as 0100000000000000 would have a decimal value of 16,384. This is equal to ½ of 32,768, which should not be surprising because we originally said that the binary number 0.100000000000000 was equal to ½ (it's all done with mirrors!). The integer 0101101000000000 that we used before would of course be equal to 23,040 (16,384 + 4096 + 2048 + 512). A quick check with a calculator shows that 23,040/32,768 is exactly 0.703125. (Does this number look familiar?)

It should be clear that treating the fixed-point binary numbers as normal integers is the same as moving the binary point 15 places to the right (for a 16-bit system), which is the same as multiplying the binary fractions by 32,768. We can recover the fractional form by dividing by 32,768, which was demonstrated above.

In order to map the unit screen to a physical display screen, more simple shifting can be used. The sign bits of the x and y components must be positive for the coordinate to be in the unit screen. If the rightmost 7 bits of the 16 bits above are dropped by shifting the integer right seven places, the numbers that result are in the range 0 to 255.

This operation maps the 32K- by 32K-bit grid to a 256 by 256 grid. Each point on the 256 by 256 grid then represents a 128 by 128 area on the original grid. This indicates that when 16-bit integers are used, 128 would have to be added to a coordinate component to move to a different point on the physical display.

If a 512- by 512-bit-resolution display screen is available, another bit on the right of the coordinate integer would be saved. (The 16-bit integer would be shifted right six places instead of seven.) In this case, each point on the 512 by 512 grid

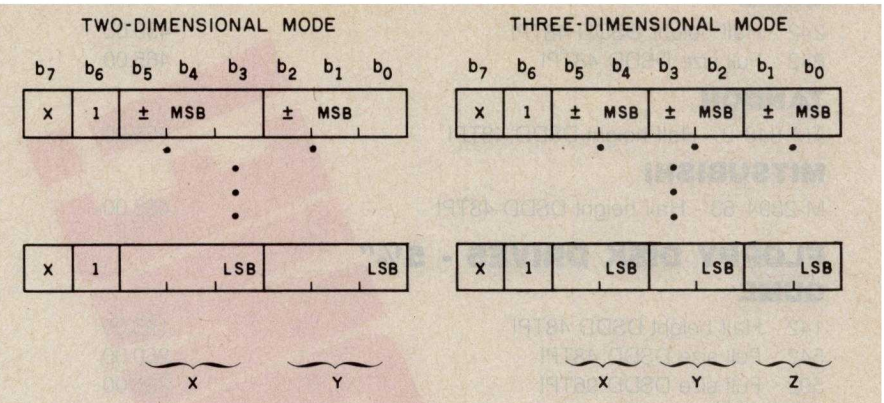


Figure 10: *In NAPLPS, coordinates are specified with a varying number of bytes. In the two-dimensional mode, each byte contains 3 bits of the x coordinate and 3 of the y. In the three-dimensional mode, each byte contains 2 bits each for the x, y, and z coordinates. MSB indicates the most significant bit; LSB, the least significant bit.*

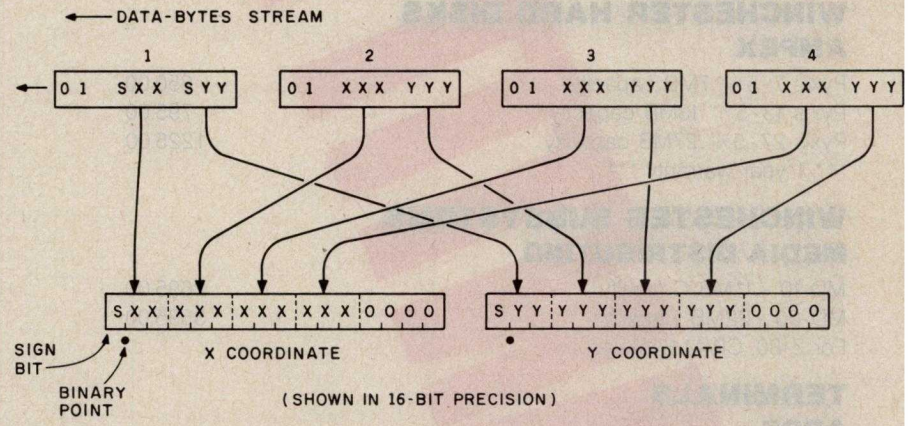


Figure 11: *The data bytes shown in figure 10 can be combined to specify coordinates of almost unlimited resolution. Here, 4 data bytes in the two-dimensional mode are combined to form a pair of 12-bit coordinates. This would support a resolution of 2048 by 2048.*

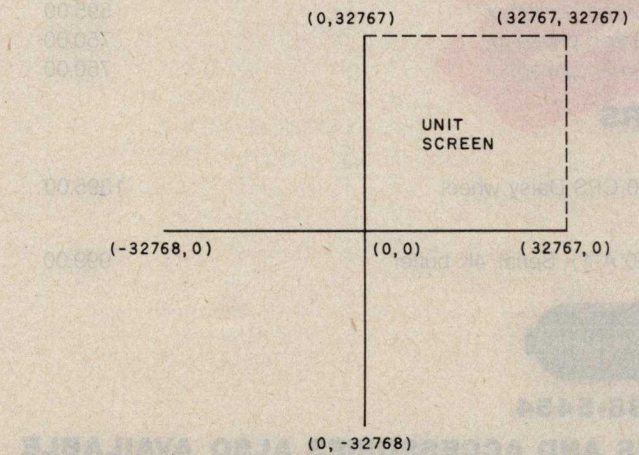


Figure 12: *The maximum resolution of a 16-bit coordinate system. The unit screen occupies only the first quadrant of the grid.*

represents a 64 by 64 area on the original grid. Adding 128 to an integer in this case would move the coordinates by *two* display points, not one. If this did not occur, pictures developed for a 256 by 256 grid would end up in the lower corner of a 512 by 512 display. That lack of portability would discourage increasing the resolution of the terminal. Fortunately, with NAPLPS we can increase the resolution of a display and still be able to receive pictures developed for older displays. They will look as good or better on the new display.

So far, we have discussed only positive coordinates and integers. Negative values can occur in the normal two's complement form used by most microprocessors. Negative values can be used to code relative coordinates (*dx* and *dy* values) when relative movements are needed, rather than absolute coordinates. The values *dx* and *dy* can also be used to indicate sizes of areas on the screen.

Part 2 of this series will describe how the *dx* and *dy* values are used to specify character sizes. We will also see that many of the graphics commands have an absolute form and a relative form. The absolute forms are used when the drawing must appear at a particular spot on the unit screen. Relative forms are useful when one wants to draw relative to the current drawing point, which may be in different places depending on the previous figure.

Color Control

NAPLPS supports a wide range of color control. Three color modes (0, 1, and 2) are available to satisfy many different applications. The first of these (color mode 0) is fairly simple and is designed to be compatible with almost all color display screens. The other two (color modes 1 and 2) use what is known as *color mapping*. This allows you to create some fantastic visual effects, but this technique requires special hardware not found in most color displays.

Color mode 0 is the most primitive mode in NAPLPS. It can best be described by the following analogy using the robot mentioned at the

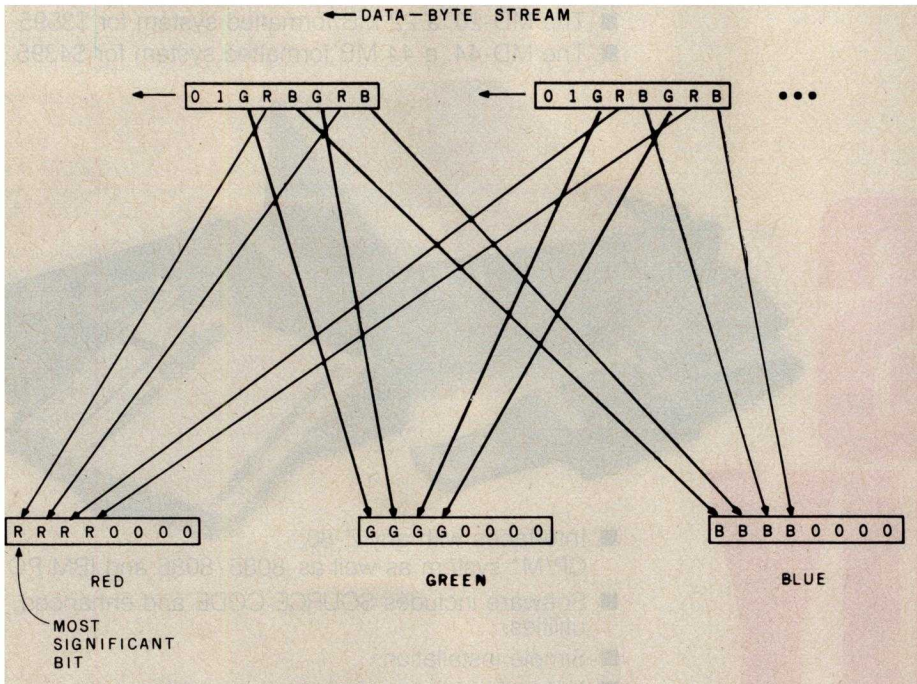


Figure 13: *Color information is encoded in a manner similar to that used for coordinates. Each data byte contains 2 bits of information for each of the primary color components: red, green, and blue. A varying number of bytes can be combined to specify colors with almost unlimited precision. Here, 2 data bytes have been combined to yield 4 bits of information on each red, green, and blue component of a color.*

beginning of the article. Imagine that the robot has one pen and three inkwells filled with the primary colors red, blue, and green. By mixing various amounts of each of these colors in the pen, the robot can draw in almost any color. For example, we could instruct the robot to mix three drops of red, one drop of blue, and seven drops of green, and then tell the robot to draw various shapes or text characters. When we tell the robot to mix a new color, the robot would automatically clean out the pen and mix the next color.

In NAPLPS, color is similarly specified in terms of its red, green, and blue intensities. Each byte of color data contains 6 bits of color information, 2 each for red, green, and blue. Several bytes, however, can be grouped together so that colors can be specified with as much precision as desired. In figure 13, 2 bytes have been used to yield a total of 12 bits of color information (i.e., 4096 possible colors). As with coordinate encoding,

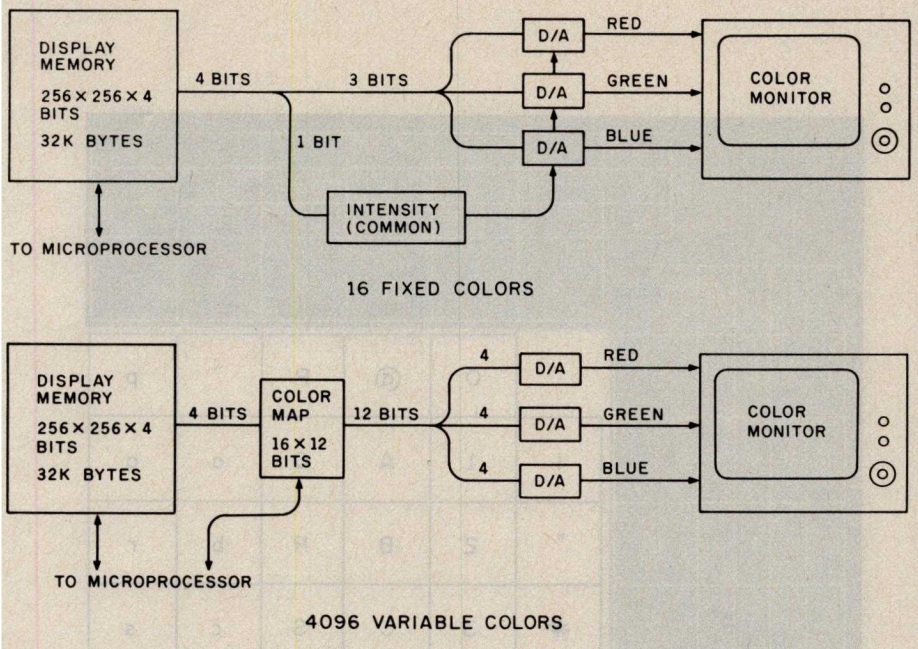


Figure 14: *Two popular schemes for storing color information. Both use the same amount of display memory. In the top scheme, the 4 bits for each pixel specify 16 fixed colors. In the lower scheme, the 4 bits specify 16 color registers in the color map. Each color register in turn specifies one of 4096 colors. D/A designates a digital-to-analog converter.*

the most significant bits are sent first, and a terminal is free to ignore the least significant bits.

With this kind of system, a tremendous spectrum of colors may be displayed, depending on the amount of memory available.

Most personal computers have only a small number of colors available. In the above analogy, the robot might have 8 or 16 pens with premixed colors. When we gave the robot instructions to mix a certain color, it would merely pick the pen with the color closest to the specified color.

The advantage of color mode 0 is that it can be received on almost all terminals. An inexpensive color terminal can display the same picture—although much less vividly—as an expensive, dedicated graphics terminal.

Color mapping, which is used in color modes 1 and 2, allows a terminal to display a wide spectrum of colors without requiring a large amount of memory. The Atari 400 and 800 are two of the few home computers that make use of this technology (see "Computer Animation with Color Registers" by David Fox and Mitchell Waite, BYTE, November 1982, page 194).

In color mapping, if we return to the above analogy, the robot has the three primary-color inkwells again and a set of, say, 16 pens numbered 0 through 15. Using NAPLPS, we can instruct the robot to mix various colors in each of the pens. We can then instruct the robot to draw with a given pen, referring to it by its number rather than by its color. In a computer, we would store the color information not in a pen, but in a color register as part of a color map or color table.

With NAPLPS, an inexpensive color terminal can display the same picture—although much less vividly—as an expensive, dedicated graphics terminal.

In figure 14, we compare a system using fixed colors with one using color mapping. Both have the same amount of display memory (32K bytes). In the fixed-color system, the 4 bits in memory for each pixel specify one of 16 combinations of red, green, blue, and intensity. In the color-mapped system, the 4 bits refer to one of 16 color registers, each of which in turn refers to one of 4096 combinations of red, green, and blue.

Another important advantage of color mapping is that if we instruct the robot to change the color in a given pen, everything previously drawn with that pen will also change color. This amazing capability can be used to create some dramatic animation effects. These effects are typically referred to as color-table animation.

Color-table animation is a very complex area of NAPLPS. A mechanism has been provided that allows you to specify color interchanges in the color map based on timed relationships. (This command has been given the innocuous name BLINK.) Time intervals can be set in units of $\frac{1}{10}$ of a second, which allows compatibility with 60-Hz (U.S.) and 50-Hz (Europe) systems. Color-table animation will be discussed in greater detail in the third part of this series.

As we mentioned before, the major drawback of color modes 1 and 2 is the dependence on special hardware to achieve the full capabilities of the modes. This drawback was known at the time NAPLPS was designed, but it was determined that because of the incredible special effects that can be achieved using these modes they would be included. Anyone who does not have a need for these special effects should concentrate on using color mode 0 to insure portability of information.

Text Features

Text is handled as a subset of graphics. Text is a special form of graphics that involves predefined "templates" that are rectangular in shape. The rectangular templates can be scaled to any size and positioned *anywhere* on the unit screen. The

	b_6	0	0	1	1	1	1
	b_5	1	1	0	0	1	1
	b_4	0	1	0	1	0	1
b_3 b_2 b_1 b_0		2	3	4	5	6	7
0 0 0 0	0		0	@	P	`	p
0 0 0 1	1	!	1	A	Q	a	q
0 0 1 0	2	"	2	B	R	b	r
0 0 1 1	3	#	3	C	S	c	s
0 1 0 0	4	$	4	D	T	d	t
0 1 0 1	5	%	5	E	U	e	u
0 1 1 0	6	&	6	F	V	f	v
0 1 1 1	7	'	7	G	W	g	w
1 0 0 0	8	(	8	H	X	h	x
1 0 0 1	9	)	9	I	Y	i	y
1 0 1 0	10	*	:	J	Z	j	z
1 0 1 1	11	+	;	K	[	k	{
1 1 0 0	12	,	<	L	\	l	|
1 1 0 1	13	-	=	M	]	m	}
1 1 1 0	14	.	>	N	^	n	~
1 1 1 1	15	/	?	O	_	o	

Figure 15: *The Primary Character Set, which is very similar to ASCII. Note that bit 7 is not shown. The value of bit 7 would depend on which graphics area (GL or GR) this G-set was placed in.*

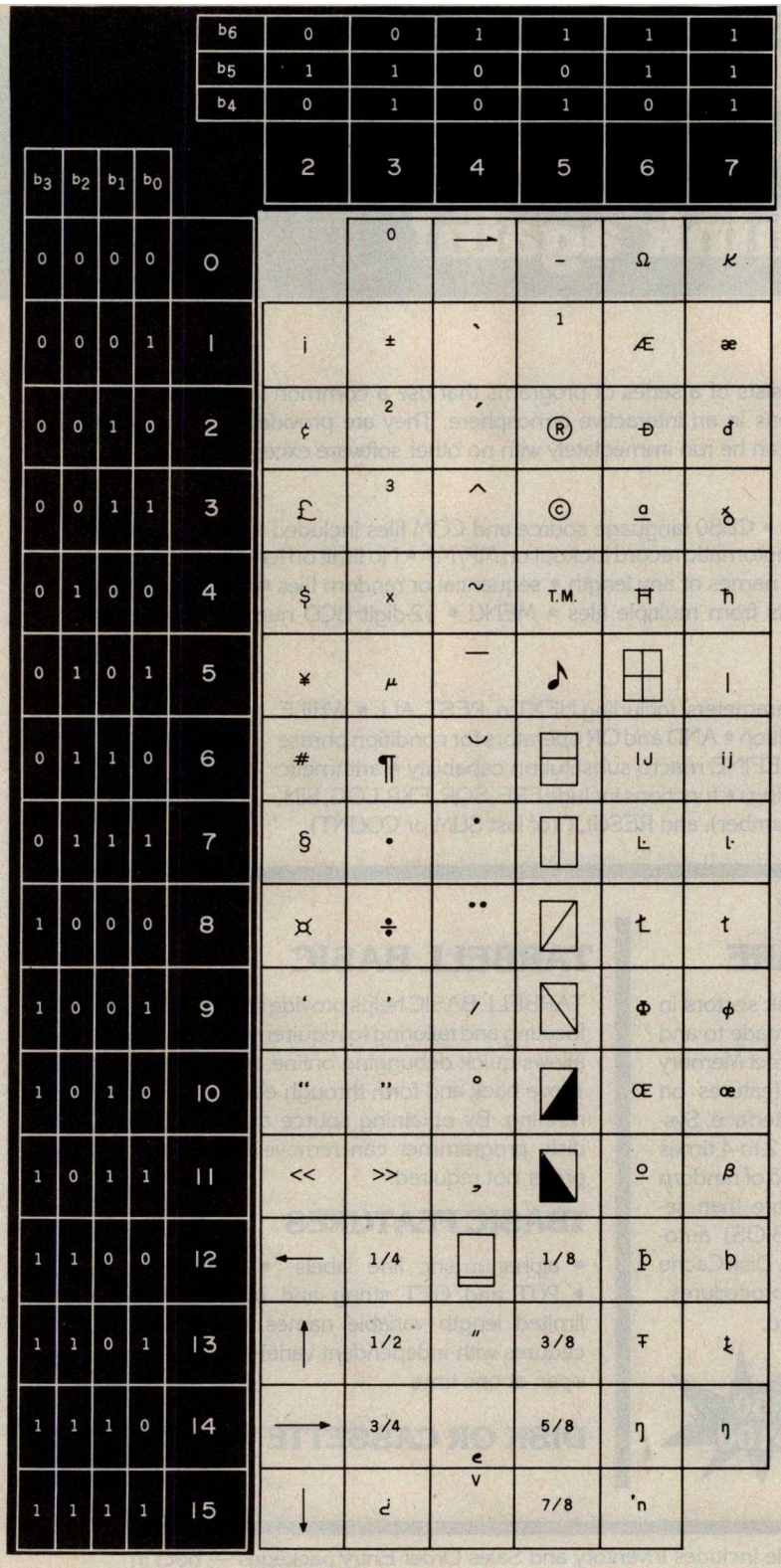


Figure 16: *The Supplementary Character Set of NAPLPS.*

"pattern" on the template is transferred to the screen, overwriting only those areas drawn with the template.

