File No. S360-25
Form C27-6932-2

OS

IBM Systems Reference Library

IBM System/360 Operating System
Graphic Programming Services for FORTRAN IV

Program Number 360S-LM-537

This publication describes how a FORTRAN programmer can write graphic programs for use with the IBM 2250 Display Unit in association with the IBM System/360 Operating System. It also describes how the graphic programming services for FORTRAN may be used in a program written in Assembler Language.

The graphic programming services for FORTRAN IV consist of subroutines and functions for displaying characters or graphic forms on the 2250 screen and for controlling communication between the program and the 2250 operator. The subroutines may be called from a program written in the E, G, or H level of FORTRAN IV, or from a program written in Assembler Language. They are not an extension of the FORTRAN IV language, but may be used in conjunction with it.

It is assumed that the FORTRAN user of this publication is experienced in the IBM System/360 Operating System FORTRAN IV language. It is assumed that the Assembler Language user of this publication is experienced in both FORTRAN IV and Assembler Language.

PREFACE

This publication describes subroutines and functions that can be called from a FORTRAN program to cause displays of characters and graphic data to be produced on the screen of an IBM 2250 Display Unit attached to an IBM System/360 Computing System. The displays may consist of charts, circles, arcs, rectangles, or numerous other configurations.

This publication is divided into five major sections, a series of appendixes, and an index.

The first section familiarizes the reader with the graphic programming services and the 2250. It also discusses the format used in the detailed descriptions of each subroutine and function described in this manual.

The second section presents an overall view of how the graphic programming services may be used to create, modify, and terminate a display. It also defines terminology used in the publication.

The third section provides detailed descriptions of all the graphic subroutines, except attention related and specialized subroutines.

The fourth section describes subroutines that enable two-way communication between the 2250 operator and the user's program, and provides an overall view of how these subroutines (called attention related subroutines) may be used.

Third Edition (May 1968)

This publication corresponds to Release 16 of the operating system. It is a major revision of, and obsoletes, IBM System/360 Operating System Graphic Programming Services for FORTRAN IV, Form C27-6932-1. Major changes have been made to include descriptions of a new routine called Specify Link or Load Status (SPEC), of a change in the use of the alpnameric keyboard CANCEL key, and of the use of programmed function key C.

Changes to the text are indicated by a vertical line to the left of the change; revised illustrations and tables are denoted by the symbol • to the left of the caption.

Significant changes or additions to the specifications contained in this publication are continually being made. When using this publication in connection with the operation of IBM equipment, check the latest SRL Newsletter for revisions or contact the local IBM branch office.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for reader's comments appears at the back of this publication. Address any additional comments concerning the contents of this publication to: IBM Corporation, Programming Publications, Department 637 Neighborhood Road, Kingston, New York 12401

The final section describes functions that allow the programmer to check whether a particular subroutine was able to perform the desired operation on the basis of the data supplied to it, and the type of errors (if any) that were encountered.

The appendixes consist of a sample program and descriptions of specialized subroutines. Included are discussions of the use of the subroutines and functions in an assembler language program, the displaying of characters by means of lines (called strokes), and the direct specification of graphic orders and data. Also included is a listing of the statements necessary to invoke all the GSP subroutines and functions.

Before using this publication, the reader must be familiar with the publication IBM System/360: FORTRAN IV Language, Form C28-6515. He must also be familiar with one of the following publications:

IBM System/360 Component Description: IBM 2250 Display Unit Model 1, Form A27-2701

IBM System/360 Component Description: IBM 2250 Display Unit Model 3; IBM 2840 Display Control Model 2, Form A27-2721.

In addition to the publications listed above, the Assembler Language programmer should be familiar with the publication IBM System/360 Operating System: Graphic Programming Services for IBM 2250 Display Unit, Form C27-6909.

ILLUSTRATIONS

FIGURES

TABLES

The IBM graphic programming services for FORTRAN IV consist of subroutines that enable a FORTRAN programmer to create displays on one or more IBM 2250 Display Units attached to an IBM System/360 Computing System. These displays consist of figures that can be constructed with points, lines, and characters.

The set of subroutines available for use by the FORTRAN programmer is called the graphic subroutine package (GSP). This package is not an extension of the FORTRAN IV language, but is to be used in conjunction with it. The execution of each subroutine is requested by using the CALL statement.

A program that uses GSP includes calls to GSP subroutines in a sequence that causes displays to be produced and that provides two-way communication between the 2250 operator and the program. Such a program is described in detail in the section "Structure of the GSP Graphic Program."

Displays are produced on the basis of control information and data supplied by the programmer in the call to each GSP subroutine. This control information and data define what is to be displayed (e.g., points, lines, or characters) and where it is to be displayed.

Input data can be provided as scalar values or in main storage arrays. As supplied by the programmer, this data is meaningful to GSP but not to the 2250. Therefore, GSP converts this data to a form the 2250 can use.