As mentioned earlier, NAPLPS currently specifies three fixed character sets and one redefinable character set. The Primary Character Set (ASCII) is shown in figure 15 on page 236. Most text is taken from this set. The ASCII character set is the default for the G0 and GL sets in figure 6. Therefore, it is accessed via the usual codes, 32 through 127 decimal.

A Supplementary Character Set has also been specified in NAPLPS (see figure 16). This character set contains a smorgasbord of symbols and international characters. Most applications will require only a few of these symbols. This character set is the default for the G2 designated set, and must be moved to GL or GR before these characters can be accessed.

The Mosaic Character Set is the third of the fixed sets (see figure 17 on page 242). Although the Mosaic characters do not look like text characters, they are treated exactly like text because of their rectangular shape. The Mosaics have very little use because of the extensive graphics capabilities contained in NAPLPS. The Mosaics are the default for the G3 designated set. Thus, they cannot be directly accessed without a G-set change. (We should have made it harder than that to use.)

The fourth text set in NAPLPS is the Dynamically Redefinable Character Set (DRCS). The templates in this character set are initially blank rectangles. We can define each template, however, by using NAPLPS to draw a pattern on the unit screen and mapping that pattern to the template. The pattern can be drawn with either graphics or text commands. Once the template is defined, it can be used just like any other character. (Yes, existing DRCS characters can even be used to define a new DRCS character.) Thus, the 96 characters in the DRCS set can be used to create custom fonts and special symbols.

NAPLPS provides a variety of text-oriented features, which can be applied to any of the four text sets. Figure 18 on page 244 illustrates many of the available capabilities. In parts 2 and 3 of this series, we will describe how these features are selected and applied.

Graphics Features

The graphics instructions (or primitives) are specified using codes from the Picture-Description Instruction (PDI) G-set. As shown in figure 19 on page 246, the PDI G-set is a 96-character set that is divided into

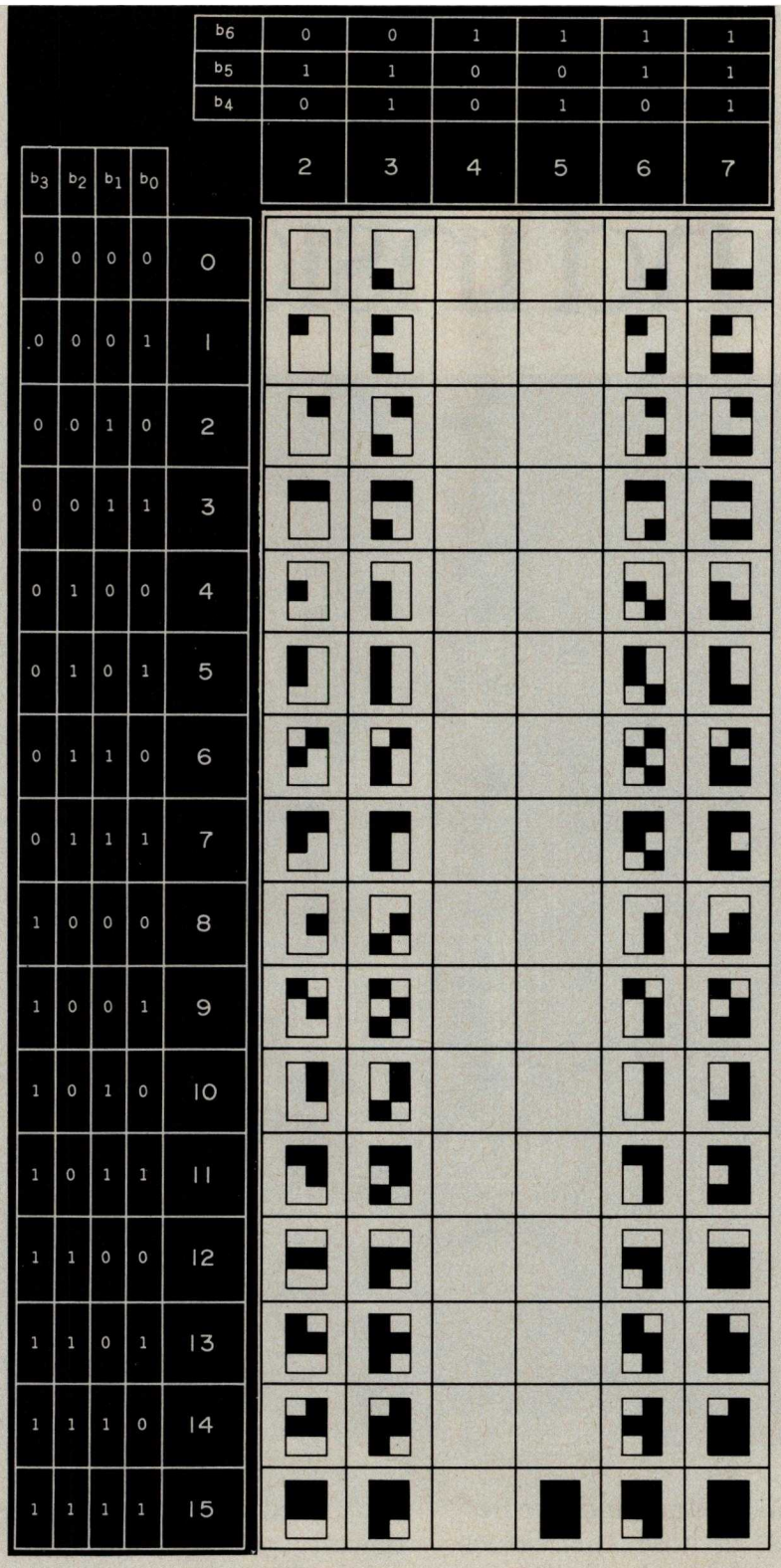


Figure 17: *The Mosaic Set.*

two smaller sets. The first 32 characters are graphics operation codes. These op codes are used to specify text control, drawing primitives, and color control.

The 64 codes in the right four columns of the PDI G-set are used to encode data for these op codes. These data bytes are encoded and interpreted according to the preceding op code. Six bits are available for information in each byte. Many of the op codes require multiple data bytes to encode one data item. Coordinates, for example, are typically encoded in 3 consecutive data bytes.

As shown in figure 20 on page 250, this distinction of op codes and data within the PDI G-set leads to a convenient decoding structure. Once it has been determined that a code falls in the PDI set, bit 6 (the seventh from the right) can be used to determine if an op code is specified or data. If bit 6 is 0, the byte is interpreted as an op code; if it is 1, it is a data byte.

Such a distinction is necessary because the picture-description instructions have been set up so that a *variable* amount of data can follow an op code. The bytes following the op code are assumed to be data as long as bit 6 is a 1.

Figure 21 on page 250 illustrates how text, graphics, and color can be integrated to draw a simple picture. Approximately 180 bytes of NAPLPS were needed to specify this picture. In parts 2 and 3, we will describe in detail how graphics commands for such pictures are encoded.

Control

Up to this point, the emphasis has been on the 96-character G-sets. Two C-sets (control sets), C0 and C1, are also specified in NAPLPS. These control sets contain the codes needed to accomplish the G- and C-set swapping. They also contain codes for moving the cursor, controlling the DRCS, clearing the screen, and so on.

Figure 22 on page 252 illustrates the C0 and C1 control sets. The C0 set should be familiar to those of you who have worked with ASCII. The C1 set contains a variety of codes associated with the new features of NAPLPS.

A mechanism has been provided, but not used, that allows C-sets to be changed like G-sets. The C-sets were originally going to be used whenever a small (fewer than 32) number of similar codes were added to NAPLPS. As it turns out, the 96-character G-sets have proven to be more useful. The C-sets have ended up becoming a catchall for codes that do not seem to "fit" (either physically or logically) anywhere else. This

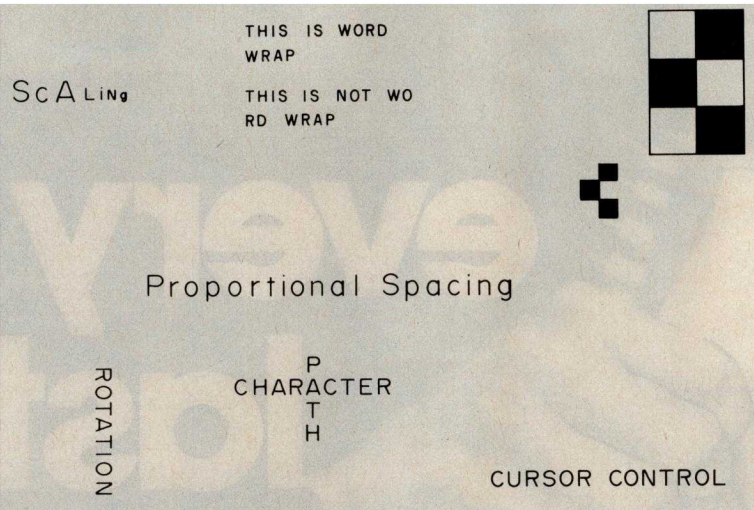


Figure 18: *Some examples of the text and Mosaic features of NAPLPS.*

compromise was not desired, but compromises such as this occur frequently when standards are being developed.

User Input

Because of the heavy emphasis on text and graphics in NAPLPS, the *user-input* features are often overlooked. User input is needed to allow a terminal user to enter information that will eventually be sent to the central host computer. This input could be used to request information from a database, order products, schedule an airline reservation, or send electronic mail.

User input has been integrated with the rest of NAPLPS in an elegant manner. Certain areas or fields of the unit screen can be designated as user-input areas. These areas are called *unprotected fields*.

The user can enter information into the unprotected fields using a variety of input devices such as keyboards, light pens, joysticks, graphics tablets, and even a "mouse." Information entered in the fields is stored as NAPLPS data. The user must eventually indicate (usually via a Send key) that all the information has been entered and should be sent to the host.

When the host computer receives the block of information, it may or may not decode it, depending on the application. For example, a graphics electronic-mail message would merely be sent to the appropriate addressee and would not have to be decoded.

The text of a message does not have to be entered on rigid lines as in most terminal systems. In applications such as electronic mail, a user who

		b_6	0	0	1	1	1	1
		b_5	1	1	0	0	1	1
		b_4	0	1	0	1	0	1
b_3	b_2	b_1	b_0	2	3	4	5	6	7
0	0	0	0	0	RESET	RECT (OUT-LINED)	NUMERIC DATA		
0	0	0	1	1	DOMAIN	RECT (FILLED)			
0	0	1	0	2	TEXT	SET & RECT (OUT-LINED)			
0	0	1	1	3	TEXTURE	SET & RECT (FILLED)			
0	1	0	0	4	POINT SET (ABS)	POLY (OUT-LINED)			
0	1	0	1	5	POINT SET (REL)	POLY (FILLED)			
0	1	1	0	6	POINT (ABS)	SET POLY (OUT-LINED)			
0	1	1	1	7	POINT (REL)	SET & POLY (FILLED)			
1	0	0	0	8	LINE (ABS)	FIELD			
1	0	0	1	9	LINE (REL)	INCR POINT			
1	0	1	0	10	SET & LINE (ABS)	INCR LINE			
1	0	1	1	11	SET & LINE (REL)	INCR POLY (FILLED)			
1	1	0	0	12	ARC (OUT-LINED)	SET COLOR			
1	1	0	1	13	ARC (FILLED)	WAIT			
1	1	1	0	14	SET & ARC (OUT-LINED)	SELECT COLOR			
1	1	1	1	15	SET & ARC (FILLED)	BLINK			

Figure 19: *The operation codes (or op codes) of the Picture-Description Instruction (PDI) G-set. The four columns on the right (that is, bits 0 through 5) are used as data for various op codes.*

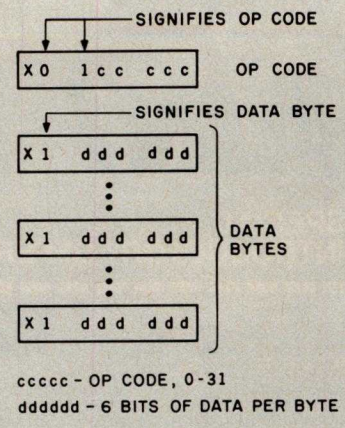


Figure 20: *In the PDI G-set of NAPLPS, op codes are distinguished from data bytes by bit 6. If bit 6 is 0, the byte is an op code; otherwise, it is a data byte.*

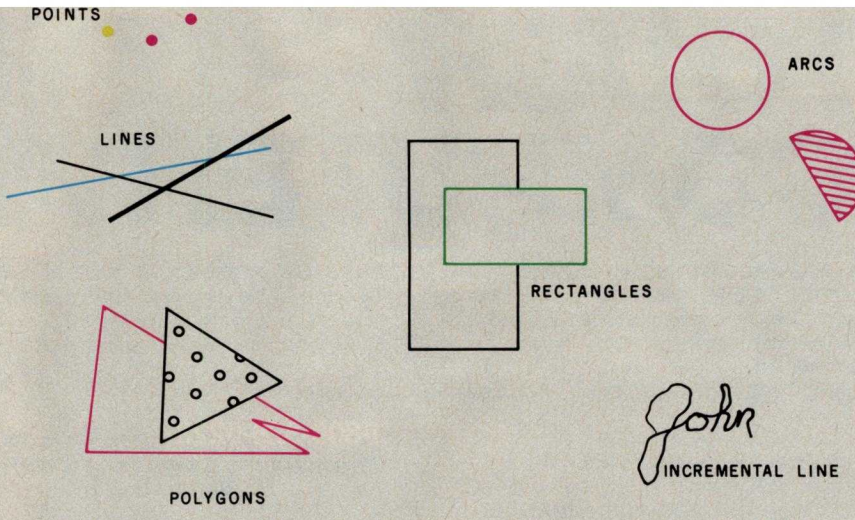


Figure 21: *Some examples of pictures that can be created with NAPLPS instructions. Approximately 180 bytes would be used to encode the entire figure. The signature alone requires 51 bytes.*

has the appropriate input device can even send *handwritten* messages using NAPLPS as the encoding mechanism.

The best analogy to describe user input in NAPLPS is to imagine that the user is handed one or more blank sheets of paper. (When the three-dimensional mode is supported, the user will be given an empty box.) The user is able to type on the paper, draw a sketch on the paper, or do anything that his or her terminal allows.

The "paper" is eventually passed to a host computer, where it can be forwarded to another user (electronic mail), stored for later recall, or analyzed by the host. The analysis by the host can be minimal or extensive, again depending on the application.

At this point, remember that NAPLPS is only a sixth-level specification in a seven-level model.

b_3	b_2	b_1	b_0		b_7 0 0 b_6 0 0 b_5 0 0 b_4 0 1 0	1
0	0	0	0	0	NUL	DLE
0	0	0	1	1	SOH	DC_1
0	0	1	0	2	STX	DC_2
0	0	1	1	3	ETX	DC_3
0	1	0	0	4	EOT	DC_4
0	1	0	1	5	ENQ	NAK
0	1	1	0	6	ACK	SYN
0	1	1	1	7	BEL	ETB
1	0	0	0	8	APB (BS)	CAN
1	0	0	1	9	APF (HT)	SS2
1	0	1	0	10	APD (LF)	SUB
1	0	1	1	11	APU (VT)	ESC
1	1	0	0	12	CS (FF)	APS
1	1	0	1	13	APR (CR)	SS3
1	1	1	0	14	SO	APH
1	1	1	1	15	SI	NSR

b_3	b_2	b_1	b_0		b_7 1 1 b_6 0 0 b_5 0 0 b_4 0 1 8	9
0	0	0	0	0	DEF MACRO	PROTECT
0	0	0	1	1	DEFP MACRO	EDC_1
0	0	1	0	2	DEFT MACRO	EDC_2
0	0	1	1	3	DEF DRCS	EDC_3
0	1	0	0	4	DEF TEXTURE	EDC_4
0	1	0	1	5	END	WORD WRAP ON
0	1	1	0	6	REPEAT	WORD WRAP OFF
0	1	1	1	7	REPEAT TO EOL	SCROLL ON
1	0	0	0	8	REVERSE VIDEO	SCROLL OFF
1	0	0	1	9	NORMAL VIDEO	UNDER LINE START
1	0	1	0	10	SMALL TEXT	UNDER LINE STOP
1	0	1	1	11	MED TEXT	FLASH CURSOR
1	1	0	0	12	NORMAL TEXT	STEADY CURSOR
1	1	0	1	13	DOUBLE HEIGHT	CURSOR OFF
1	1	1	0	14	BLINK START	BLINK STOP
1	1	1	1	15	DOUBLE SIZE	UNPRO-TECT

Figure 22: *The two control sets used in NAPLPS.*

NAPLPS merely provides a vehicle for the seventh level (commonly called the application level) that comprises the application programs and software (e.g., a banking program) that will run on NAPLPS. Many special applications can be developed and standardized at level 7 using NAPLPS as a foundation. These applications may be very specialized and might use only a subset of NAPLPS.

When we discuss user input, it should be noted that NAPLPS was *not* developed as a standard to be used for massive amounts of data entry in large data-processing centers. NAPLPS was developed to be used by people at home, at work, and at play. It was designed to be elegant and free-form.

NAPLPS was designed in this manner based on the assumption that most people do not want to interact with computers in robot-like ways. People will enter data by looking at menus and pointing to selections, rather than learning some complex command syntax. As we mentioned earlier, with a graphics tablet or other digitizer, people will even be able to input handwritten messages. Studies have shown that people want as much of their personality as possible to be reflected in their communication. And they expect that if they enter something reasonable, that it should be accepted and handled in a reasonable manner.

Macros

Macros (or macroinstructions) are specified in NAPLPS to reduce the amount of data that must be transmitted from the host to the terminal. Macros provide a mechanism whereby a frequently used multibyte string of text and/or graphics can be represented by a single-character macro. If the name of that macro appears later in the incoming data stream, the terminal retrieves the multibyte string and inserts it into the incoming stream in place of the macro name.

Once the string has been inserted into the incoming stream, the terminal processes it as if it had come from the host. Also, nesting of macros is allowed so that one macro can be used to retrieve several other macros. Of course, you must be careful to avoid looping and recursive macros that will endlessly refer to each other.

Ninety-six macro names are available. NAPLPS allows a unique, variable-length string to be stored for each name. Also, macros can be used in two directions: from the host to the

NAPLPS CODE

```
DEFINE MACRO 26
   SELECT BLUE
   CLEAR SCREEN
   SELECT WHITE
   POSITION (.05, .25)
   TEXT "READY:"
END
 •
 •
 •
MACRO 26
```

RESULT

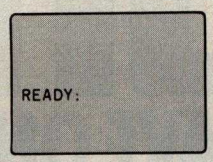


Figure 23: *An example of the use of macros in NAPLPS. Each time the code for Macro 26 occurs in the data stream, the word "READY:" will appear on a blank screen.*

terminal and vice versa. The direction can be specified when the macro is defined. The typical direction is to expand the macro *into* the terminal as described above. In the so-called transmit macros, the expansion of the macro occurs toward the host.

Transmit macros are usually associated with programmable function keys on the terminal. When a key is pressed, the string associated with the macro and the key is sent to the host.

Figure 23 illustrates a typical application of macros. Here, a macro has been defined. In this case, it was given the number 26. Later in the stream of NAPLPS instructions, the macro name 26 appears and the macro is expanded and processed by the terminal. The screen will be cleared to blue, the color white will be selected, and the word "READY:" will appear one-fourth of the way up the screen and a little in from the left edge. (Note that on the display screen the word "READY:" may appear to be *one-third* of the way up from the bottom; this results from the fact that the top quarter of the unit screen is not displayed.) Only 1 byte was sent to invoke this multibyte sequence. With this type of compression, a system can be made to appear very fast, even over 300-bit-per-second data links.