THE 2250 DISPLAY UNIT

The 2250 Display Unit basically consists of a cathode ray tube on which images are displayed under programmed control, and of optional features that enable data to be entered from the 2250 into the computer. Among these optional features are a light pen, an alphameric keyboard, and a programmed function keyboard.

The screen (12" x 12") is defined by a matrix (1024,1024) of addressable point positions. Each point, or screen location, is specified by a pair of x- and y-coordinates called raster units.

The origin begins at the lower left corner of the screen, and extends horizontally to the right in the x-direction and vertically in the y-direction so that the coordinates at the top right corner represent the maximum boundary of the screen.

A display is created when an electron beam in the 2250 moves over the screen as directed by graphic orders and data being processed in the 2250 buffer. The orders may designate that the beam is to be on (unblanked) or off (blanked) while it is being moved.

Images are only displayed if the beam is moved in the unblanked mode. However, the images fade rapidly and must be continually regenerated to make the display appear steady and stationary. Display regeneration is accomplished by repeating the execution of the orders and data in the 2250 buffer thirty to forty times each second. The actual regeneration rate is a function of the amount of data displayed.

Alphameric characters may be displayed by using the character generator feature or by drawing the desired characters by a series of lines. The

character generator produces a standard set of characters of two sizes (basic and large) and of one orientation (vertical). The dimensions and spacing of characters produced by the character generator are listed in Appendix H.

For a more detailed discussion of the 2250 (including descriptions of the light pen, the alphameric keyboard, and the programmed function keyboard), refer to either of the following publications as appropriate:

- IBM System/360 Component Description: IBM 2250 Display Unit Model 1, Form A27-2701.
- IBM System/360 Component Description: IBM 2250 Display Unit Model 3; IBM 2840 Display Control Model 2, Form A27-2721.

ERROR HANDLING

Errors that occur while the program is communicating with the 2250 are handled automatically by standard IBM error-handling routines. These routines diagnose the errors and apply error-recovery procedures. If an error cannot be corrected, the routines return control to the user's program (if possible) and make available an indication that such an error has occurred.

Specification of invalid arguments in the CALL statement for a GSP subroutine results in GSP making codes available to the programmer that tell what type of error occurred. These codes, called return codes, are described in the section entitled "The Return Codes."

MACHINE AND SYSTEM REQUIREMENTS

GSP may be used with any IBM System/360 Operating System that contains graphic programming services for the 2250 with basic attention handling. For further information, refer to the publication IBM System/360 Operating System: Graphic Programming Services for the IBM 2250 Display Unit, Form C27-6909.

The 2250's attached to the system may be any combination of Models 1 and 3. However, each 2250 must be equipped with absolute vectors and a buffer. The character generator, alphameric keyboard, light pen, programmed function keyboard, and graphic design features are optional.

LANGUAGE COMPATIBILITY

Facilities provided by GSP can be used by programs written in either the E, G, or H level of the FORTRAN IV Language, or in Assembler Language. The body of this publication is directed to the use of GSP by programs written in FORTRAN IV. The use of GSP by programs written in Assembler Language is described in Appendix B.

STORAGE REQUIREMENTS

Refer to the publication IBM System/360 Operating System: Storage Estimates, Form C28-6551, for a description of the amount of storage necessary to process GSP programs.

LINKAGE EDITOR REQUIREMENTS FOR USING GSP IN A FORTRAN PROGRAM

For calls to GSP subroutines to be resolved, the following statement must be included as input to the linkage editor:

```
   INCLUDE   SYSLIB(IHCGSP03)
```

When using a FORTRAN cataloged procedure for compilation, linkage editing, and execution, that statement is included as follows:

```
//LKED.SYSIN   DD   *
               INCLUDE   SYSLIB(IHCGSP03)
/*
```

Additional job control requirements for processing a FORTRAN program are contained in the programmer's guide for the FORTRAN compiler (E, G, or H) being used.

Linkage editor requirements for using GSP in an Assembler Language program are discussed in Appendix B.

SYSTEM GENERATION REQUIREMENTS

GSP is included in the operating system by specifying GSP=INCLUDE in the GRAPHICS system generation macro instruction. The manner in which this is specified is described in the publication IBM System/360 Operating System: System Generation, Form C28-6554.

PROCESSING EFFICIENCY OF GSP PROGRAMS

Copies of most GSP subroutines are located on direct access storage and are brought into main storage and given control the first time their use is requested by the user's program. The length of time a copy of a particular subroutine will remain in main storage has been predefined on the basis of anticipated usage of the subroutines in a typical program.

Loaded Subroutines

Copies of GSP subroutines for which frequent use is anticipated are left in main storage after their processing is complete. Subsequent requests for the use of such a subroutine merely cause control to be passed to the copy of the subroutine left in main storage. All copies of such subroutines are deleted from main storage when the use of GSP is terminated. A copy of a particular subroutine is deleted when its load status is changed to link-to.

This technique eliminates the need for an input/output operation to be performed each time the subroutine is requested, thus reducing program processing time. However, it also increases the total amount of main storage needed for a GSP program because main storage is occupied by a subroutine no matter if the subroutine is or is not in use.

Within this publication, such subroutines are referred to as loaded subroutines.

Link-To Subroutines

Copies of GSP subroutines for which infrequent use is anticipated are removed from main storage when their processing is complete. Subsequent requests for the use of such a subroutine cause a new copy of the subroutine to be brought into main storage at each request and control to be passed to that new copy.

This technique eliminates the need for main storage to be occupied by a subroutine that is not in use, thus reducing total main storage requirements for a GSP program. However, it also requires additional processing time because an input/output operation must be performed each time a subroutine is requested.

Within this publication, such subroutines are referred to as link-to subroutines.

Altering Predefined Link/Load Status

Some GSP subroutines most frequently used in one program may be infrequently used in another program. Therefore, to use the operating system most efficiently, the predefined link-to or load designations may need to be changed to ensure that the subroutines most frequently used are loaded and those less frequently used are linked to. GSP provides a subroutine, called Specify Link or Load Status (SPEC), that allows this change to be easily made.

A description of the SPEC subroutine and a table that identifies the predefined link/load status of each GSP subroutine is included in the section "Initiation and Termination Subroutines." Estimates of the amount of storage required by each GSP subroutine are contained in the publication IBM System/360 Operating System: Storage Estimates, Form C28-6551.

FORMAT OF SUBROUTINE DESCRIPTIONS

In this publication, the detailed description of each GSP subroutine or function is organized as follows:

1. Name--The mnemonic entry name of the subroutine or function, and a phrase explaining the meaning of that mnemonic.

2. Function--A brief summary of what the subroutine or function accomplishes, and its relationship to other GSP subroutines or functions.

3. Format Description--An illustration showing how the statement for invoking the subroutine or function is written. Symbols used in the illustration are discussed below in "Format Illustration Conventions." To conserve space, a reference to the statement number is not shown in the illustration.

4. Argument Descriptions--Detailed information about writing each argument, including any cautions or default conditions applicable to a particular argument.

5. Cautions--A warning of any special restrictions on the use of the subroutine or function.

6. Programming Considerations--Tutorial material describing the use of the subroutine or function and the results it accomplishes.

7. Examples--Wherever necessary, one or more examples showing how the statement that invokes the subroutine or function is written to accomplish desired results. Statement numbers are used in DIMENSION statements for reference purposes only.

Items 1 through 4 are included in all subroutine and function descriptions. Other items are included as appropriate.

FORMAT ILLUSTRATION CONVENTIONS

For consistency and clarity, the following conventions are used in the format illustrations in this publication:

- Upper case (capital) letters, numbers, parentheses, and commas must be written by the programmer exactly as shown. Brackets [ ], braces { }, vertical bars |, and ellipsis ... are never written.

- Lower case (small) letters and words represent arguments for which specific variables or constants must be supplied by the programmer. Unless otherwise noted, all variables supplied in calls to GSP subroutines must be of standard length.

- Brackets denote optional arguments. The items within the brackets may be included or omitted at the programmer's discretion. If an argument is omitted that is not the last argument the programmer wishes to specify in the CALL statement, the null variable must be substituted for the omitted argument. Otherwise, a terminating right parenthesis is sufficient. The null variable is discussed in the detailed description of the Initialize the Graphic Subroutine Package (INGSP) subroutine.

- Braces indicate grouping. One argument from the group must be chosen unless a default option is indicated.

- An ellipsis indicates that the preceding syntactical unit can be written one or more times in succession.

- A vertical bar indicates that either one of two arguments, but not both, must be specified and that the null variable need not be substituted for the omitted argument.

This section provides an overall view of the use of GSP to create, modify, and terminate displays on one or more 2250's, and defines terminology used later in this publication. The subroutines referred to in this section are described in detail later in this publication.

STRUCTURE OF THE GSP GRAPHIC PROGRAM

Preparation of the GSP graphic program requires that the programmer:

1. Establish communication links between the program and GSP (the initiation subroutines);
2. Identify the 2250's on which displays are to be produced (the initiation subroutines);
3. Define one or several graphic data sets (the initiation subroutines);
4. Define the characteristics of the data used to produce the display (the option definition subroutines);
5. Create the graphic orders and data necessary for the display (the image generation subroutines);
6. Cause the display to be produced on the 2250 screen (the image control subroutines);
7. Modify the images making up the display as desired (the image control subroutines);
8. Establish communication between the GSP program and the 2250 operator if desired (the attention related subroutines);
9. Terminate the display and the use of GSP (the termination subroutines).

Terminating the use of GSP when the graphic processing portion of the user's program has been completed is recommended, but is not required.

Establishing Communication Links and Identifying 2250's

To use GSP, the programmer must first establish communication links between his program and GSP. That is, he must inform the operating system that GSP is going to be used. This is accomplished by calling the Initialize the Graphic Subroutine Package (INGSP) subroutine.