The Future of NAPLPS

NAPLPS has finally started to emerge as the most extensive text and graphics standard in existence. Many companies have hundreds of people working on NAPLPS-related projects. A survey in *Data Communications* magazine predicted that NAPLPS will be one of the most significant achievements in information exchange in the latter half of this century.

Part of the reason for this popularity is the fact that NAPLPS is not only a video-graphics protocol but an information-exchange language. NAPLPS has been used to encode pictures for plotters, printers, laser printers, and phototypesetters. NAPLPS can be used to encode precise descriptions of logos, trademarks, and physical objects, things which heretofore have been very difficult to describe precisely.

NAPLPS comes at a time when the information industry is bursting with new technology that exceeds existing standards for information interchange. NAPLPS is a standard that pushes this new technology to its limits and still provides the capability to accommodate unknown expansions.

NAPLPS is only the tip of the iceberg. In subsequent parts of this series, we will describe how NAPLPS fits into the larger scheme of local and regional area networks and distributed intelligent-terminal systems. Topics such as down-loading, file transfer, and operating-system evolution and compatibility will be covered.

Next month, we will begin to describe in detail how to write and decode NAPLPS information. In the meantime, anyone interested in obtaining more information about NAPLPS should obtain a copy of the ANSI standard specification. ■

NAPLPS: A New Standard for Text and Graphics

Part 2: Basic Features

How to encode text and simple graphics elements in a standard and efficient manner.

Jim Fleming
Unir Corporation
Suite 106
5987 East 71st St.
Indianapolis, IN 46220

Last month in part 1 of this series we introduced the North American Presentation-Level-Protocol Syntax (NAPLPS, or "nap-lips"), which is an ASCII-like standard that can be used to facilitate the interchange of both textual and graphical information. The graphical information is encoded in a very portable and resolution-independent form, which can be displayed on a large number of suitably equipped display terminals, printers, or plotters.

This month the basic features and specific coding formats of NAPLPS are introduced. The emphasis will be on the set of Picture-Description Instructions (PDIs), around which most of the important features of NAPLPS revolve.

A Picture Is Worth 284 Bytes

The easiest way to explain the detailed coding formats of NAPLPS is to use the simple picture (or frame) shown in figure 1 (on page 164), which illustrates many of the basic NAPLPS features. Listing 1 (pages 154-163) is an annotated version of the NAPLPS codes used to produce this picture. As you can see, although the annotated listing is quite long, the actual coding consists of only 284 bytes.

For the sake of simplicity, this picture was created using the 7-bit form of NAPLPS. As you may remember from last month, NAPLPS can use either 7 or 8 bits. If we had used the 8-bit form, the coding would be even shorter.

Op Codes and Operands

As can be seen in listing 1, a Picture-Description Instruction usually consists of an op code and an operand. The op code specifies a particular function; the optional operand(s) specify the data needed by the function. Figure 2 (on page 166) illustrates the general op code/operand structure used in NAPLPS.

In NAPLPS it is very easy to distinguish between the op codes and the operands. As can be seen, bit 6 is a 0 for an op code and a 1 for an operand. This distinction allows us to have variable-length operands, as long as each operand byte has bit 6 set to a 1. Another nice feature is that if the PDIs are presented in octal form as in listing 1, it is easy to distinguish the operands from the op codes. Octal codes with a first digit of 0 (e.g., 045) are op codes, while a first digit of 1 (e.g., 154) indicates an operand.

Bit 5 will always be a 1 for an op code. This distinguishes op codes from the standard control codes in the C0 set. The lower 5 bits of an op-code byte are used to indicate the particular function. These 5 bits accommodate 32 op codes, which are shown in figure 3. Most of these op codes will be covered in this article.

The operand bytes shown in figure 3 all have bit 6 set to 1. The lower 6 bits (bits 0 through 5) are thus available to encode data, the format of which is dependent on the op code preceding the data.

The 6 bits available in each operand byte can be formatted in a variety of ways. Figure 4 illustrates the four standard operand-encoding formats used in NAPLPS.

The *fixed* format for operand encoding is the simplest and most flexible. (Isn't it interesting that something "fixed" can be "flexible"?) Fixed-format operands are used for small bit fields (6 bits or less) and often contain a few suboperands. For example, in the Text op code (see figure 7), a fixed operand is used to encode the Text Rotation (2 bits: 0, 90, 180, or 270 degrees), Character Path (2 bits: Right, Left, Up, or Down), and Character Spacing (2 bits: 1, 1.25, 1.5, or Proportional). The fixed-format operands are used in most of the

Text continued on page 164

About the Author

Jim Fleming was a member of the original small group of engineers at Bell Laboratories who developed PLP (Presentation-Level Protocol). PLP was later standardized as NAPLPS by the ANSI X3L2.1 committee. He is now an independent consultant specializing in interactive computing systems.

Listing 1: *An annotated listing of NAPLPS codes used to produce the designs in figure 1. Note that each byte is given in its octal form. This makes it easy to distinguish op codes (first digit = 0) from operands (first digit = 1). Coordinates are described in terms of both their fractional form and their equivalent form for a 256 by 256 screen. For example, in lines 11–13 the coordinates (0.375,0.25) are equivalent to (96,64) on a 256 by 256 grid. The notation (dx,dy) refers to coordinates relative to the present drawing point.*

```
Byte  Octal Symbolic
No.   Form  Form     Description
-------------------------------------------------------------------------
        .
        .     Get ready for graphics (7 Bit Mode)
        .
  1     016   SO       Select G1 (PDI Graphics)
        .
        .     Set color to BLUE
        .
  2     074   SET      Set Color
  3     111   BLU      X100B00B
        .
        .     Draw the sky by clearing
        .     the screen to the current color (BLUE)
        .
  4     040   RES      Reset
  5     120            Clear screen to current color
        .
        .     Change color to GREEN
        .     for the grass
        .
  6     074   SET      Set Color
  7     144   GRN      X1G00G00
        .
        .     Make sure polygons are not highlighted
        .     or textured
        .
  8     043   TEX      Texture
  9     100            Solid areas, lines and no highlight
        .
        .     Draw the grass
        .
 10     067   SPF      Set Polygon Filled
 11     111             }
 12     140             } - (x,y) = (.375,.25) => (96,64)
 13     100             }
        .
 14     110             }
 15     140             } - (dx,dy) = (+.375,+.0) => (+96,+0)
 16     100             }
        .
 17     110             }
 18     102             } - (dx,dy) = (+.25,+.0625) => (+64,+16)
 19     100             }
        .
 20     106             }
 21     106             } - (dx,dy) = (+.0,-.3125) => (+0,-80)
 22     100             }
        .
 23     140             }
 24     100             } - (dx,dy) = (-1.0,+.0) => (-256,+0)
 25     100             }
        .

 26     100             }
 27     106             } - (dx,dy) = (+.0,+.21484375) => (+0,+55)
 28     107             }
        .
 29     100             }
```

Listing 1 continued on page 156

Listing 1 continued:

```
30   152          } - (dx,dy) = (+.171875,+.0625) => (+44,+16)
31   140          }
       .
       .      Change to RED
       .
32   074      SET     Set Color
33   122      RED     X1OR0OR0
       .
       .      Make sure highlighting is on
       .
34   043      TEX     Texture
35   104
       .
       .      Draw the house
       .
36   044      SPA     Point Set Absolute
37   110              }
38   127              } - (x,y) = (.3125,.234375) => (80,60)
39   104              }
       .
40   061      REF     Rectangle Filled
41   100              }
42   174              } - (dx,dy) = (+.21875,+.125) => (+56,+32)
43   100              }
       .
       .      Draw the roof
       .
44   045      SPR     Point Set Relative
45   170              }
46   104              } - (dx,dy) = (-.234375,+.125) => (-60,+32)
47   140              }
       .
48   074      SET     Set Color
49   100      BLK
       .
50   065      POF     Polygon Filled
51   100              }
52   141              } - (dx,dy) = (+.125,+.05859375) => (+32,+15)
53   107              }
       .
54   107              }
55   146              } - (dx,dy) = (+.125,-.0625) => (+32,-16)
56   101              }
       .
       .      Label the "House"
       .
57   045      SPR     Point Set Relative
58   107              }
59   125              } - (dx,dy) = (+.078125,-.078125) => (+20,-20)
60   144              }
       .
61   017      SI      Select G0 (ASCII Text)
       .
       .      "House"
       .
62   110      H
63   157      o
64   165      u
65   163      s
66   145      e
       .
       .      Back to graphics
       .
67   016      SO      Select G1 (PDI Graphics)
       .
       .      Set color to CYAN (Light Blue)
       .
68   074      SET     Set Color
69   155      CYN     X1G0BG0B
       .
       .      Label "BIRDS" before drawing them
```

Listing 1 continued on page 159

Listing 1 continued:

```
      .
 70   044      SPA      Point Set Absolute
 71   102               }
 72   150               } - (x,y) = (.15625,.52734375) => (40,135)
 73   107               }
      .
 74   017      SI       Select G0 (ASCII Text)
      .
      .       "BIRDS"
      .
 75   102      B
 76   111      I
 77   122      R
 78   104      D
 79   123      S
      .
      .       Back to Graphics
      .
 80   016      SO       Select G1 (PDI Graphics)
      .
      .       Draw bird with black wing tips
      .
 81   057      SAF      Set Arc Filled
 82   101               }
 83   167               } - (x,y) = (.1953125,.46875) => (50,120)
 84   120               }
      .
 85   107               }
 86   107               } - (dx,dy) = (+.015625,-.015625) => (+4,-4)
 87   144               }
      .
 88   107               }
 89   107               } - (dx,dy) = (+.0078125,-.015625) => (+2,-4)
 90   124               }
      .
 91   055      ARF      Arc Filled
 92   100               }
 93   100               } - (dx,dy) = (+.0078125,+.015625) => (+2,+4)
 94   124               }
      .
 95   100               }
 96   100               } - (dx,dy) = (+.0234375,+.0234375) => (+6,+6)
 97   166               }
      .
      .       Draw bird without black wing tips
      .
 98   043      TEX      Texture
 99   100
      .
100   045      SPR      Point Set Relative
101   100               }
102   111               } - (dx,dy) = (+.03515625,+.0390625) => (+9,+10)
103   112               }
      .
104   055      ARF      Arc Filled
105   107               }
106   107               } - (dx,dy) = (+.015625,-.015625) => (+4,-4)
107   144               }
      .
108   107               }
109   107               } - (dx,dy) = (+.0078125,-.015625) => (+2,-4)
110   124               }
      .
111   055      ARF      Arc Filled
112   100               }
113   100               } - (dx,dy) = (+.0078125,+.015625) => (+2,+4)
114   124               }
      .
115   100               }
116   100               } - (dx,dy) = (+.0234375,+.0234375) => (+6,+6)
117   166               }
```

Listing 1 continued on page 161

Listing 1 continued:

```
        .
        .       Draw Cloud
        .
118     044     SPA     Point Set Absolute
119     122             }
120     160             } - (x,y) = (.6875,.5) => (176,128)
        .
121     074     SET     Set Color
122     177     WHT     X1GRBGRB
        .
123     055     ARF     Arc Filled
124     170             }
125     170             } - (dx,dy) = (-.015625,+.0234375) => (-4,+6)
126     146             }
        .
127     100             }
128     110             } - (dx,dy) = (+.04296875,+.01171875) => (+11,+3)
129     133             }
        .
130     055     ARF     Arc Filled
131     100             }
132     110             } - (dx,dy) = (+.03125,+.02734375) => (+8,+7)
133     107             }
        .
134     107             }
135     107             } - (dx,dy) = (+.0234375,-.01171875) => (+6,-3)
136     165             }
        .
137     055     ARF     Arc Filled
138     100             }
139     121             } - (dx,dy) = (+.06640625,+.03515625) => (+17,+9)
140     111             }
        .
141     107             }
142     105             } - (dx,dy) = (+0.0,-.078125) => (+0,-20)
143     104             }
        .
144     055     ARF     Arc Filled
145     177             }
146     157             } - (dx,dy) = (-.08203125,-.01953125) => (-21,-5)
147     133             }
        .
148     170             }
149     150             } - (dx,dy) = (-.06640625,+.01171875) => (-17,+3)
150     173             }
        .
151     065     POF     Polygon Filled
152     100             }
153     101             } - (dx,dy) = (+.02734375,+.03515625) => (+7,+9)
154     171             }
        .
155     100             }
156     110             } - (dx,dy) = (+.0546875,+.015625) => (+14,+4)
157     164             }
        .
158     107             }
159     126             } - (dx,dy) = (+.06640625,-.04296875) => (+17,-11)
160     115             }
        .
        .       Label "CLOUD"
        .
161     045     SPR     Point Set Relative
162     170             }
163     142             } - (dx,dy) = (-.1171875,+.078125) => (-30,+20)
164     124             }
        .
165     017     SI      Select G0 (ASCII Text)
        .
        .       "CLOUD"
        .
166     103     C
167     114     L
```

Listing 1 continued on page 162

Circle 304 on inquiry card. →

Listing 1 continued:

```
168   117     O
169   125     U
170   104     D
        .
        .     Back again
        .
171   016     SO      Select G1 (PDI Graphics)
        .
        .     Set color to CYAN again for the rain
        .
172   074     SET     Set Color
173   155             X1G0BG0B
        .
        .     Draw Rain using various textured lines
        .
174   045     SPR     Point Set Relative
175   107             }
176   104             } - (dx,dy) = (+.01953125,-.1171875) => (+5,-30)
177   152             }
        .
178   043     TEX     Texture
179   102             }
        .
180   051     LIR     Line Relative
181   177             }
182   165             } - (dx,dy) = (-.0390625,-.078125) => (-10,-20)
183   164             }
        .
184   045     SPR     Point Set Relative
185   100             }
186   122             } - (dx,dy) = (+.078125,+.0703125) => (+20,+18)
187   142             }
        .
188   043     TEX     Texture
189   101             }
        .
190   051     LIR     Line Relative
191   177             }
192   165             } - (dx,dy) = (-.0390625,-.078125) => (-10,-20)
193   164             }
        .
194   043     TEX     Texture
195   100             }
        .
196   045     SPR     Point Set Relative
197   100             }
198   122             } - (dx,dy) = (+.078125,+.08984375) => (+20,+23)
199   147             }
        .
200   051     LIR     Line Relative
201   177             }
202   165             } - (dx,dy) = (-.0390625,-.078125) => (-10,-20)
203   164             }
        .
204   045     SPR     Point Set Relative
205   100             }
206   122             } - (dx,dy) = (+.078125,+.0625) => (+20,+16)
207   140             }
        .
        .     Label the "RAIN" vertically
        .
208   042     TXT     Text
209   114             Char Path Down
        .
210   017     SI      Select G0 (ASCII Text)
        .
        .     "RAIN"
        .
211   122     R
212   101     A
213   111     I
214   116     N
        .
        .     Back to Graphics
        .
```

Listing 1 continued:

```
215   016     SO      Select G1 (PDI Graphics)
        .
        .     Reset to normal text
        .
216   042     TXT     Text
217   100             Char Path Right
        .
        .     Set color to BLACK
        .     (actually transparent)
        .
218   074     SET     Set Color
219   100     TRN     X1000000
        .
        .     Draw the road
        .
220   044     SPA     Point Set Absolute
221   100             }
222   100             } - (x,y) = (0.0,0.0) => (0,0)
223   100             }
        .
224   065     POF     Polygon Filled
225   120             }
226   106             } - (dx,dy) = (+.5,+.1953125) => (+128,+50)
227   102             }
        .
228   100             }
229   121             } - (dx,dy) = (+.078125,+.0546875) => (+20,+14)
230   146             }
        .
231   100             }
232   120             } - (dx,dy) = (+.078125,+.0) => (+20,+0)
233   140             }
        .
234   177             }
235   155             } - (dx,dy) = (-.0703125,-.0703125) => (-18,-18)
236   166             }
        .
237   167             }
238   142             } - (dx,dy) = (-.3515625,-.1796875) => (-90,-46)
239   162             }
        .
        .     Label the "ROAD"
        .
240   044     SPA     Point Set Absolute
241   120             }
242   102             } - (x,y) = (.5,.078125) => (128,20)
243   104             }
        .
244   017     SI      Select G0 (ASCII Text)
        .
        .     "ROAD"
        .
245   122     R
246   117     O
247   101     A
248   104     D
        .
249   016     SO      Select G1 (PDI Graphics)
        .
        .     Change Size of text
        .
250   042     TXT     Text
251   100             }
252   100             }
253   100             }
254   112             } - (dx,dy) = (+.046875,+.078125) => (+12,+20)
255   144             }
        .
        .     Draw BLACK "Figure 1"
        .     as base for drop shadow
        .
256   044     SPA     Point Set Absolute
257   112             }
258   105             } - (x,y) = (.25,.6859375) => (64,175)
259   107             }
```

Listing 1 continued on page 164

Listing 1 continued:

```
      .
260   017     SI      Select G0 (ASCII Text)
      .
      .     "Figure 1"
      .
261   106     F
262   151     i
263   147     g
264   165     u
265   162     r
266   145     e
267   040     space
268   061     1
      .
      .     Finish drop shadowing with yellow over black
      .
269   016     SO      Select G1 (PDI Graphics)
      .
270   074     SET     Set Color
271   166     YEL     X1GR0GR0
      .
272   044     SPA     Point Set Absolute
273   102             }
274   176             } - (x,y) = (.24609375,.6875) => (63,176)
275   170             }
      .
276   017     SI      Select G0 (ASCII Text)
      .
      .     "Figure 1"
      .
277   106     F
278   151     i
279   147     g
280   165     u
281   162     r
282   145     e
283   040     space
284   061     1
      .
      .     The end
      .
      .           Text is still large and
      .           YELLOW is the current color
      .
```

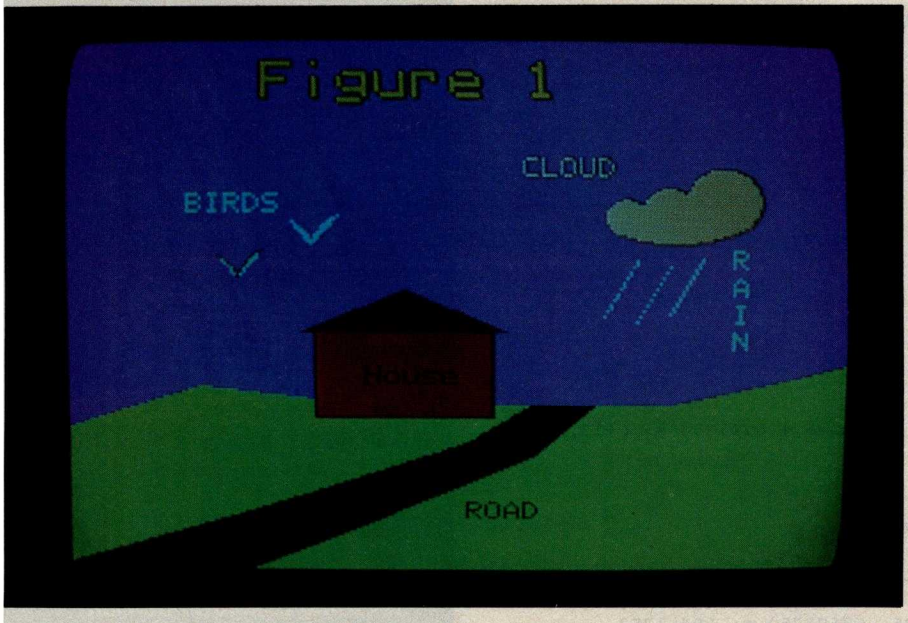


Figure 1: *A simple picture produced by the NAPLPS codes in listing 1. (Photo courtesy of the Unir Corporation.)*

Text continued from page 152:

"control-oriented" NAPLPS functions.

The *single-value* format is used when a common integer is needed. This format is used when specifying color indexes and blink rates (in tenths of a second). The single-value format is encoded using 1 to 4 bytes, each containing 6 bits of data. In the default mode, 1 byte is used, thus allowing numbers in the range 0 to 63 to be encoded. In the maximum mode (4 bytes or 24 bits), numbers from 0 to 16,777,215 can be specified.