The programmer may next specify which subroutines he wants loaded and which subroutines he wants linked to, or may not specify anything thus causing the predefined link/load status conditions to be assumed (see "Processing Efficiency of GSP Programs"). Specification of link/load status is accomplished by the Specify Link or Load Status (SPEC) subroutine. This subroutine may be called anywhere in the user's program.

Once communication links have been established and the link/load status has been defined, the user must identify at least one (but preferably all) of the 2250s on which displays are to be produced by his program. This is accomplished by calling the Initialize a Graphic Device (INDEV) subroutine.

Defining Graphic Data Sets

Once GSP and a 2250 have been initialized, one or several graphic data sets must be created for that 2250.

A graphic data set is a named collection of all the graphic orders and graphic data necessary to display an image on a 2250. It is created by a call to the Initialize a Graphic Data Set (INGDS) subroutine.

Graphic orders are requests to the 2250 to perform operations; they instruct the 2250 whether to plot a point, draw a line, display a character, move the position of the beam, etc. Graphic data follows the orders and provides the information necessary to perform operations requested by the orders. Graphic data represents the programmer's input data after that input data has been transformed by image generation subroutines into configurations that are meaningful to the 2250.

A display may consist of the contents of one or of several graphic data sets. Each graphic data set is represented as a rectangular area on the screen within which an image may be displayed. The boundaries of this rectangle (hence, the boundaries of the graphic data set) do not appear on the screen.

The size and position of each graphic data set are defined in relationship to the fixed size of the screen. An entire graphic data set may or may not lie within the screen boundaries. However, at least one portion of it must lie within those boundaries.

Figure 1 depicts five graphic data sets and their relationship to images of a display. Several of the graphic data sets overlap one another, and one lies partially outside the screen boundaries. The graphic data sets in the figure make up a graphic program that is to display a resistor on the screen. Broken lines represent the boundaries of the graphic data sets. Solid lines and characters within the screen represent images that are produced from the graphic orders and data in each graphic data set. Explanatory statements appear outside the screen boundaries.

The graphic data set is important for data scaling and image scissoring operations. Data scaling involves the conversion of the programmer's input coordinates so that these coordinates conform with the coordinate system of a particular graphic data set or the screen and still remain in relative position to one another. Image scissoring involves truncating portions of a display that extend beyond the graphic data set or screen boundaries.

Defining Characteristics of the Data Used to Produce Displays

Once a graphic data set has been initialized, the following characteristics must be established for it either by the programmer or by default:

- The relation of the graphic data set to the boundaries of the screen;
- The type and form of input data that will be provided;
- The type and form of graphic orders and data that will be produced as output data;
- The size of characters that are to be displayed; and
- How data scaling and image scissoring are to be performed.

The subroutines used for supplying this type of information are called option definition subroutines. Unless they are called by the programmer, default options are assumed.