The most common format in NAPLPS is the *multivalue* operand. The multivalue-operand format has two coordinate forms and a color form, as shown in figure 4.

The coordinate forms are used to encode (x,y) or (x,y,z) coordinate locations in the unit screen. In the two-dimensional mode, each 6-bit operand contains 3 bits of x and 3 bits of y. Multivalue operands are normally encoded in 3 bytes. Therefore, 9 bits of resolution are encoded for each coordinate. The 9 bits allow for a sign bit and 8 data bits, which results in coordinates suitable for a 256 by 256 resolution display.

NAPLPS supports multivalue operands up to 8 bytes. The 8 bytes each contain 6 data bits. Therefore, 48 bits are available to be split between the coordinates. In two-dimensional mode the 24 bits available for each coordinate can support displays with a resolution of 8 million by 8 million points! This exceeds the resolution of most media, including a page in this magazine.

The multivalue-operand format is also used for color specification. Various amounts of green, red, and blue are specified using this multibyte format. Each 6-bit data item contains 2 bits of each color. The colors are interlaced as shown in figure 4, with green being first and thus least likely to be truncated. This takes advantage of the fact that the human eye is more responsive to green than it is to red and blue.

The 8-byte multivalue-operand format will again yield 48 bits of color information that results in 280,000,000,000,000 colors. With the maximum display resolution and the

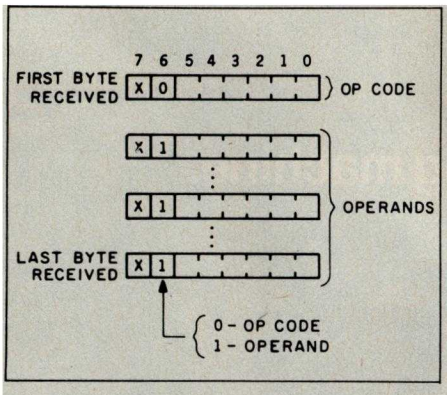


Figure 2: *The general structure for op codes and operands of the Picture-Description Instructions (PDIs) in NAPLPS.*

maximum color resolution, NAPLPS can support displays with 2^{96} bits of display memory! At today's memory prices, such a display would cost $750 billion billion billion dollars. (No wonder semiconductor companies are interested in NAPLPS.)

The final operand format is the *string* operand. This format is used when a long string of bits is needed that may require hundreds or thousands of bytes to encode. This format is used when sending high-resolution pictures and for encoding compressed chain-coded images. These techniques will be discussed in part 3 of this series.

The operand/op code encoding structure of NAPLPS allows a variety of formats and subformats. Many of the op codes contain one or more of the operand types. For example, the Text op code, which will be described in detail later on, is followed by two fixed-format operands and a multivalue operand. The total number of operand bytes for this op code is variable, but the first 2 bytes will always be interpreted as fixed-format bytes and the remaining bytes will be considered as part of a multivalue format. Because of the variable-length nature of the operand encoding in NAPLPS, operands can be truncated and/or omitted with a consistent result dependent on the op code active at the time.

	b_6	0	0	1	1	1	1
	b_5	1	1	0	0	1	1
	b_4	0	1	0	1	0	1
b_3 b_2 b_1 b_0		2	3	4	5	6	7
0 0 0 0	0	RESET	RECT (OUT-LINED)				
0 0 0 1	1	DOMAIN	RECT (FILLED)				
0 0 1 0	2	TEXT	SET & RECT (OUT-LINED)				
0 0 1 1	3	TEXTURE	SET & RECT (FILLED)				
0 1 0 0	4	POINT SET (ABS)	POLY (OUT-LINED)				
0 1 0 1	5	POINT SET (REL)	POLY (FILLED)				
0 1 1 0	6	POINT (ABS)	SET POLY (OUT-LINED)	NUMERIC DATA			
0 1 1 1	7	POINT (REL)	SET & POLY (FILLED)				
1 0 0 0	8	LINE (ABS)	FIELD				
1 0 0 1	9	LINE (REL)	INCR POINT				
1 0 1 0	10	SET & LINE (ABS)	INCR LINE				
1 0 1 1	11	SET & LINE (REL)	INCR POLY (FILLED)				
1 1 0 0	12	ARC (OUT-LINED)	SET COLOR				
1 1 0 1	13	ARC (FILLED)	WAIT				
1 1 1 0	14	SET & ARC (OUT-LINED)	SELECT COLOR				
1 1 1 1	15	SET & ARC (FILLED)	BLINK				

Figure 3: *The complete set of Picture-Description Instruction op codes in NAPLPS.*

Picture-Description Instructions

The Picture-Description Instructions (PDIs) are used to encode graphics images in NAPLPS. Codes from the PDI G-set and the ASCII-like text set can be intermixed on the same frame. Most of the common PDIs have been used to encode the image in figure 1. These PDIs are described here with references to the coding in listing 1.

Reset

The Reset PDI is illustrated in figure 5. It is used to clear the screen

text continued on page 170

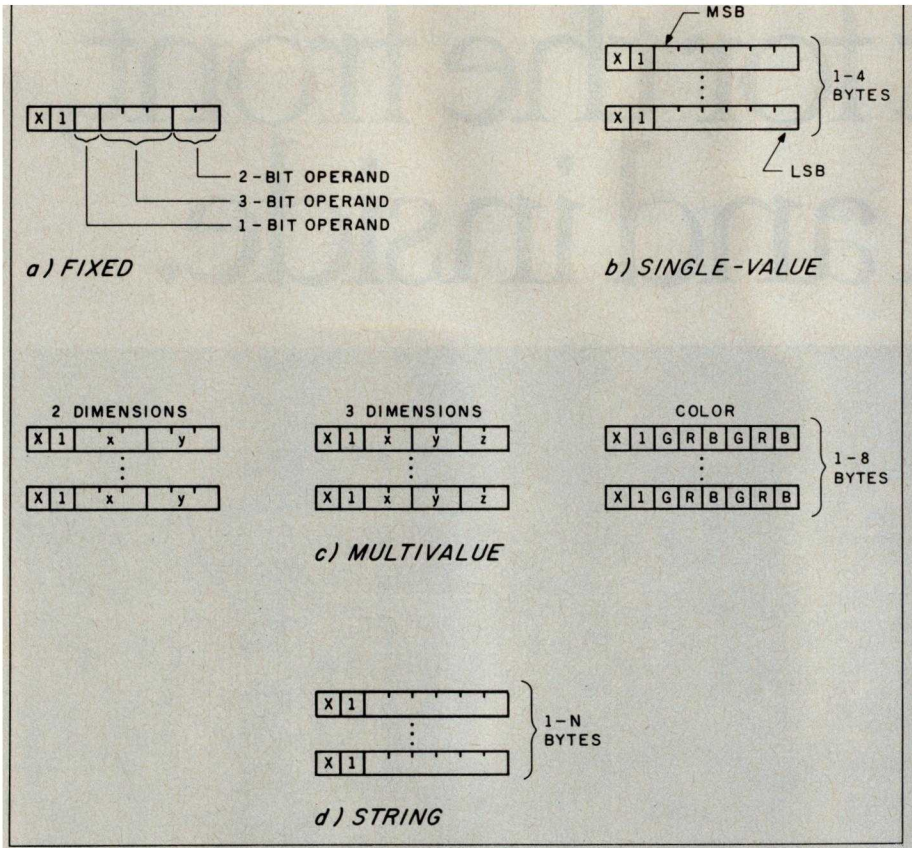


Figure 4: *The various formats for the operands of the PDIs in NAPLPS.*

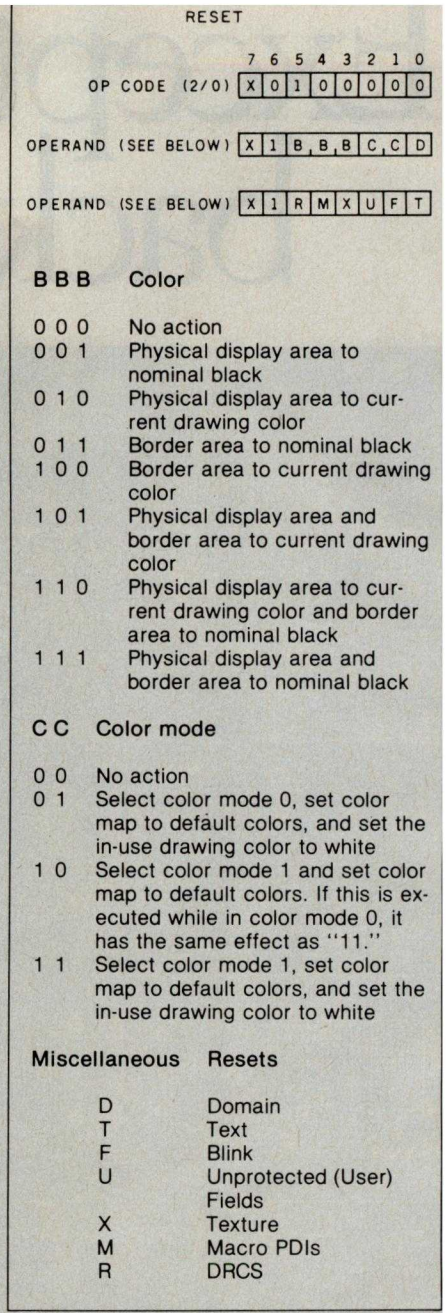


Figure 5: *The operand structure for the Reset instruction.*

and initialize various attributes. Two fixed-format operand bytes contain nine suboperands. The second operand byte can be omitted when those operations are not needed. If both operand bytes are omitted, a complete Reset is performed.

The screen is cleared based on the value in bits 4 to 6 of the first operand byte. The eight combinations are shown in figure 5. In the example frame (line 4), the screen is cleared once to establish the blue sky. The fixed-format operand (octal 120 at line 5) indicates that the screen should be cleared to the current in-use color (in this case, blue). Note that the second fixed-format operand byte is omitted. The op code at line 6 indicates that the previous operation and op code have ended.

Domain

The Domain PDI is used primarily to control the size of data operands for subsequent PDIs. As shown in figure 6, the Domain PDI is made up of a fixed-format operand followed by a multibyte operand. The fixed-format operand controls the size of single-value operands and multivalue operands as well as the dimensionality of coordinates.

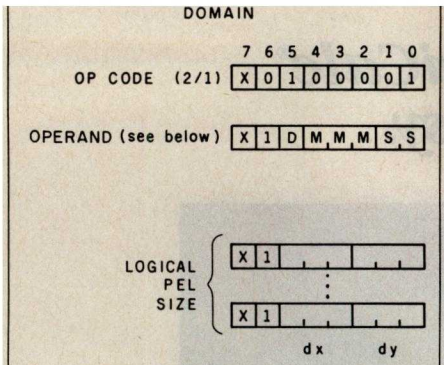
The multivalue operand is used to control the size of the logical drawing point.

Text

The Text PDI controls attributes related to text and "text-like" symbols. As discussed in part 1, text symbols are unique in the sense that they are rectangular templates that contain a figure. When a text symbol is requested, the proper template is positioned at the current drawing point, the template is scaled as specified by the text size, and the drawing is performed.

Figure 7 illustrates the Text PDI and operands. Two fixed-format operand bytes contain six suboperands. Each of the suboperands has four possible values. As can be seen, these suboperands control attributes such as rotation, spacing, and cursor style.

The multivalue operand following the two fixed-format operands is used to specify the size and orientation of the text template. The size is expressed in terms of relative coordinates, which we will indicate by the notation (dx,dy). This is to distinguish relative coordinates from absolute coordinates (x,y) that refer

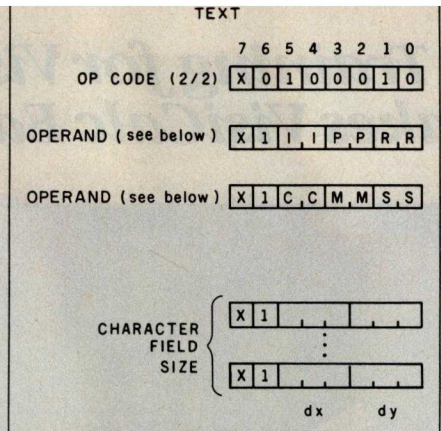


D	Dimensionality
0	two-dimensional
1	three-dimensional

M M M	Length of Multivalue Operands (Bytes)
0 0 0	1
0 0 1	2
0 1 0	3 (default)
0 1 1	4
1 0 0	5
1 0 1	6
1 1 0	7
1 1 1	8

S S	Length of Single-Value Operands (Bytes)
0 0	1 (default)
0 1	2
1 0	3
1 1	4

Figure 6: *The operand structure for the Domain instruction. The Logical Pel Size can be thought of as the size of the drawing pen.*



I I	Intercharacter Spacing
0 0	1 (default value)
0 1	1.25
1 0	1.5
1 1	Proportional spacing

P P	Character Path
0 0	Right (default)
0 1	Left
1 0	Up
1 1	Down

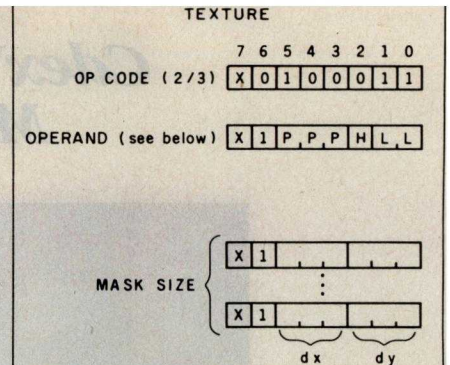
R R	Rotation
0 0	0 (default)
0 1	90
1 0	180
1 1	270

C C	Cursor Style
0 0	Underscore (default)
0 1	Block
1 0	Cross-hair
1 1	Custom

M M	Move Attribute
0 0	Move together (default)
0 1	Cursor leads
1 0	Drawing point leads
1 1	Moving independently

S S	Interrow Spacing
0 0	1 (default)
0 1	1.25
1 0	1.5
1 1	2

Figure 7: *The operand structure for the Text instruction.*



P P P	Texture Pattern
0 0 0	Solid (default pattern)
0 0 1	Vertical hatching
0 1 0	Horizontal hatching
0 1 1	Vertical and horizontal crosshatching
1 0 0	Mask A
1 0 1	Mask B
1 1 0	Mask C
1 1 1	Mask D

H	Highlight
0	Off
1	On

L L	Line Texture
0 0	Solid (default)
0 1	Dotted
1 0	Dashed
1 1	Dotted-dashed

Figure 8: *The operand structure for the Texture instruction.*

to specific points on the unit screen.

In the example frame, text is used to label the objects as well as the entire figure. Most of the text is encoded in the standard manner and therefore no Text PDI is needed. The first Text PDI appears in line 208 and is used to change the Character Path from left-to-right to down. This allows the word "RAIN" (lines 211–214) to be sent without repositioning the drawing point.

Note that the second fixed-format operand and the multivalue size operand are omitted because only the Character Path is being changed. Also note that because the Character Path is being changed, the other two suboperands in that byte (Intercharacter Spacing and Rotation) have to be restated or "refreshed." It is assumed that the NAPLPS code generator will always have knowledge of the current settings of these suboperands so that such a refresh is easy to do.

The Text PDI is used again in lines 250–255. The size of the text is changed to label the figure. The Character Path is also set to left-to-right. The (*dx,dy*) of (+0.046875, +0.078125) results in a character twice as big in both dimensions as the default characters. If you want to find out how many of these characters could fit on a line, you could divide 1.0 by 0.046875, which results in 21.3 characters per line.

It should be noted that no other Text PDIs appear after the one in line 250. At the end of the frame, the text size is still large. When the next frame is sent, the text size should be changed back to its default state. This is typically done with a global Reset at the beginning of the frame.

Texture

The Texture PDI applies to the texturing of filled areas and lines (see figure 8). Line texturing can be set so that dotted, dashed, or dotted-dashed lines will be drawn instead of the nor-

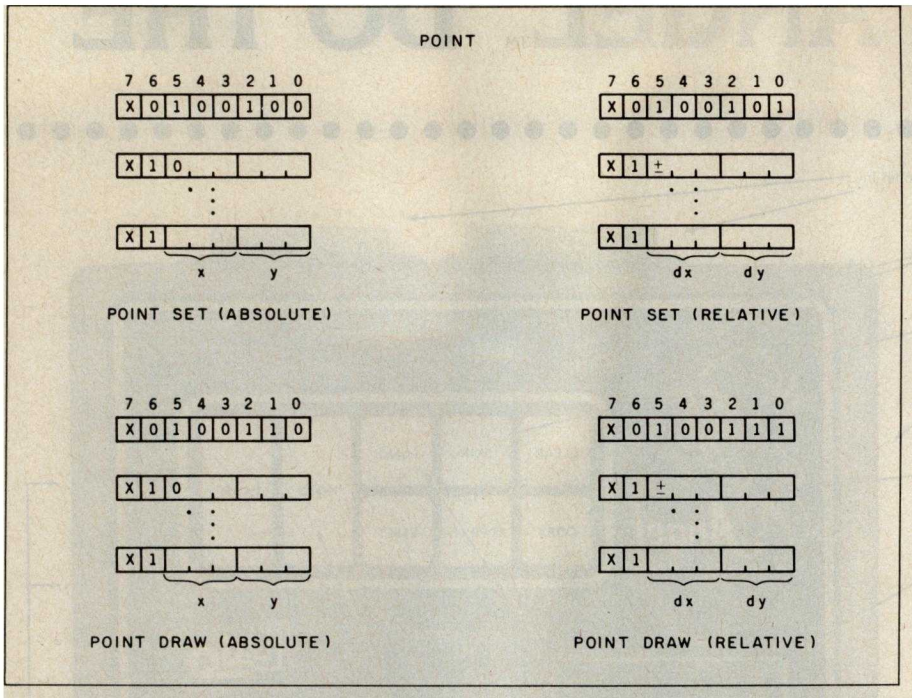


Figure 9: *The Point instructions in NAPLPS. Point Set merely moves the "drawing point" to the desired position. Point Draw actually draws a point at that position. Coordinates can be either absolute (x,y) or relative (dx,dy). The first bit of each coordinate is a sign bit. The remaining bits are encoded as fixed-point binary numbers, with the "binary point" assumed to be just to the right of the sign bit.*

mal solid line. A variety of area textures can be selected so that large objects can have recognizable interiors. The area textures can be chosen from a "stock" set of patterns or "programmable" patterns can be used.

A "cartoon-like" highlighting feature is included. When enabled, filled areas are highlighted (usually in black) to accent the edges. This is especially useful in low-resolution video-display systems that have trouble making rapid color changes.

The Texture PDI is used several times in the example frame (lines 8, 34, 98, 178, 188, and 194). The highlighting is turned off for the grass and on for the house. The highlighting is also used on the left bird to add a little diversity. The line textures are demonstrated in creating the rain (lines 171–203).

Outlined Drawings

The majority of drawings are created using the basic primitives *Point*, *Line*, *Arc*, *Rectangle*, and *Polygon*. All these primitives are supported in NAPLPS with each one having several forms.

Points

Points can be drawn on the unit screen in a variety of ways. As shown in figure 9, four Point PDIs are provided. Two of these commands are used to actually draw points (Point Draw), while the other two merely position the drawing point prior to drawing text or graphics (Point Set). The coordinates for both Point Draw and Point Set can be expressed in either absolute or relative terms.

At this point (no pun intended), it is probably useful to distinguish between the drawing point and the cursor. The drawing point is the imaginary pen point or brush tip that is used to draw graphics on the screen. The cursor is the typical block or underscore that marks the position where the next text entry will be made. The drawing point and cursor usually "track" each other, but this is not required. In other words, the cur-

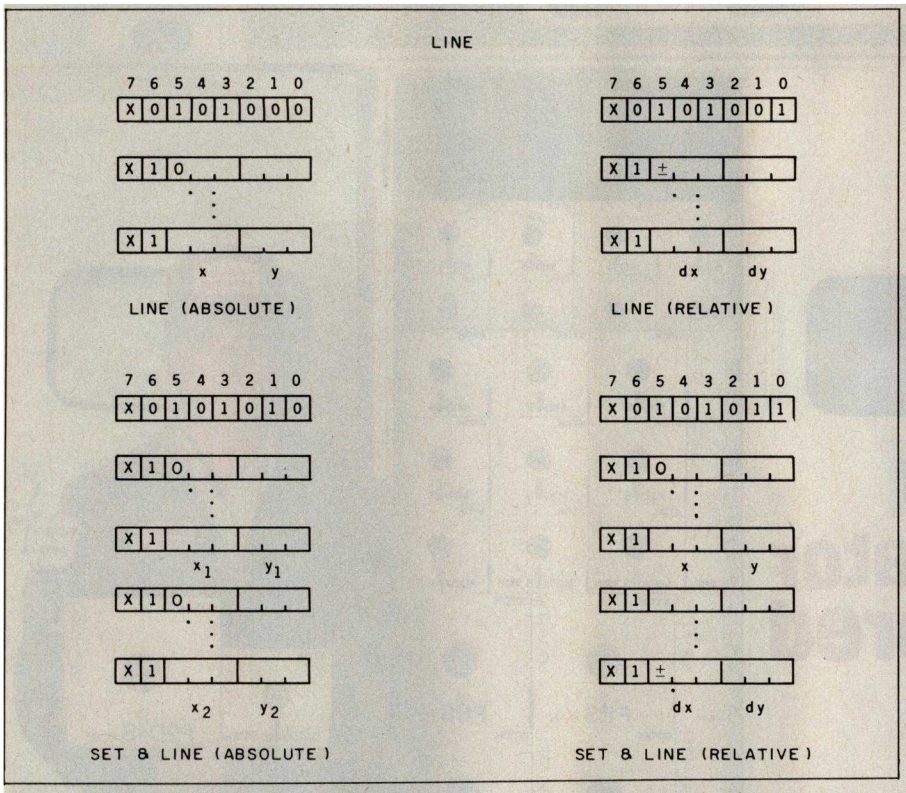


Figure 10: *The Line instructions. The Set & Line instructions move the drawing point to a new position and draw a line from that position. The Line instructions draw a line from the present drawing point.*

sor can be positioned on the screen and the drawing point can be moved independently.

The example frame uses the "Set" forms of the Point PDIs, but not the "Draw" forms. Line 36 is an example of a Set Point Absolute op code. This op code is used to position the drawing point to a specific place on the screen regardless of where the drawing point is currently located. This is in preparation for drawing the house.

Line 44 is an example of a Set Point Relative op code. This op code is followed by a (*dx,dy*) operand that specifies a distance to move from the current position. This move is made in preparation for drawing the roof. Note that the relative form of the op code is useful because the roof should always be "tied" to the house. If a specific (absolute) screen coordinate had been specified, the roof would be fixed at a certain location. In this example, if the initial coordinate (lines 37–39) is changed, the roof will move with the house.

Lines

Lines are used in almost every graphics display. Four forms of the Line PDI are provided, as shown in figure 10. The major difference in the four op codes is that two of them draw a line from the present drawing point and the other two draw from a new set point. Also, two of the op codes involve relative positions and two involve absolute positions.

Lines are used to create the rain in the example frame. The relative form of the Line PDIs is used in lines 180, 190, and 200. As mentioned, the lines are drawn using the current texture setting.

Arcs

The Arc PDIs are extremely powerful, but may be confusing to the casual observer. Most people can eventually be convinced that only one circular arc can be drawn through three points if two of the points are known to form the endpoints. In NAPLPS the three points on the arc are specified rather than the center and radius. The three points are specified just like other points in the unit screen.

text continued on page 180

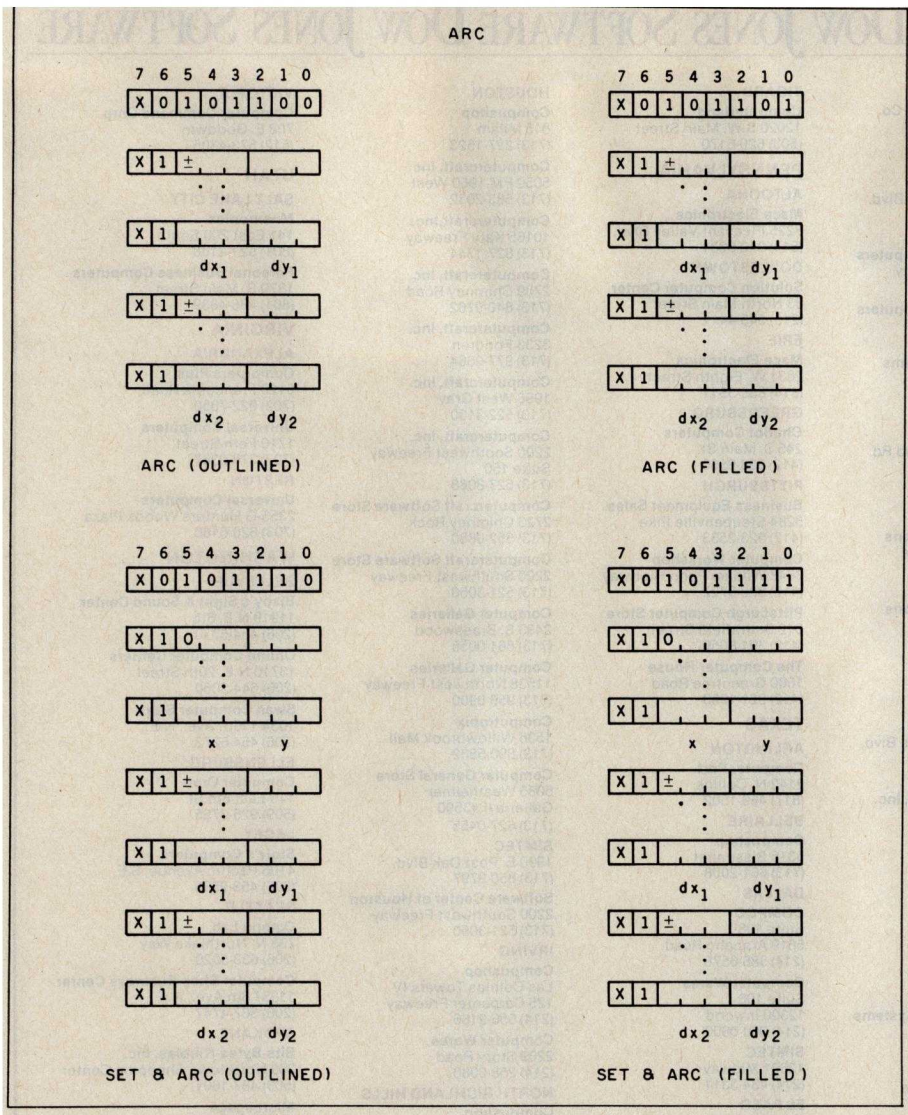


Figure 11: *The Arc instructions.*

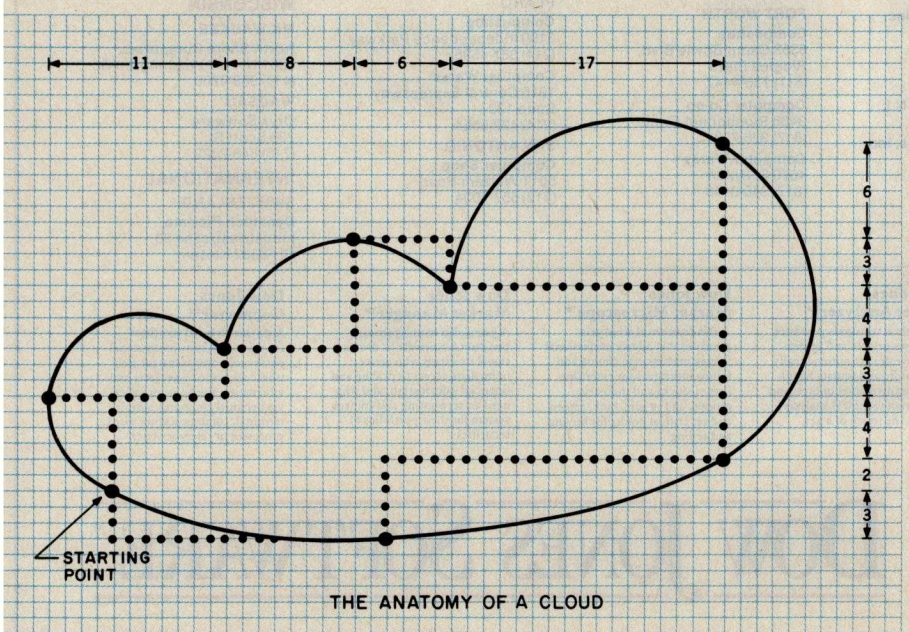


Figure 12: *A diagram showing how the cloud in figure 1 was constructed from four filled arcs and a filled polygon.*

Four forms of the Arc PDIs are included in NAPLPS, as shown in figure 11. Two of the forms allow arcs to be filled so that solid areas with curved edges can be created.

Arcs are used in the example frame to create the birds and the cloud. As shown in figure 12, the cloud is made up of four arcs and a polygon. The area between each arc and a line (or chord) connecting the endpoints of the arc is filled by the Arc (Filled) command. The Polygon (Filled) command fills the middle area.

Circles are a subset of the more general arc. If only two points are specified (instead of three), those points are assumed to form endpoints of a diameter of a circle. Circles can also be encoded using three points in the normal arc format, but the starting and ending points must be equal for a circle to be drawn.

A "hook" has been provided in NAPLPS so that it might eventually support complex curves or *splines*. These curves cannot be described by using simple arcs of circles. But if more than three points are specified for an arc, it should be possible to draw a smooth curve connecting the points. Until algorithms are developed that can efficiently draw a spline, lines can be used to connect the points.

Rectangles

Both filled and outlined rectangles are supported by NAPLPS. The four forms of the Rectangle PDI are shown in figure 13. Rectangles are described by specifying the opposite corner in terms of relative (dx,dy) coordinates. Negative values for dx or dy can be used to produce rectangles in various directions from the current drawing point.

One difference that should be noted with Rectangles is the final destination of the drawing point. Most drawing commands cause the drawing point to be left at the last point involved in the figure. In the case of the Rectangle, only the x coordinate is modified so that the drawing point moves horizontally. This allows for histograms or bar charts to be generated in an efficient manner.

A Rectangle is used to generate the

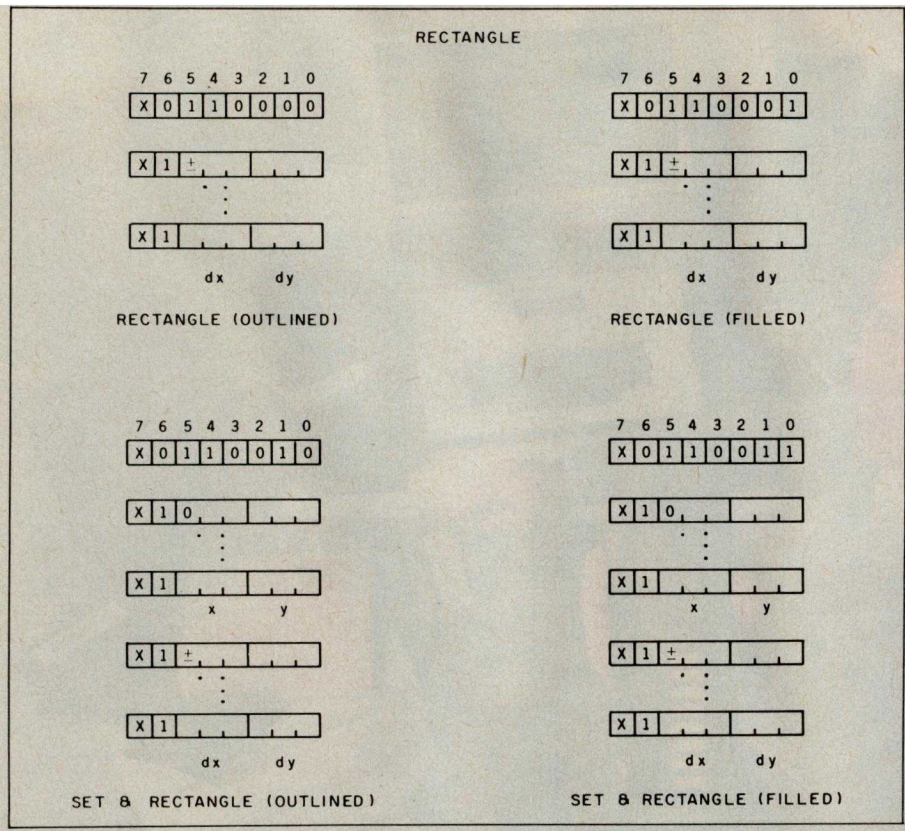


Figure 13: *The Rectangle instructions. Note that only one point is required to define a rectangle.*

house in the example frame. The op code at line 40 could have been a Set Rectangle Filled with the data from lines 37–39 moved into the operation. This would eliminate the need for the Point Set Absolute op code at line 36. Both encodings would yield the same result.

Polygons

The irregular Polygon is a very useful feature in NAPLPS. Many objects can be broken down into multisided irregular objects. These objects can be encoded using the endpoints of the lines forming the sides.

Four forms of the Polygon op code are available, as shown in figure 14. The outlined polygons do not offer much more than an efficient way to send a lot of lines. It should be noted that the last line in a polygon is not explicitly sent. The polygon is automatically "closed" by an edge connecting the last point sent and the starting point.

The filled polygons offer the ability to define an entire object disregarding

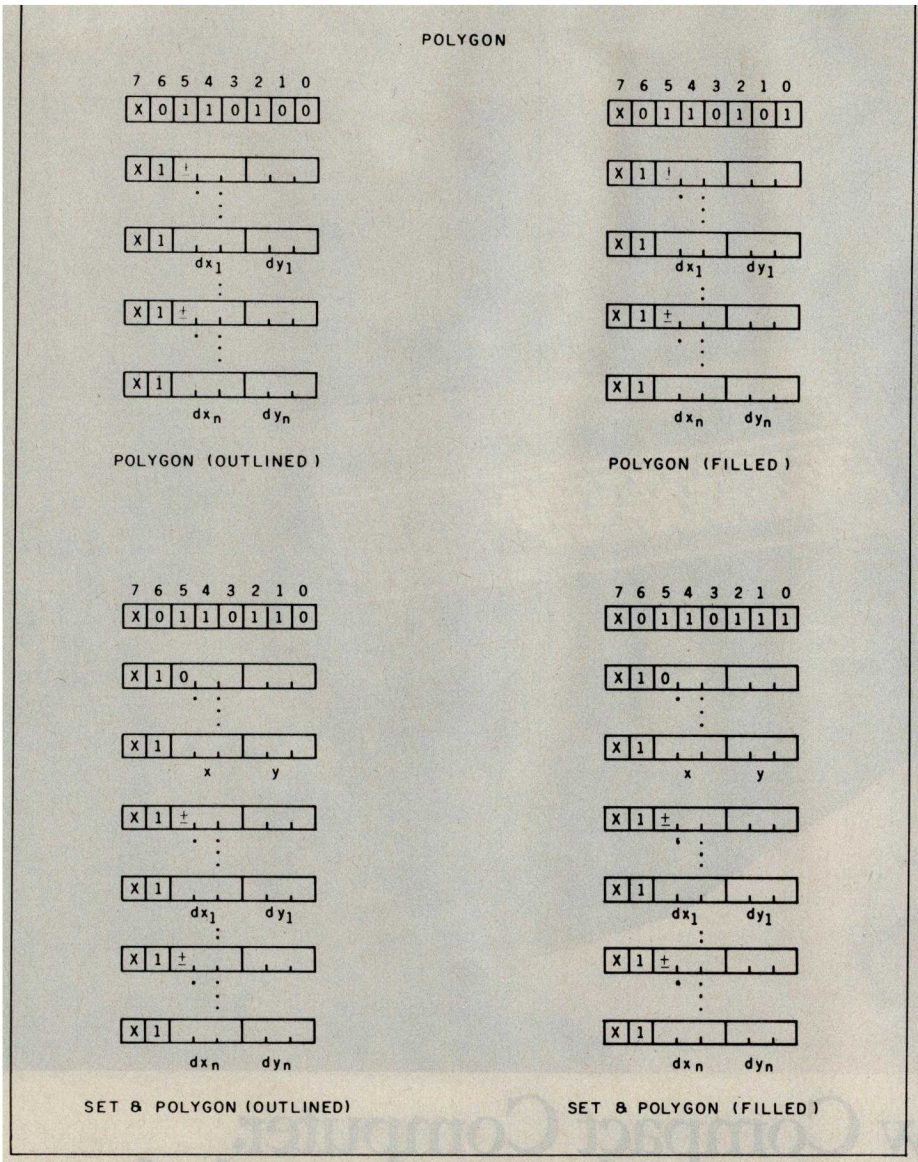


Figure 14: *The Polygon instructions. Any number of points can be used to define the polygon.*

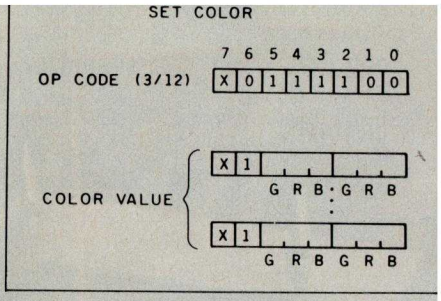


Figure 15: *The Set Color instruction. This instruction defines the color with which all succeeding characters or graphics designs will be drawn.*

what may be "under" the object. Pictures can be built up in the same manner that kids create pictures using construction paper.

In the example frame, the largest polygon is the grass (lines 10–31). When the house is drawn on top of the grass, a piece of the polygon is covered. Likewise, when the road is drawn (lines 220–239), more of the grass is covered. If the grass had been drawn last, part of the house and the entire road would not be seen.

The polygon that is used to fill the center of the cloud (lines 151–160) can be derived directly from the arcs that surround it. As shown in figure 12, the (dx,dy) values for the polygon end up being the sum of the (dx,dy) values for the three points that describe the arc.

Other PDIs

Several other PDIs are available in NAPLPS. Some of them allow compressed encoding of high-resolution images and detailed line drawings. PDIs are included that allow "logical" areas on the screen to be specified for user input. Timed waits and blinking capabilities are also part of NAPLPS, but will not be discussed here.

Color Control

Color control in NAPLPS ranges from primitive, static color definitions to exotic color mapping and animation. Here I shall describe only the primitive color-control capabilities of NAPLPS.

The basic color-control capability of NAPLPS allows a color to be expressed as relative amounts of red, green, and blue. The "resolution" of the color specification can vary just as with coordinates (see figure 15). A display device is expected to display the "closest" color that is available.

For simple display devices, 4 to 6 bits of color specification are usually sufficient to select every available color (unless color maps are available). These color-specification bits are usually encoded in a truncated multivalued-operand byte. The first color specification in the sample frame appears in lines 2 and 3. The Set Color PDI is an op code and is followed by a data byte that specifies three units of blue, zero units of red, and zero units of green. The resulting color of the sky is a "very blue" blue.

When a color is specified, it becomes the "current in-use color." Anything drawn after the Set Color will be drawn in the new color. Note that after the sky is created, the green grass color is specified in lines 6 and 7. If this was not there, the grass would be drawn in blue and would not be visible.

Changing Character Sets

If you have been carefully decoding the information in listing 1, you have

probably come across a few SO and SI codes (octal 016 and 017). These codes are used to indicate a change in the character sets or G-sets that are to be used. In the 7-bit mode of NAPLPS, only one character set can be used at a time. The SO code specifies that the set of PDIs should be used, and the SI code specifies that the Text character set should be used.