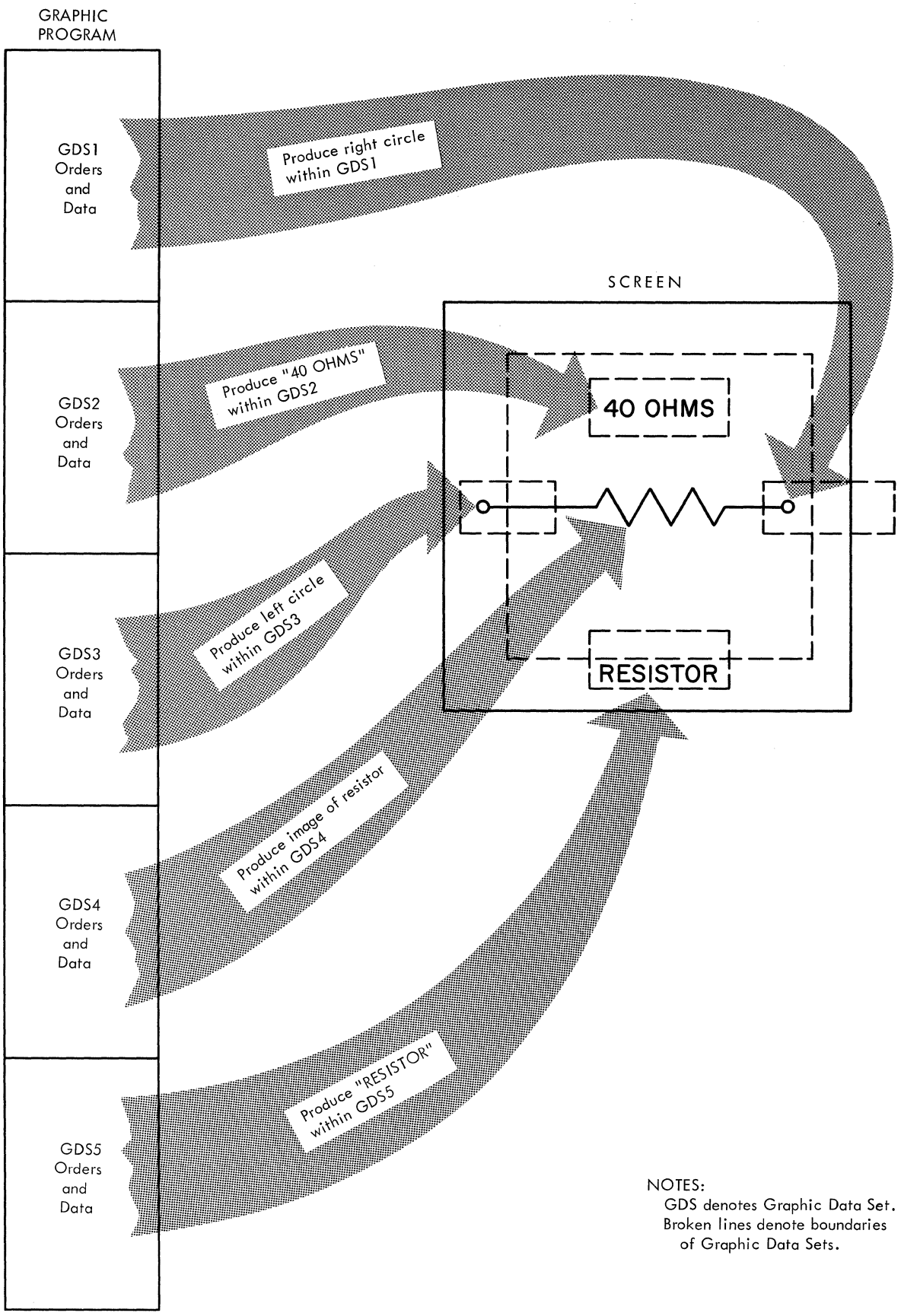


Figure 1. Relationship of Graphic Data Sets to Images

Input data may be main storage arrays of x- and y-coordinates or scalar values that describe images to be displayed. This input data may be of real or integer type, and of absolute or incremental form. Absolute data represents the actual coordinates where an image is to be displayed on the screen. Incremental data represents coordinate values that are displacements from the coordinate values that precede them.

Output data produced by GSP subroutines may be of absolute, incremental, or optimized form. Optimized data represents input data that has been transformed by image generation subroutines into a form that minimizes the amount of buffer storage required to display an element. This form consists of a combination of absolute and incremental data. (Note: Incremental output can only be produced on the 2250 Model 1 with the graphic design feature and on the 2250 Model 3.)

Creating Graphic Orders and Data

After the characteristics of the data have been defined for a graphic data set, the programmer may call the subroutines that create the graphic orders and data necessary for displaying an image associated with that graphic data set. These subroutines, called image generation subroutines, create graphic orders and data for the following operations:

- Moving the beam to a position on the screen where a display is to begin (MVPOS or STPOS subroutine).
- Displaying a single line or a number of continuous lines by means of one CALL statement (the PLINE subroutine).
- Displaying a single point or a number of points by means of one CALL statement (the PPNT subroutine).
- Displaying a single line segment or a number of line segments by means of one CALL statement (the PSGMT subroutine).
- Displaying text by means of the 2250 character generator (the PTEXT subroutine). (Note: Text may also be displayed by means of stroke tables. This is discussed in Appendix C.)
- Placing an end-order-sequence order within a graphic data set (the STEOS subroutine).

When incremental or optimized output is to be produced within a graphic data set, the STPOS subroutine should be the first image generation subroutine called for that graphic data set. This establishes a beam position to which subsequent beam positions can be related.

The set of graphic orders and data created by one call to an image generation subroutine is called an element. Elements are grouped into three types: graphic elements, positioning elements, and text elements. Graphic elements are created by calls to the PLINE, PPNT, and PSGMT subroutines. Positioning elements are created by calls to the MVPOS and STPOS subroutines. Text elements are created by calls to the PTEXT subroutine.

As elements are created by image generation subroutines, they are sequentially stored in a main storage area associated with the graphic data set until such time that they are transferred to the 2250 buffer for execution. The area in which elements are stored is called the graphic data output area. Its size is defined either by default, or by means of arguments in calls for the INDEV and the INGDS subroutines.

Elements may be generated either in include status or omit status. Elements generated in include status are in such a form that the images

associated with them are displayed on the screen. Elements generated in omit status are in such a form that the images associated with them are not displayed on the screen. The programmer specifies that an element is to be placed in include or omit status either by means of an argument in the call to the subroutine that creates the element or by a call to the Place in Include Status (INCL) and Place in Omit Status (OMIT) subroutines.

A number of elements may be grouped into single units defined either as sequences or buffer subroutines. Each sequence or buffer subroutine may be manipulated as a single element, and each element within it may be manipulated independently of the sequence or buffer subroutine itself. A buffer subroutine can only be used with the 2250 Model 3.