You have also probably noticed that the high-order bit of all the codes has not been used. The reason for this of course is that we have been using the 7-bit mode of NAPLPS. If the 8-bit mode were desired, a simple conversion can be made. Each time an SO is found it should be removed, and all bytes following that code should have their high bit set to 1. When an SI is encountered, it should also be removed and the bytes that follow should have a high bit equal to 0. The result would be that all graphics-related codes would be in the form 1XXXXXXX. All text-related codes would have the form 0XXXXXXX.

In the 8-bit mode of NAPLPS, the 14 SI and SO bytes could be removed, which would allow the figure to be stored in only 270 bytes. This may not seem like a big savings, but for large national databases with thousands of frames, every byte counts. There would also be a payoff in transmission time. At 30 characters per second, those 14 bytes might represent almost ½ second, which adds up as a user interacts with a system.

Next Month

In part 3 of this series, I will cover some of the more advanced topics in NAPLPS, including Incremental Lines, Macros, Dynamically Redefinable Character Sets, and Fields.

This series of articles should give the reader a very good overview of this coding system. But as was mentioned last month, anyone seriously interested in working with NAPLPS should obtain a copy of the complete specifications for $18 from X3 Secretariat, CBEMA, 311 First St., NW, Washington, DC 20001, (202) 737-8888.■

NAPLPS: A New Standard for Text and Graphics

Part 3: Advanced Features

NAPLPS can draw irregular lines, compress repeated code segments, define new text characters, and divide the display screen into separate fields.

Jim Fleming
Unir Corporation
Suite 106
5987 East 71st St.
Indianapolis, IN 46220

The North American Presentation-Level-Protocol Syntax (NAPLPS) is a communications standard that can encode both text *and* graphics. The goal of this standard is to facilitate the exchange of information from one machine to another without regard to differences in the individual graphics-handling capabilities of each machine. In fact, in some ways NAPLPS can be viewed as an extension of the American National Standard Code for Information Interchange (ASCII).

Part 1 of this series presented an overview of NAPLPS. In part 2 I described the basic features of NAPLPS with an emphasis on the actual coding. In this part I will concentrate on the more advanced features of NAPLPS, specifically, Incremental Lines, Macros, Dynamically Redefinable Characters, and Fields. This article will present merely an overview of these advanced features. In order to get a complete description of them, you should obtain a copy of the actual NAPLPS document from the Computer and Business Equipment Manufacturers Association (CBEMA), whose address is listed at the end of this article.

About the Author

Jim Fleming is a member of the ANSI X3L2 Committee on Character Sets and Coding. He is an independent consultant specializing in interactive computing systems.

Incremental Lines

As we saw in part 2, line drawings and objects can be encoded by specifying the endpoints of each line using a terminal-independent coordinate system. In general, it takes about 3 bytes to encode each line in a drawing. For objects with only a few sides or a regular shape (i.e., rectangle, arc, etc.), this efficient approach makes it easy to encode lines. But if an object has an irregular shape, the normal encoding method becomes very inefficient.

NAPLPS supports a method of *chain-encoding* irregular edges of shapes. An assumption is made that the irregular edge is composed of a large number of small line segments. Each segment can be encoded using just *2 bits* of information—a substantial reduction over the *3 bytes* needed using the normal method.

The method for chain-encoding line segments with 2 bits is shown in table 1. The first step is to establish a pair of relative coordinates (symbolized as *dx, dy*) that determines the length of the line segments. Multiple 2-bit values are then specified that indicate directions in which to draw the line segments. As shown in table 1, a line segment can be drawn horizontally, vertically, or diagonally.

If a horizontal segment is specified, it is drawn in the direction initially set by the value *dx*. If *dx* is positive, the horizontal segment will be drawn to the right; if *dx* is negative, the segment will be drawn to the left.

If you are following closely, you may be asking how we could get lines drawn to the left if we initially specify *dx* as positive. As shown in table 2, an Escape sequence is provided to reverse the direction of the drawing. Any time a 2-bit value is 00, the next 2 bits are interpreted as an instruction to change direction. The sequence 00 01 indicates that the horizontal direction should be reversed, the sequence 00 10 changes the vertical direction, and the sequence 00 11 reverses both directions.

Two-Bit Incremental Line Instructions

Symbolic Code	Two-Bit Encoding	Result
–	00	Change dx or dy values (see table 2)
X	01	← dx →
Y	10	↑ dy ↓
D	11	↑ dy ↓ ← dx → or

Table 1: *With the Incremental-Line features of NAPLPS, some line segments can be specified with just 2 bits of information. The values dx and dy signify relative coordinates. If dx is negative, the line will be drawn to the left; if dy is negative, the line will be drawn downward. Other lines can be drawn by using the 4-bit direction changing codes shown in table 2.*

Incremental Line Escape Sequences

Symbolic Code	Four-Bit Encoding	Result
– –	00 00	If pen is down, then lift it up. If pen is up, then put it down.
– X	00 01	dx = –dx
– Y	00 10	dy = –dy
– D	00 11	dx = –dx, dy = –dy

Table 2: *Four-bit codes are used to change the direction of the drawing and control the pen position.*

If you are still with me, you may ask what happens when 00 is followed by 00. That sequence indicates that the state of the *pen* should be reversed. If the pen is currently *down*, it will be raised; if the pen is *up*, it will be lowered.

The 2-bit values are encoded in a series of bytes, 6 bits per byte (from left to right in table 3). The pen is assumed to be down at the beginning of a command. The last byte can be padded with a pen reversal if the number of line segments is not even.

Figure 1 shows a chain-encoded outline of the state of Indiana that is composed of 217 short line segments. These 217 segments require 87 bytes of coding in NAPLPS. If I had encoded this drawing using the normal

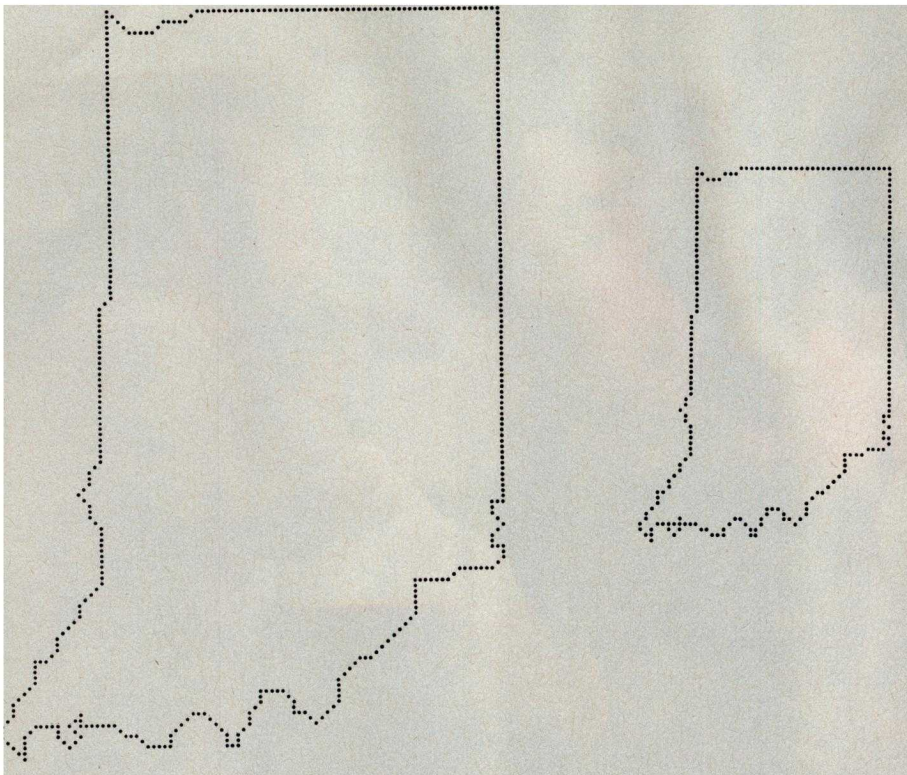
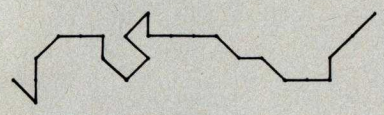


Figure 1: *An example of an incremental-line drawing. The size can be changed simply by changing the values of the relative coordinates dx and dy. This drawing was done on a Diablo 630.*

Sample of Incremental Line Encoding



Symbolic Code	Binary Code	Octal Code
- Y D	X1 00 10 11	113
- Y Y	X1 00 10 10	112
Y D X	X1 10 11 01	155
X - Y	X1 01 00 10	122
Y D -	X1 10 11 00	154
Y D -	X1 10 11 00	154
X D -	X1 01 11 00	134
X D -	X1 01 11 00	134
Y Y X	X1 10 10 01	151
X X D	X1 01 01 11	127
X D X	X1 01 11 01	135
X - Y	X1 01 00 10	122
Y D D	X1 10 11 11	157

Table 3: *An example of incremental-line coding. This is part of the coding used to produce the drawing of Indiana in figure 1.*

method, 651 (217 × 3) bytes would have been needed. (Although these calculations ignore the fact that the northeast corner of the state could be efficiently drawn with normal straight lines, a large amount of memory would be required nonetheless.)

A symbolic representation of part of the chain encoding for figure 1 appears in table 3. It is the coding for part of the southwestern border of Indiana. The corresponding binary and octal values of the encoding are also included. The two outlines were produced from the same encoded data.

The scaling effect achieved in figure 1 was accomplished by changing the size of the *dx* and *dy* values. As shown in figure 2, the Incremental-Line op code (3A hexadecimal or 3/10 using a 16 by 16 column/row representation) and a relative coordinate pair (*dx, dy*) precede the encoded data. This (*dx, dy*) value can be changed to create some interesting effects. For example, it can be scaled so that the resulting drawing is larger or smaller, even though the *same* encoded data is used. Also, the values of *dx* and/or *dy* can be negative, resulting in mirror-image drawings from the same encoded data. The values of *dx* and *dy* can be very different, resulting in a stretching effect in one of the dimensions.

Besides chain-encoded outlined drawings, NAPLPS also supports chain-encoded filled polygons. These solid figures are subject to the same attributes of objects such as rectangles and arcs. These attributes can be used to draw the state of Indiana with a textured interior to help distinguish it from other objects on the screen or page.

Incremental encoding is a powerful technique for compactly representing irregular objects. The efficiency obtained by incremental encoding pays off in data-transmission time and disk storage. Another feature of NAPLPS that improves coding efficiency is the Macro capability.

Macros

Macros provide a means of storing an arbitrary string of NAPLPS codes under one of 96 names. The string can be recalled using the 1-byte name associated with the string.

Macros are useful when a common string of NAPLPS codes is used for a particular application. For example, in a home-banking application the words Balance, Amount, and Payment will appear many times during a session. NAPLPS lets us store these words as macros and recall them by transmitting just 1 byte.

Macros can be used to store both text and graphics. Entire screens of in-

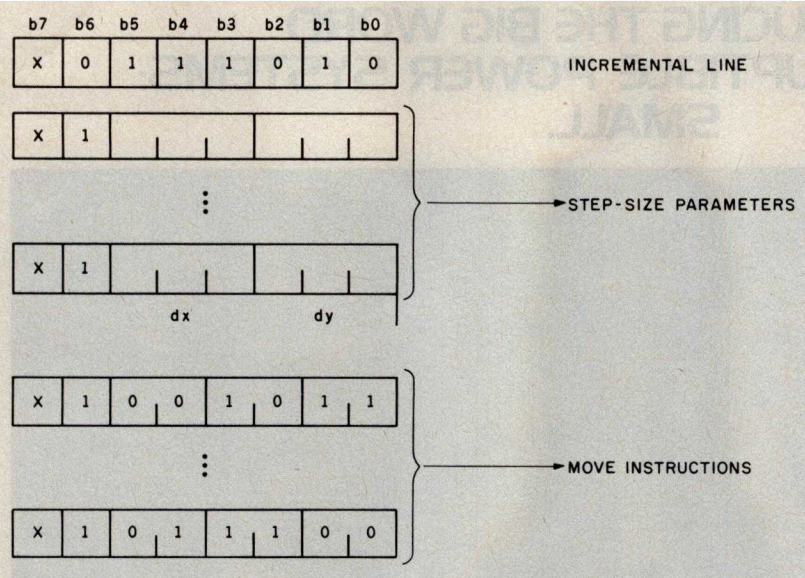


Figure 2: *Structure of the Incremental-Line instructions. The first byte indicates that a series of Incremental-Line instructions will follow. The next series of bytes indicates the size of the relative coordinates dx and dy to whatever resolution is desired. The last series of bytes is composed of the actual 2- and 4-bit drawing instructions.*

```
DEFINE CHARACTER 15
   POSITION (0,.2)
   LINE (0,.3)
   LINE (.15,0)
   LINE (0,-.3)
   ARC (.05,-.05)(.05,.05)
   LINE (0,.3)
END
```

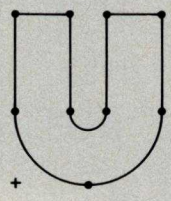


Figure 3: *Using the Dynamically Redefinable Character Set (DRCS) capability of NAPLPS, we can define our own characters.*

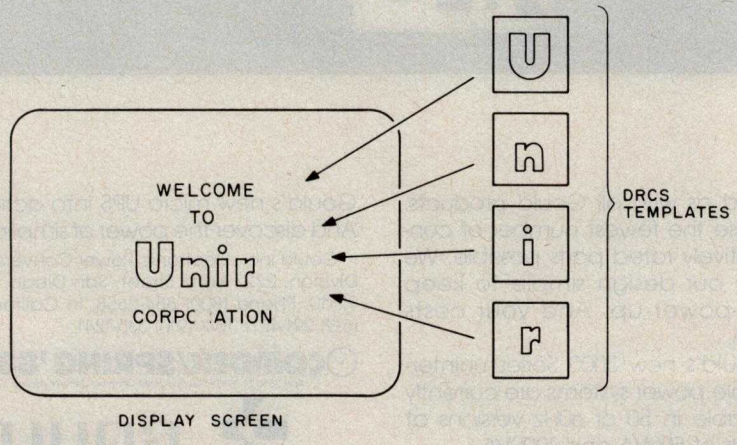


Figure 4: *Once a DRCS set has been defined, its characters can be mapped just like text characters onto the display screen.*

formation can be stored as one macro. In addition, other text and graphics can be intermixed with macros.

In applications like adventure games, each time a new room is entered, a macro can be defined with the picture of the room. If the room is reentered later in the game, the long graphics-and-text description does not have to be retransmitted. The host computer simply has to refer to the macro name and the entire room is drawn.

If the host computer is really smart, it can send down macros for all the rooms that are near the current room you are in. This can be done while you are deciding your next move. The screen will not be changed as these macros are defined. When you finally move to a new room, the host refers to the proper macro that was previously defined, and the user sees an almost instant response.

Macros can also be applied in conjunction with the Incremental-Line op code described earlier. Nothing prevents you from storing just the encoded data as a macro. If this is done, an Incremental-Line op code (3/10) can be sent followed by a (*dx, dy*) coordinate value followed by the macro reference. In the previous example (Indiana), the 87 bytes of encoded data would be retrieved and processed as if they had been sent after the (*dx, dy*) value. Thus, the size of the shape could be changed almost instantaneously.

Part 1 of this series includes an example of macro definition and reference. A NAPLPS code segment for that example is shown in listing 1. As you may recall, a macro (Macro 26) was defined that cleared the screen to blue and put the string "READY:" on the screen in white. Whenever Macro 26 is later referenced, the screen will be cleared and "READY:" will appear.

The amount of storage allocated to macros has been left open as a terminal- and application-dependent parameter. With 96 macro names and the ability to store long variable-length strings, the macro storage required could exceed the internal memory capacity of most personal

Listing 1: *A short segment of NAPLPS code (in octal form) showing how a macro can be defined and referenced. The contents of Macro 26 will be bytes 4 to 21 inclusive.*

```
BYTE    OCTAL   SYMBOLIC        DESCRIPTION
 #      FORM    FORM
-------------------------------------------------------------------
        .
        .
        ***** Define Macro 26 *****
        .
        .
1       033     ESC     Define Macro 26
2       100     DFM
3       072     M26
        .
        .
        .
4       016     SO      Select G1 (Graphics)
        .
        .
        .
5       074     SET     Set color to Blue
6       111     BLU
        .
        .
        .
7       040     RES     Clear Screen to in-use color
8       120
        .
        .
        .
9       074     SET     Set color to white
10      177
        .
        .
        .
11      044     SPA     Set Point Absolute
12      101             }
13      110             } - (x,y) = (.05078125,.25) => (13,64)
14      150             }
        .
        .
        .
15      017     SI      Select G0 (Text)
        .
        .
        .
16      122     R
17      105     E
18      101     A
19      104     D
20      131     Y
21      072     :
        .
        .
        .
22      033     ESC     End Macro Definition
23      105     END
        .
        .
        .
***** Referencing the Macro *****
        .
        .
        .
??      035     SS3     Assume 7 Bit Mode with Macros in G3
??      072     M26     Macro 26
        .
```

computers. There is absolutely no reason why disk storage cannot be used to store macros. When disks are used, a crude form of file transfer begins to emerge.

Macros thus provide a powerful capability of code reduction when using NAPLPS. The important thing to remember about macros is that a simple string replacement is performed. The resulting stream of code (after the macro expansion) is processed as if it had been sent from the host. No special scaling, rotation, or attributes are applied to the macro part of the resulting stream. In fact, in most systems, once the macro is expanded all knowledge is lost that a macro was used.

Dynamically Redefinable Character Sets

The Dynamically Redefinable Character Set (DRCS) capability in NAPLPS is similar to the Macro feature. The DRCS facility allows arbitrary streams of NAPLPS code to be associated with one of 96 names. The 96 DRCS names, however, are completely different from the macro names.

The primary difference between macros and DRCS characters is that when the DRCS image is referenced, it is scaled, rotated, and mapped to the current character field, just like a text character. In other words, a simple code replacement is not performed. Instead, the current environment of attributes and character-field size is used when the DRCS character is requested. Figures 3 and 4 show an application of the DRCS capability.

One of the primary functions of the DRCS is to define new text characters, because all TEXT command attributes (scaling, rotation, spacing, etc.) are applied when the DRCS is requested. Because of this text-character orientation, only two colors can be used when the DRCS is drawn. In other words, the drawing of the DRCS is done with a "spray-paint" technique, as with other characters. The DRCS thus acts only as a stencil.

The DRCS can also be viewed as a simplified form of the computer-graphics idea of a *window* and a *viewport*. When using a DRCS, the

Text continued on page 202

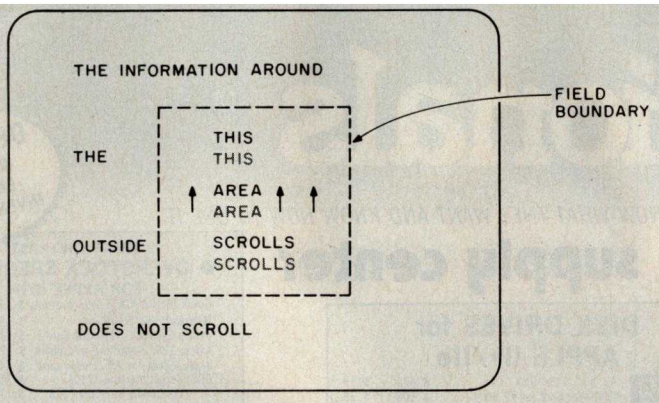


Figure 5: *In NAPLPS, some fields of the display screen can be scrolled; others remain stationary.*

window is completely mapped to the viewport, which is mapped to the character field on the screen. Figure 4 illustrates this mapping. Because the character field can cover an area from one pixel (picture element) to the entire screen, a scaling effect is achieved.

The DRCS capability is another method for reducing the transmission time of data sent from the host. In applications that require many special symbols, the symbols can be defined, given a name, and referenced with only 1 byte—even for a symbol that may require hundreds of bytes of definition. By replacing the hundreds of bytes with a single one, the transmission is compressed to the point that the users think they are connected to the host by a high-speed link.

Two popular methods of handling DRCS in terminals are available. The first method cuts the template as soon as the DRCS is defined. When the DRCS is requested, the previously cut template is retrieved and handled like any other character-definition template. The templates are usually stored on 8 by 8 or 16 by 16 grids internally in the terminal.

The second method for handling DRCS is to store the graphics commands themselves. No template is cut. When a DRCS is requested, the template is cut in a size appropriate for the current character field. The template may be 32 by 32, or even 256 by 256 for full-screen DRCS. Once this template is cut, it is mapped to the display screen using the spray-paint concept.

The primary difference between these two methods appears in the areas of speed and resolution. The first method is good when DRCS characters must be displayed very fast and with minimum resolution. If the DRCS is scaled to a much larger size, however, a crude image results from the precut template. The crude image results from the fact that some of the information may be lost when the cutting is performed, especially if the template is very small.

The second method preserves all the information by maintaining the DRCS definition in its original form. If a large DRCS is requested (i.e., the character field is large), all the original information can be used to draw the character. The disadvantage of this method is that the original definition has to be decoded each time the DRCS is requested. This can be quite time consuming for a DRCS that contains several hundred bytes of definition.

These two methods are not the only ways of handling DRCS. As with most features of NAPLPS, no restriction is placed on the method of implementation. The cleverness of the implementation will determine how much of the original information is transferred to the user. The appropriate time, space, and cost trade-offs have to be made in choosing implementation strategies. These decisions, of course, will help differentiate various companies' products. It would be an extremely boring world if NAPLPS were so restrictive that every terminal looked and performed *exactly* the same.

Fields

The last advanced feature to be discussed here is Fields. Fields are logical rectangular areas defined on the unit screen by the Field command (3/8). These areas are not visible to the user. Only one active field exists although, as will be seen, multiple fields can be defined as long as they do not overlap. Fields are used in a variety of ways in NAPLPS.

A common use of the Field capability is for setting margins. The Field command is used to establish the current active field, which is normally the entire screen. Each time a text character is placed on the screen, an internal cursor position is updated. If the boundaries of the field are exceeded, the cursor is moved to the other edge of the field in a manner compatible with most data terminals. If the cursor moves beyond the top or bottom of the field, a feature called partial screen scrolling comes into play. If scrolling is enabled, only the information inside the field is scrolled. The information around the field is not changed. Figure 5 attempts to show this concept.

Fields are also used to establish an area for high-resolution image display. The Incremental-Point command (3/9) can be used to specify the color information for each point inside a given field. In applications like real-estate listings, these features can be used to combine a photographic-like picture of a house with textual information concerning the same. Figure 6 illustrates a result that can be obtained.

But the most important use of the Field capability is for user input in systems such as videotex. As described in part 1, users are given one or more blank "sheets of paper" on their screen. The Field command is used to define where these sheets are placed.

When these so-called unprotected fields are placed on the screen, the

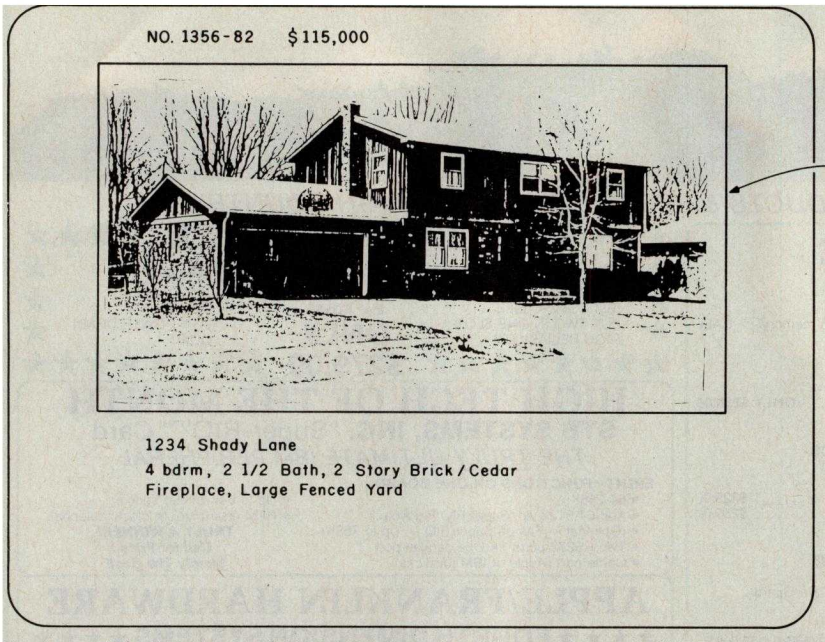


Figure 6: *The Incremental-Point command can be used to specify the color information for each point inside a field. In an application such as real-estate listings, a photographic-quality picture of a house can be included with textual information.*

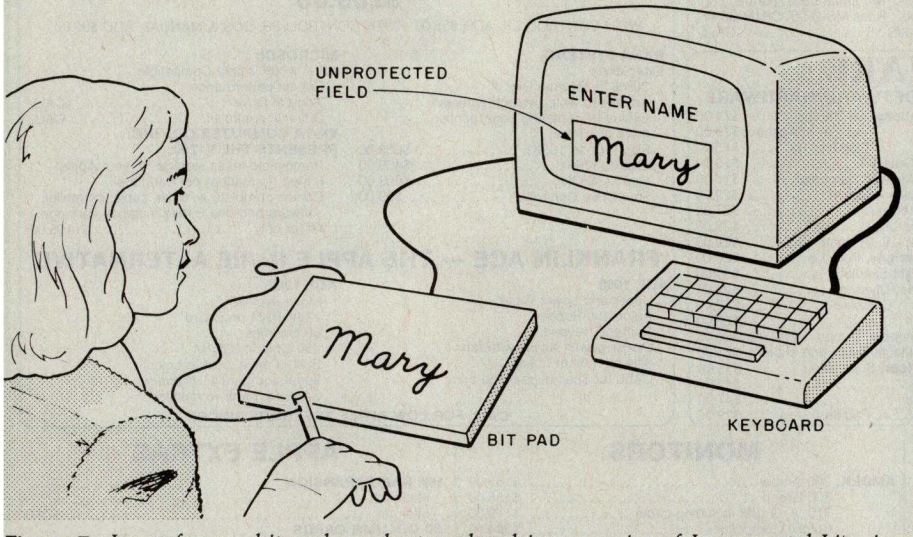


Figure 7: *Input from a bit pad can be translated into a series of Incremental-Line instructions and placed into an unprotected (user) field.*

user is allowed to enter information. The user can also edit the information with any local capabilities provided by the terminal. Once the information is entered, the user sends the contents of the field to the host.

Any legal NAPLPS stream can be placed in a field. In the example shown in figure 7, a person has used a bit pad to put a signature in the field. When the field is eventually sent to the host, the contents of the field would be an Incremental-Line command followed by a stream of data for the signature.

If, on the other hand, the user had *typed* information into the field, the information would be sent to the host as characters from the Primary Character Set.

The information that is sent to the host is identified with the location and dimensions of the field. If multiple fields are set up, the host is sent data only from the fields that are modified by the user.

The Field capability provided in NAPLPS is extremely versatile. In comparing this capability with the fields commonly found on data-processing terminals, you should note two items. First, the fields described here are true two-dimensional areas on the screen. This is quite different from the typical one-and-one-half-dimensional fields found on most terminals. Figure 8 illustrates this difference.

The other item of note is that NAPLPS places no restrictions on the user in regard to what type of information can be placed into a field. The

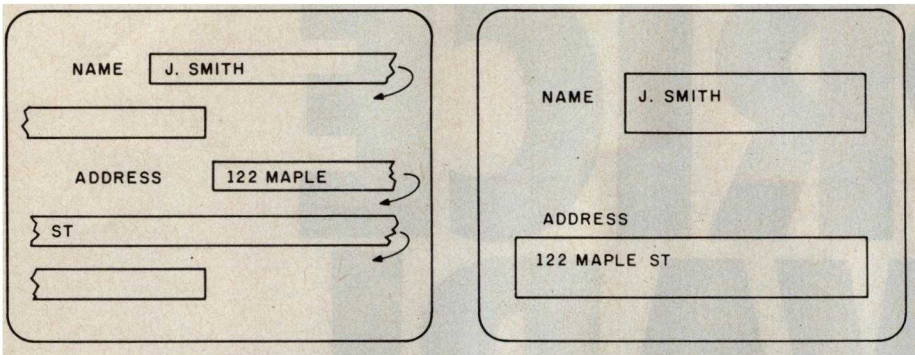


Figure 8: *An example of the one-and-one-half-dimensional fields found in most terminal systems and the true two-dimensional fields possible with NAPLPS.*

options available to the user for entering information into fields become merely terminal-dependent features. Terminal manufacturers are thus able to provide unique input and editing facilities to distinguish their products.

No facilities are available in NAPLPS to specify the type of data that can be entered into a field. Many data-processing terminals now forbid you to enter, for example, alphabetic information into a numeric field. In contrast, the spirit of NAPLPS is to allow free-form user input.

Someday host computers may become smart enough to know that "100," "One hundred," and "1 hundred" all mean the same thing. NAPLPS will be ready to accommodate this capability when it becomes available.

NAPLPS has many other powerful features. It is impossible to cover all the features in the context of this series of articles. As I have indicated in the past, anyone who wants more information about NAPLPS, or who is interested in doing serious work with NAPLPS, should obtain a copy of the specification. Copies are available for $18 from

X3 Secretariat
CBEMA
311 First St., NW
Washington, DC 20001
(202) 737-8888

Next Month

NAPLPS offers us a powerful new communications medium, one that should have a significant impact on the amount and the *type* of information we can exchange among ourselves. Next month, I will describe some of the advanced color capabilities and speculate on the ways in which NAPLPS will affect the personal-computer user. ■

NAPLPS: A New Standard for Text and Graphics

Part 4: More Advanced Features and Conclusions

A standard way to encode color mapping and animation, closing with some predictions on how NAPLPS will be used by personal computers.

Jim Fleming
Unir Corporation
Suite 106
5987 East 71st St.
Indianapolis, IN 46220

This is the fourth and final part of this series of articles on NAPLPS, the North American Presentation-Level-Protocol Syntax, a new communications standard for encoding both textual and graphics information. With this standard, graphics information can be sent from computer to computer, or from peripheral to peripheral, with little regard to inherent differences in the display capabilities of the various machines available. Such a standard will be particularly important for proposed mass-market datacommunications systems such as videotex.

The first part of the series presented an overview of NAPLPS. The second part gave a detailed analysis of its basic features. And last month, some of the advanced features of NAPLPS were discussed. This month I will continue that discussion and conclude with some predictions about the future of NAPLPS.

About the Author

Jim Fleming is a member of the ANSI X3L2 Committee on Character Sets and Coding. He is an independent consultant specializing in interactive computing systems.

These predictions are based on the premise that personal computing has *not* yet reached the majority of people in the world. For that to happen, personal computers must be very easy to use. Those readers who may be dreaming about the day when everyone will be assembling S-100 kits and toggling in bootstrap programs should probably reassess their thinking.

INDEX	COLOR		
	RED	GREEN	BLUE
0	0.2	0.3	0.0
4	0.0	0.0	0.0
5	0.5	0.8	0.8
6	0.0	0.8	0.8
25	1.0	1.0	1.0
26	0.5	0.5	0.5
27	0.6	0.0	1.0
31	0.0	1.0	0.0

Figure 1: *A typical color table used with color modes 1 and 2 in NAPLPS. Each index value corresponds to a specific color. Note that the index values are arbitrary and that no index value appears more than once. Each color is represented as a mixture of the primary colors red, green, and blue.*

Advanced Color Capabilities

As was mentioned in part 1, NAPLPS supports a variety of color modes. The most primitive one (mode 0) is quite portable and will be used in the majority of applications. As was previously described, colors are specified in color mode 0 by indicating the relative amounts of red, green, and blue that should be mixed to form the desired color.

Color modes 1 and 2, however, use a technique called *color maps* or *color tables*. As shown in figure 1, a color table is a set of indexes and their corresponding colors. These are tied together using the Set Color and Select Color commands. As can be seen in the figure, a given mixture of primary colors can be used more than once, and all the indexes do not have to be used. Note that the index value does *not* have any direct relationship with a hardware *drawing value* at this point.

In order to make an entry in the color table, you must first use the Select Color command to specify a

certain index (see figure 2). This index value is encoded in the byte or bytes following the command and usually ranges from 0 to 63, although indexes as high as 16,777,215 can be encoded. After an index is chosen, you can then use the Set Color command to specify the mixture of red, green, and blue values that should be associated with the index.

When you want to select a color for drawing in color modes 1 and 2, you must specify the index for that color—not the actual color. The primary difference between modes 1 and 2 is that mode 2 allows you to specify a background color for text characters.

As shown in figure 3, when a picture is drawn into display memory, a drawing value is allocated to each index. This value is written into the display memory. The color information currently associated with each index is put into a hardware register associated with each drawing value, and anything drawn with that display-memory value will have that color. Note that a display value is allocated to an index only when drawing occurs. Also, each display value is allocated to only one index.

In order to create blinking and animation effects, the color associated with an index can be changed using the Select/Set sequence de-

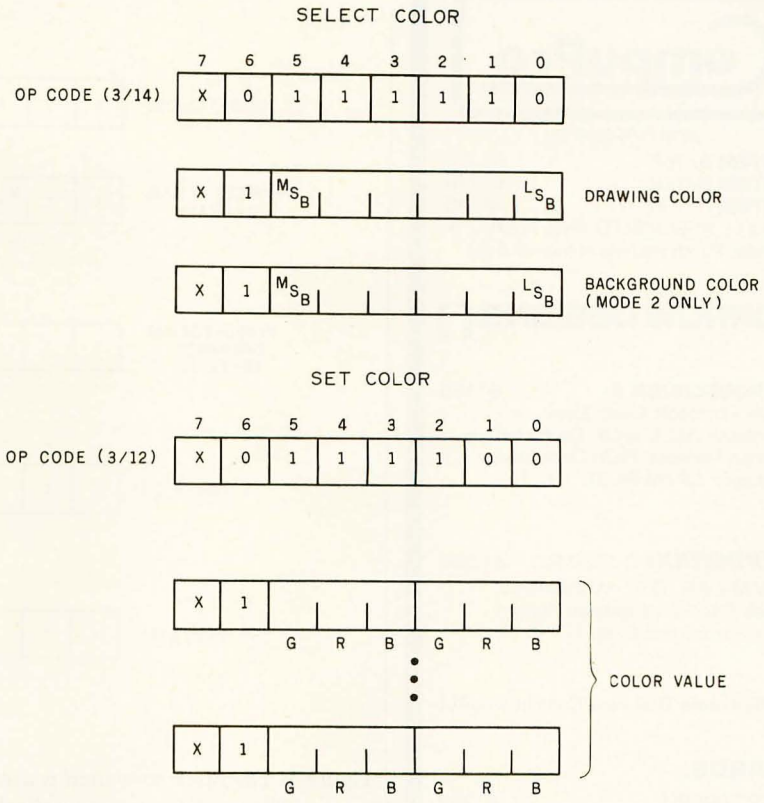


Figure 2: *The Select Color and Set Color commands used to specify a color-table index value and a red, green, and blue mixture, respectively. (The codes 3/14 and 3/12 are an alternate way of indicating the hexadecimal numbers 3E and 3C.) In color mode 2, two indexes can be specified in the Select Color command if a background color for text is desired. The normal sequence used to load the color table is to first select an index and then set a color for that index. These actions cause a relationship to be established between an index value and a color. After a table is set up, each set color can be used by using the Select command. Also, the color associated with each index value can be changed by using another Set Color command.*

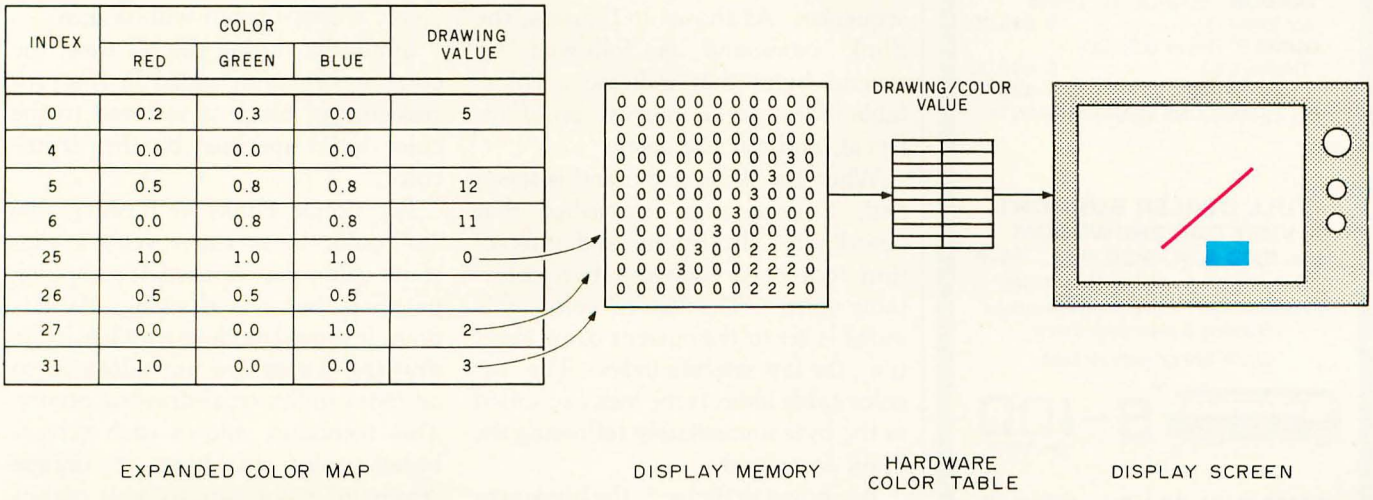


Figure 3: *A typical color-table implementation. Note that special color-map hardware is required. The drawing value is the one used when drawing occurs in the display memory. It is an index into the hardware-based color map.*

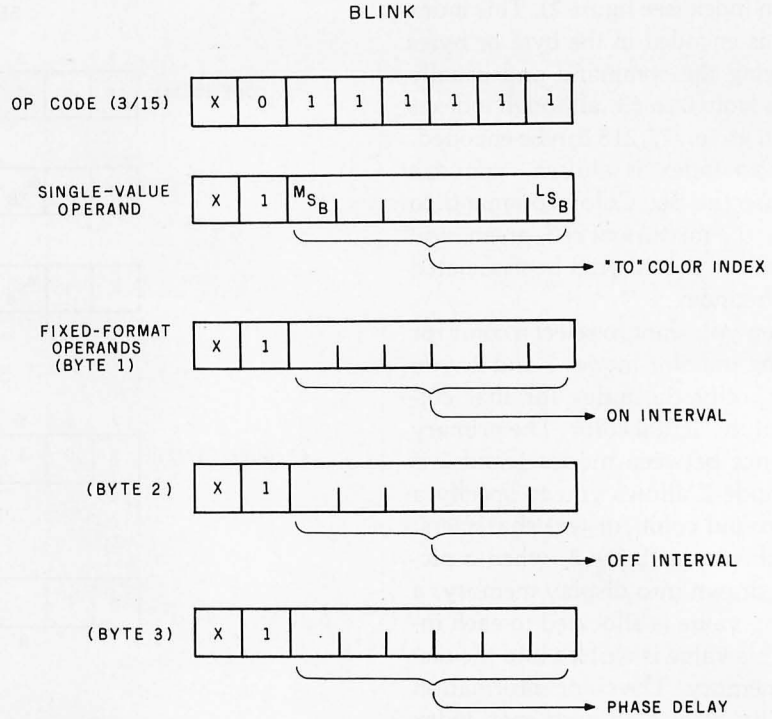


Figure 4: *The Blink command is used to establish one or more asynchronous blink processes. These processes automatically modify the contents of the color map. In short, the Blink command causes the color of the current color index to be temporarily changed to that of the "to" color index. The on, off, and phase-delay times are specified in 1/10-second resolution, which allows this system to be compatible with both North American (60 frames per second) and European (50 frames per second) television systems. These times are used to control how often the color map is modified.*

scribed above. This change will cause everything on the display screen that has been drawn with that value to change color.