A sequence consists of all elements within a graphic data set that are generated between calls to the Begin a Sequence of Elements (BGSEQ) subroutine and a corresponding End a Sequence of Elements (ENSEQ) subroutine. Sequences are described in detail in the section "Grouping Elements Into a Sequence."

A buffer subroutine consists of all elements within a graphic data set that are generated between calls to the Begin a Buffer Subroutine (BGSUB) subroutine and a corresponding End a Buffer Subroutine (ENSUB) subroutine. It represents one copy of all the elements necessary to display an image. This copy is placed at a fixed location in the 2250 buffer and can be repeatedly invoked to display its associated image at several different screen locations. Use of a buffer subroutine eliminates the need for creating separate copies of all elements necessary to produce a particular image each time that image is to be produced. Buffer subroutines are described in detail in the section "Grouping Elements Into a Buffer Subroutine."

Causing An Image to be Displayed

Once one or several elements have been created, the programmer may have the images associated with those elements displayed on the screen at any time. He does so by calling the Execute (EXEC) subroutine.

The EXEC subroutine causes all elements within a particular graphic data set that have been generated since a previous call to the EXEC subroutine for that graphic data set to be transferred to the 2250 buffer and executed. This is true no matter if the elements are in include or omit status. However, as mentioned previously, only images associated with elements in include status are displayed. Images associated with elements in omit status are not displayed. An element can be changed from include status to omit status, and vice versa, at any time.

The EXEC subroutine must be called for any new element (except those created via the update facility described below) to be executed in the buffer and thus have its associated image displayed. This is true even when the elements are within sequences or buffer subroutines.

Modifying the Images Making Up a Display

The programmer can modify images that make up a display. He can replace elements within a graphic data set with new elements, or can eliminate elements from a graphic data set.

GSP provides two methods of modifying an image: updating (changing) and resetting (deleting). Updating is accomplished by the update facility. Resetting is accomplished by the Reset a Graphic Data Set (RESET) subroutine.

The update facility involves calling an image generation subroutine to create an element and substitute it for a previously created element. The RESET subroutine removes a particular element and all elements that follow it from a graphic data set.

To modify elements, the programmer must be able to refer to the elements he wants to change. For this purpose, GSP provides keys and correlation values.

A key is a value, unique within a graphic data set, assigned by GSP to identify a particular element, sequence, or buffer subroutine. Once this value is assigned, the element, sequence, or buffer subroutine is said to be keyed.

To key an item, the programmer specifies an integer variable as the "key" argument in the call that creates the item. The value assigned to this integer variable by GSP is the key. Once keyed, an item can be referred to by any integer variable the value of which is the key.

A correlation value is a value assigned by the programmer that may be used later in the program to identify one or more elements, sequences, or buffer subroutines within a graphic data set. Once this value is assigned, elements, sequences, or buffer subroutines are said to be correlated.

The programmer provides correlation values by means of the "corrval" argument. Correlation values are used to correlate one or several elements, sequences, or buffer subroutines so that these items can be referred to later in the program.

Every key within a graphic data set is unique. However, the same correlation value may be assigned to more than one item in a graphic data set.

Establishing Communication Between the GSP Program and the 2250 Operator

The programmer can include in his GSP program calls to subroutines that facilitate communication between the program and the 2250 operator. These subroutines are called attention related subroutines.

The program communicates with the 2250 operator by placing displays on the screen. The operator communicates with the program by depressing keys on the alphameric keyboard or on the programmed function keyboard, or by touching a part of an existing display with the light pen. Such actions cause attentions.

An attention is an interruption that causes the executing program to change its course at an unpredictable point. When it occurs, GSP makes available to the program information that describes the type of attention that occurred. The attention related subroutines permit this attention information to be requested and obtained. They are described in detail in the section "Communicating With the 2250 Operator."

Terminating the Display and the Use of GSP

Anywhere within the graphic program, the programmer may terminate the use of a graphic data set or a 2250. This is accomplished by calling the Terminate the Use of a Graphic Data Set (TMGDS) and the Terminate the Use of a Graphic Device (TMDEV) subroutines. The TMDEV subroutine also terminates the use of all graphic data sets associated with the 2250 being terminated.

The entire graphic program is terminated by calling the Terminate Use of the Graphic Subroutine Package (TMGSP) subroutine. This subroutine also terminates the use of all graphic data sets and 2250's. It is

recommended, but not required, that this subroutine be called when the graphic processing portion of the user's program has been completed.

It is advisable to terminate the use of a graphic data set when it is no longer needed in the program, rather than waiting until the end of the program to do so. This allows storage occupied by the graphic data set to be used for other purposes as required by the program.

GSP, a 2250, or a graphic data set may be reinitialized at any time by proper calls to the initiation subroutines.

BASIC SUBROUTINES FOR CREATING AND TERMINATING A DISPLAY

Table 1 lists the basic GSP subroutines necessary to generate a display that provides for communication between the program and the 2250 operator. The subroutines are listed in the recommended sequence for their use. Optional subroutines are noted as such.

• Table 1. Basic Subroutines Necessary to Create and Terminate a Display (Part 1 of 3)

Mnemonic	Name of Subroutine	Remarks
INGSP	Initialize the Graphic Subroutine Package	Establishes communication links between the user's program and GSP. Must be the first GSP subroutine called.
SPEC	Specify Link or Load Status	Optional. Defines which subroutines are to be loaded and which are to be linked to. If omitted, predefined conditions are assumed. This subroutine is useful in reducing the time required for dynamic acquisition of GSP subroutines and the amount of main storage needed for a GSP program.
INDEV	Initialize a Graphic Device	Identifies a 2250 on which displays are to be produced.
INGDS	Initialize a Graphic Data Set	Creates a graphic data set for a particular 2250. Must not precede the call to the INDEV subroutine that identifies the 2250 with which the created graphic data set is associated.
SDATM	Set Data Mode	Optional. Defines the type and form of input data for image generation subroutines for a particular graphic data set. If not called, the input data is assumed to be real, absolute.
SGRAM	Set Graphic Mode	Optional. Defines the form of output to be produced by image generation subroutines for a particular graphic data set. If not called, pre-established default options are assumed according to the model of the 2250 being used.

Table 1. Basic Subroutines Necessary to Create and Terminate a Display (Part 2 of 3)

Mnemonic	Name of Subroutine	Remarks
SCHAM	Set Character Mode	Optional. Defines (for a particular graphic data set) the size of characters that are to be displayed, and designates whether these characters are to be protected or unprotected. If not called, basic size, protected characters are assumed.
SGDSL	Set Graphic Data Set Limits	Optional. Defines the boundaries of a graphic data set in relationship to the screen. If not called, the graphic data set boundaries are assumed to coincide with the boundaries of the screen. The graphic data set boundaries are not displayed.
SDATL	Set Data Limits	Optional. Defines the scaling factor to be used for input data associated with a particular graphic data set. If not called, it is assumed that the coordinates of the input data conform with the coordinate system of the graphic data set. In this case, no scaling is performed.
SSCIS	Set Scissoring Option	Optional. Defines the action to be taken if an image exceeds the boundaries of a graphic data set or the screen. If not called, all characters, lines, or points within the screen boundaries will be displayed.
MVPOS STPOS	Move Beam to a Position Set Beam at Absolute Position	These subroutines establish the position on the screen at which an image is to be displayed. The generated elements can be in include or omit status.
PLINE PPNT PSGMT PTEXT	Plot Line(s) Plot Point(s) Plot Line Segment(s) Plot Text	These subroutines create the orders and data necessary for displaying lines, points, line segments, or text. The generated orders can be in include or omit status.
EXEC	Execute	Must be called to transfer graphic orders and data to the buffer and cause associated displays to be produced.
CRATL	Create an Attention Level	Establishes an active attention level for a 2250. (See "Communicating With the 2250 Operator".)

Table 1. Basic Subroutines Necessary to Create and Terminate a Display (Part 3 of 3)

Mnemonic	Name of Subroutine	Remarks
ENATN	Enable Attention Sources	Designates the type of attentions that are to be accepted and the type of attentions that are to be ignored at a particular time.
RQATN	Request Attention Information	Supplies information that designates if an attention has occurred and describes the action that had caused the attention.
TMGDS	Terminate the Use of a Graphic Data Set	Optional. Allows programmer to terminate use of a particular graphic data set at any point in his program.
TMDEV	Terminate the Use of a Graphic Device	Optional. Allows programmer to terminate use of a particular 2250 at any point in his program.
TMGSP	Terminate Use of the Graphic Subroutine Package	Optional. Terminates the use of all graphic data sets and 2250's, and frees all storage used by GSP. Should be called when the graphic processing portion of the user's program has been completed.

This section describes in detail each GSP subroutine (except for attention related, light pen, and specialized subroutines). It begins with a discussion of those arguments used in many of the calling sequences for the subroutines. Attention related and light pen subroutines are described in the section "Communicating With the 2250 Operator." Specialized subroutines are described in Appendixes C through F. Unless otherwise noted, all variables supplied in calls to GSP subroutines must be of standard length.

ARGUMENTS USED BY MANY GSP SUBROUTINES

This section describes those arguments that may be used in many of the calls to GSP subroutines. These arguments are "gdsname", "devicename", "key", "corrval", and "gencode". Except where specification of the argument in the call to that subroutine has a specialized meaning, the description of a subroutine to which any of these arguments are applicable merely lists the argument and refers to this section for a description of it.

gdsname
: is an integer variable the value of which identifies the graphic data set with which the call is to be associated. This value must be the same as was returned as the "gdsname" argument in the call to the INGDS subroutine that created the graphic data set.

devicename
: is an integer variable the value of which identifies the 2250 associated with the call. This value must be the same as was returned as the "devicename" argument in the call to the INDEV subroutine that initialized the 2250.

corrval
: is a constant or variable the value of which identifies the element or sequence associated with the call.