Color-Table Animation

The Blink command is used to set up automatic color-table animation sequences. As shown in figure 4, the Blink command is followed by several bytes that indicate a color-table index, an *on* interval, an *off* interval, and a phase delay.

When the Blink command is specified, a *process* is established that coordinates the timing and interaction that occur between two color-table entries. The "from" color-table index is set to the current color index (i.e., the last selected index). The "to" color-table index is the index specified in the byte immediately following the Blink command.

As shown in figure 5, the blink processes are independent asynchronous activities that copy values from one entry in the color table to another. During the on interval, the color corresponding to the index specified by the "from" color is saved in an area of memory called the *process-control block*. Then the color corresponding to the index specified by the "to" index is copied to the "from" index. If the new color is different than the old color, a visual result will be seen.

Similarly, during the off time, the color information saved in the process-control block is restored to the color index specified by the "from" color.

For simple blinks or flashing, the "to" color index can specify a constant color that is used for copying purposes but not drawing. At this point it should be clear why hardware drawing values are not allocated to an index until actual drawing occurs. This technique allows each screen-based color to have a unique "from/to" color pair without requiring that colors be shared.

If you are following carefully, you will note that, at the beginning of the

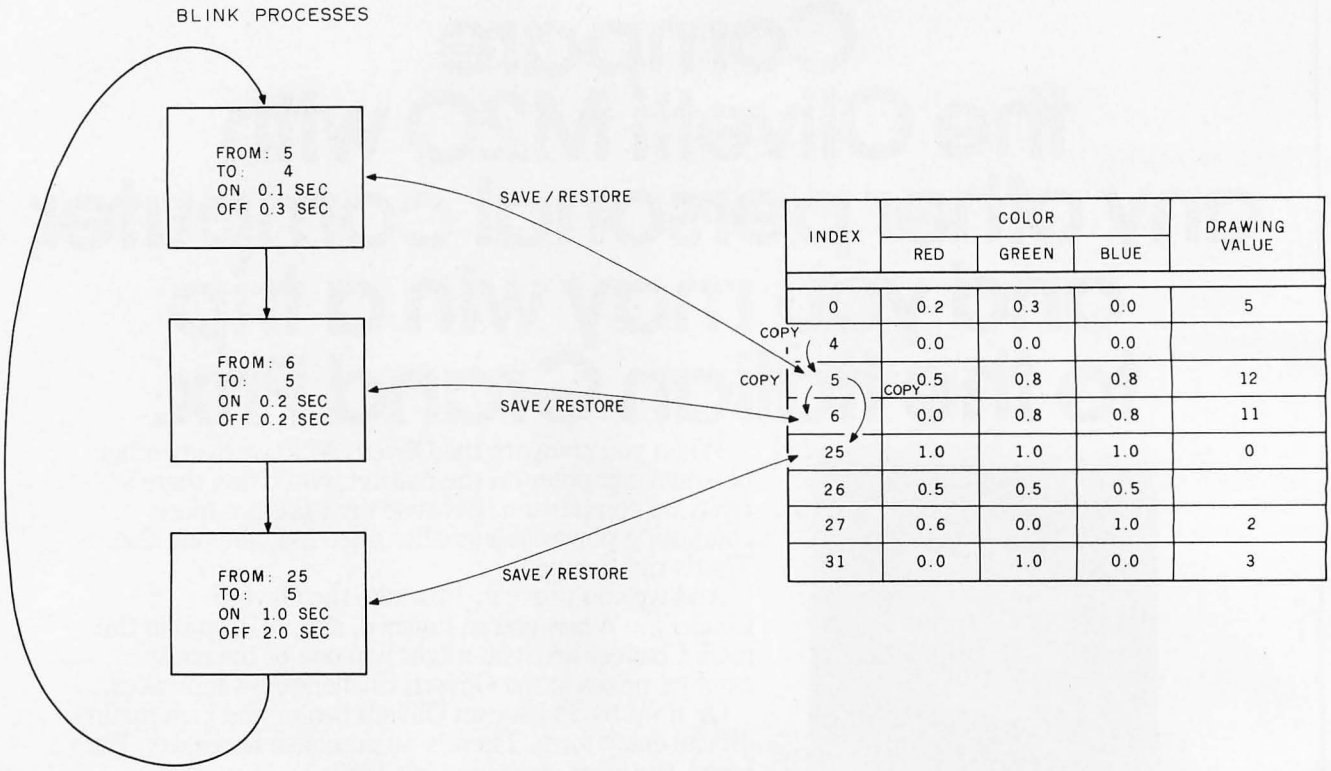


Figure 5: *An example of three blink processes used together. The first process saves the contents of the color map at index 5 and then copies the contents of index 4 into index 5. After 0.1 second, the original contents of index 5 are restored for 0.3 second. The second process saves the contents of the map at index 6 and copies the contents of index 5 into index 6. Note that the second process can copy the contents of index 5 during the time it has been changed by process 1, causing a variety of effects. The third process, which runs very slowly, saves the contents of index 25 and also copies the contents of index 5 into index 25. When changes are made to the colors in indexes 5, 6, and 25, the corresponding hardware color-map registers 12, 11, and 0 are changed, causing visual changes to the screen.*

on time, the color in the "to" color index is copied to the "from" color index. This copy is made independently of any blink activity that may be set for the "to" color. Thus, this copying could occur during a time when the "to" color has been changed by another blink process. The result is that multiple blink processes can be set up to allow colors to ripple around the color table in regular and irregular patterns. These patterns can be used to produce dramatic animation effects under full control of the terminal and without the need for host interaction.

These animation effects, plus the other features described in the previous articles, should establish NAPLPS as the most extensive information-exchange protocol available. As time goes on, it is expected that NAPLPS will replace ASCII (American National Standard Code for Information Interchange) as *the* standard for information interchange. As this occurs, almost every area of computing will be affected. In order to prepare for this impact, we need to look into the future to see how NAPLPS will help shape the world.

Predictions and Conclusions

If one sits back and attempts to predict the future of personal computing and information exchange, some obvious predictions come to mind:

1. Integrated text and graphics will be essential in all visual information exchange.
2. Personal computers must be designed to be so easy to operate that a casual user can begin doing useful things immediately.
3. A personal computer will be used as a *link* to the rest of the world rather than a *diversion* from it.
4. People's efficiency will be increased by allowing concurrent activities.
5. The average personal computer user will be more of a *consumer* than a *producer*.

It should be noted that I have said nothing about the size of memory chips or the density of disk drives. These predictions involve only functionality and the user's view of computing capability. I am a firm believer that the consumer will be the ultimate decision maker when the fate of the personal computing field is decided. These users will make these decisions based on what they see, hear, and touch. They will not make their decisions based on how many chips are in a box or whether interrupts are enabled during DMA disk transfers.

Using these predictions and a little imagination, we can hypothesize what the ultimate personal computer will be like. Figure 6 illustrates my view of such a machine. Four *functional servers* are clustered around a central switching, control, and com-

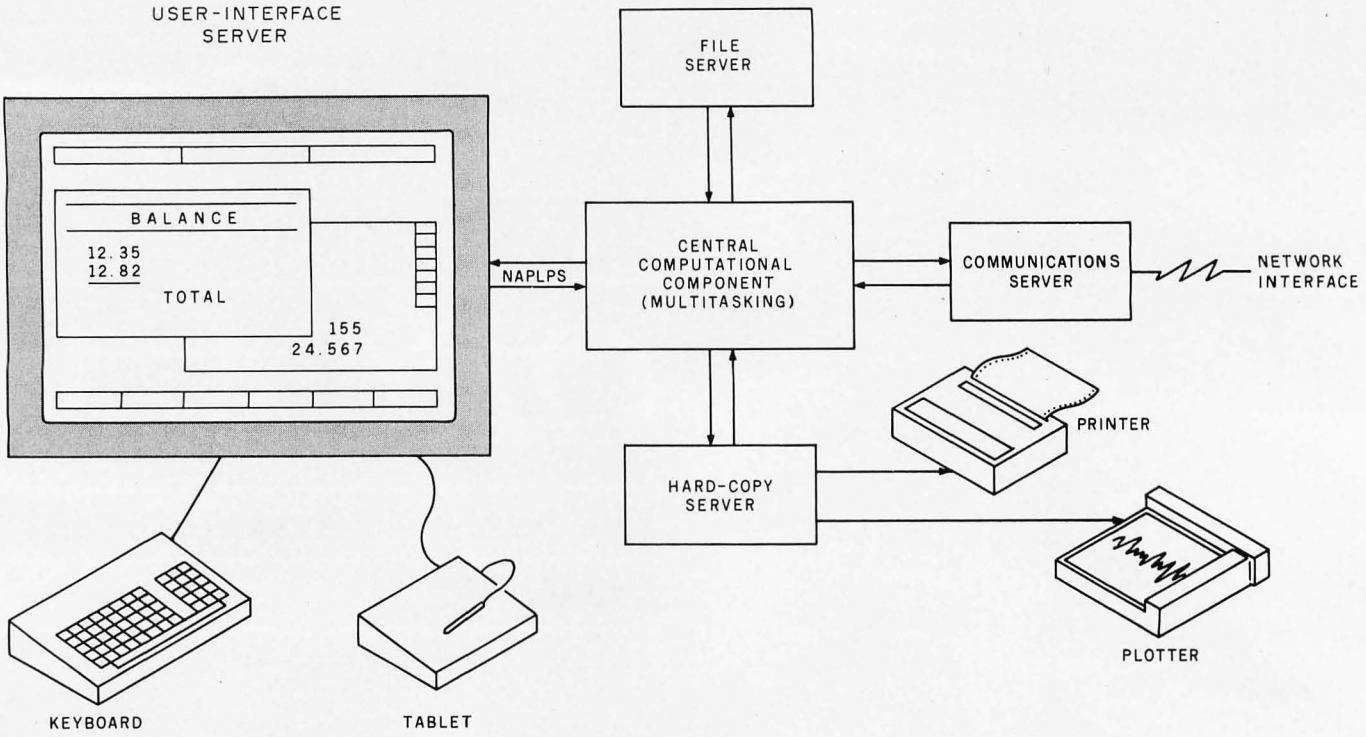


Figure 6: *The ultimate distributed personal computer system for the serious user. Each of the components is an independent entity, each with the power of a typical personal computer available today. The diagram illustrates functionality and does not necessarily imply physical partitioning. NAPLPS is the language used between all components, especially between the User Interface and the Central Computational Component. The User Interface may be able to run a variety of local screen-based text and graphics editors without informing the Central Computational Component.*

putational unit. These four functional servers are *not* meant to indicate peripherals hanging on a microcomputer. In fact, they may instead be integrated into one unit. To put things into perspective, I imagine that each of the functional servers would have the complexity equal to or greater than that of an IBM Personal Computer.

The Hard-Copy, Communications, and File Servers are straightforward. Ideally, the Hard-Copy Server would support integrated text and graphics in high-resolution black and white. Color hard copy would be desirable, but I could probably live without it. The Communications Server would provide all links to off-site services through modems and local networks. The File Server would provide the typical storage and retrieval functions. Features such as automatic redundancy and backup might be transparent to the user. Database-management capabilities could be built into the File Server, which would be responsible for organizing the database in the optimum way based on the user's answers to certain queries.

The User-Interface Server would be an extremely high resolution color-graphics display with a variety of user-input devices. All input editing would be handled by this server. The user should be able to enter text and graphics with equal ease. The ability to point to objects on the screen should be available from any of the input devices. And the User-Interface Server should be able to support multiple *windows*, representing various concurrent activities currently in progress.

The Central Computational Component would be a multitasking system with the computational capability similar to most multiuser timesharing systems. A process (or task) would be active for each window in use in the User-Interface Server. Additional processes could be created to perform tasks for the user.

The Central Computational Component would also be responsible for coordinating all server-to-server interactions. It would also act on the user's behalf if some attention is needed and the user is busy. For example, electronic mail received via the Communications Server would be stored in the File Server without the user becoming involved. The user would be able to retrieve the mail at a later time.

Given this model of an ultimate personal computer, it becomes useful to begin charting an evolutionary path from our current position to this ultimate system. It is obvious in figure 6 that the User Interface must be present in the system before any other component. It is not obvious that the user needs any other component as long as the User Interface has access to a Central Computational Component.

The first step along the evolutionary path is to give the user a User-Interface Server. From that day forward, the user's view of the system begins to form. (The old adage "love at first sight" can be applied to com-

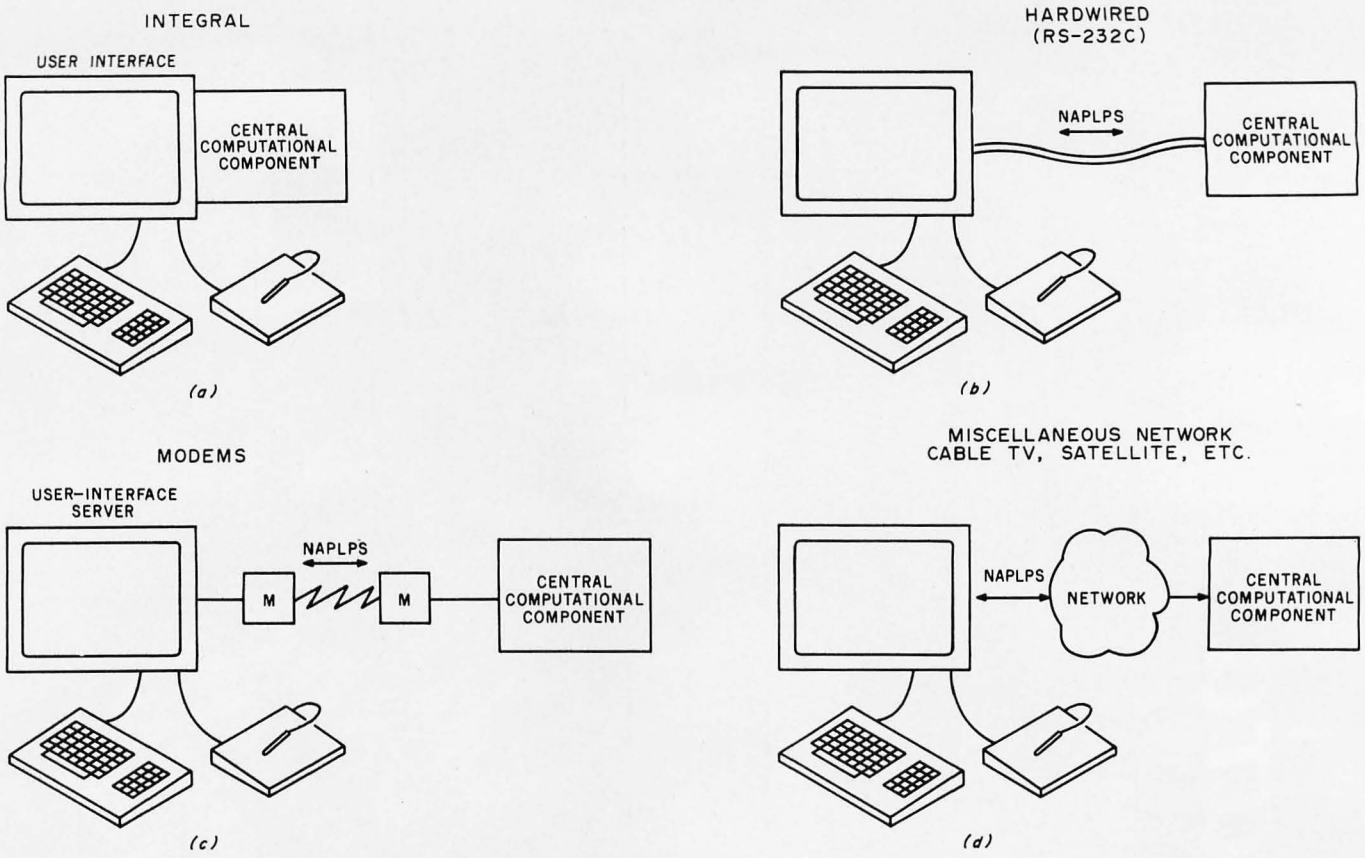


Figure 7: *The User-Interface Server can be connected to the Central Computational Component in a variety of ways. A modem or a network arrangement allows a person to start using a User-Interface Server long before the Central Computational Component is needed.*

puters as well as people.)

Because of this user view that is established, one is driven to give the user as much capability from day one as is economically feasible, even at the expense of not providing other components in the system. This strategy is compatible with the fact that a user will judge a system primarily from interaction with the User-Interface Server.

Once a user is given a User-Interface Server, another serious situation develops: the user will not want to give it up. This implies that when the user eventually obtains a Central Computational Component, he or she will expect the User-Interface Server to work with the new component.

What this all comes down to is that the user should be given the ultimate text/graphics terminal before any other component. The capabilities of this terminal should be standardized, and everything but the kitchen sink had better be available, because the capabilities of that terminal will help shape all future services. And the interface to that terminal must be clearly defined *before* any terminals are given out.

This is where NAPLPS becomes involved—as the text/graphics terminal-interface protocol for the User-Interface Server. As shown in figure 7, a NAPLPS terminal can be connected via a variety of mechanisms to a Central Computational Component. From the first day of use, the user sees a certain set of capabilities and begins to get used to the local editing features available in the terminal. If a user is connected to a host via a modem, all computation and information are obtained from the remote host. The spectacular text/graphics displays are there from the beginning and are not something the user must dream of having.

From the host computer's point of view, the user is now a constant, a known entity with a standard interface. Services can be developed with the assumption that the user will not be a moving target. This stability gives the developers of the host system more incentive to develop more services. More services attract more users, and more users of course mean more money, etc., etc.

The user begins to consume the prepackaged information and services. Most users may not want to become programmers or system administrators; they merely want to accomplish a task and get on with the rest of their lives.

At some point in life, however, a user may become more sophisticated and require a dedicated processor. A Central Computational Component

can be purchased by the user and added to the original User-Interface Server. Many of the services that had been available on the remote host will probably be available on the dedicated processor.

Additional servers could be added to the Central Computational Component as the user desires. One interesting thing to note is that the display capabilities of NAPLPS are such that the Hard-Copy Server can use the same protocol. Also, because the File Server will certainly be able to store NAPLPS, we see that a complete, integrated personal computing system begins to emerge. NAPLPS becomes the common language in the system for information interchange.

The addition of the Communications Server is the mechanism by which a user and personal computer become an entity on a local-area network. Keep in mind that the addition of the Communications Server allows a far more sophisticated system than when the user was *remoted* to a host via an unspecified network. The remoting was done to create the link between the User-Interface Server and the Central Computational Component. The Communications Server was not involved. Figure 7 illustrates this difference.

If I have not lost you completely and you are good at reading between the lines, it should be clear that NAPLPS is a small part of a large, integrated personal computing system. Many of the parts of that system still need to be specified and implemented. Luckily, the part of most concern to users is quite mature and currently available in the marketplace. I predict that in three to four years systems similar to the one I have described will be commonplace in personal computing.

Those of you who are familiar with the various videotex systems that have been proposed may realize that my ultimate personal computer would be fairly compatible with such systems. In most videotex systems, a network of inexpensive home terminals with enhanced graphics capabilities are connected to a host computer. Ideally, the manufacturers of these terminals will design them so that at some later date the user can add the above-mentioned servers.

As I said at the outset of this article, personal computing has not yet reached the majority of people in the world—but someday it will. If you are currently a personal computer user, you should feel honored to be among the pioneers. As the personal computing field begins to evolve into a mass-market, consumer-oriented, network-based information system, do not be surprised if you begin to feel like the odd man out. Also, do not be surprised if one day your neighbors invite you over to see their new computer that costs one fourth to one tenth the price of yours and has access to thousands of services via networking. And do not get upset when you find out that your neighbors cannot tell you which processor their computer uses or what transmission rates their modem supports. Instead, just feel good about the fact that you know a little bit about the language their computer uses to talk to the rest of the world. ■