  When an element or sequence is to be created by the call, the programmer is to assign a value to the variable or constant specified as the "corrval" argument. GSP then correlates that value with the element or sequence created by the called subroutine. This correlation value enables future operations to be performed on a specific element or sequence.

  When an element or sequence is to be referred to by the call (e.g.; placed in include status, placed in omit status), the programmer must specify as the "corrval" argument a constant or variable having the same value previously correlated with the element or sequence when that element or sequence was created. GSP then examines that correlation value and performs the desired operation on the element or sequence associated with that value. If more than one item is associated with the same correlation value, the desired operation is performed on the first item that had been correlated with that value.

  For a further discussion of correlation values, refer to the section "Modifying the Images Making Up a Display."

key
is an integer variable the value of which identifies the element or sequence associated with the call.

When an element or sequence is to be created by this call, GSP assigns a value (called the key) to the integer variable. This key enables future operations to be performed on this specific element or sequence.

When an element or sequence is to be referred to by the call (e.g.; placed in include status, placed in omit status), the programmer must specify as the "key" argument an integer variable having the same value previously assigned by GSP as the key when the element or sequence was created. GSP then examines the value of the "key" argument and performs the desired operation on the element or sequence associated with that value.

For a further discussion of keys, refer to the section entitled "Modifying the Images Making Up a Display."

gencode
is an integer constant or integer variable that defines what is to be done with the element, sequence, buffer subroutine, or buffer subroutine linkage generated by the call. The constant or variable must have one of the following values:

1 To place the element, sequence, buffer subroutine, or buffer subroutine linkage in include status;
2 To place the element, sequence, buffer subroutine, or buffer subroutine linkage in omit status; or
3 To substitute the element for an element generated by a previous call (see "The Update Facility").

When the "gencode" argument is explicitly specified with a value of two or three, the "key" or "corrval" argument, or both, must also be supplied. If the "gencode" argument is omitted from the CALL statement, the element, sequence, buffer subroutine, or buffer subroutine linkage is generated in include status.

INITIATION AND TERMINATION SUBROUTINES

This section describes the subroutines that initialize and terminate the use of GSP, 2250s, and graphic data sets. Once the use of GSP, a 2250, or a graphic data set has been terminated, it may be reinitialized by a call to an appropriate initiation subroutine. The initiation and termination subroutines are as follows:

- Initialize the Graphic Subroutine Package (INGSP)
- Initialize a Graphic Device (INDEV)
- Initialize a Graphic Data Set (INGDS)
- Specify Link or Load Status (SPEC)
- Terminate the Use of a Graphic Data Set (TMGDS)
- Terminate the Use of a Graphic Device (TMDEV)
- Terminate Use of the Graphic Subroutine Package (TMGSP)

INGSP--Initialize the Graphic Subroutine Package

The INGSP subroutine establishes communication links between the user's program and GSP, and names the null variable. The null variable is used to identify (1) any optional argument omitted from the calling sequence for a GSP subroutine, (2) when the program is to be abnormally terminated, and (3) if a dump is to be produced. The INGSP subroutine must be the first subroutine called in a graphic program. In effect, it tells the operating system that GSP is going to be used.

General Form
CALL INGSP(gspname,null)

gspname
is an integer variable to which a value is assigned by the INGSP subroutine to identify this call for initialization of GSP. A value of zero is assigned if this subroutine fails to complete the initialization process.

null
is an integer variable used to identify when optional arguments are omitted from the calling sequence for a GSP subroutine, when a program is to be abnormally terminated, and if a dump is to be produced. The absolute value of the variable determines when the program is to be abnormally terminated. The sign of that value determines if an abnormal termination dump is to be produced. The value may be changed by the programmer at any time.

When used to identify omitted optional arguments, the null variable is to be substituted for each intentionally omitted argument. It need not be included to omit optional arguments that would normally appear at the end of a calling sequence. For this purpose, a right parenthesis after the last argument specified is sufficient.

When used to identify when a program is to be abnormally terminated, the absolute value assigned to the null variable is compared with the largest return code value made available by the last subroutine called. If the absolute value of the null variable is less than or equal to the value of the return code, the program is abnormally terminated. If the absolute value of the null variable is greater than the value of the return code, the program is not abnormally terminated. The possible return codes are zero through five. They are described in the section "The Return Codes". If the null variable has a value of zero, a value of one is assumed instead, thus preventing the program from being abnormally terminated after valid execution of a subroutine.

When a program is abnormally terminated because of a return code produced, the sign of the null variable denotes if an abnormal termination dump is to be produced. A negative sign causes a dump to be produced. A positive sign causes no dump to be produced. Abnormal termination dumps are described in the publication IBM System/360 Operating System: Programmer's Guide to Debugging, Form C28-6670.

Table 2 summarizes the information contained in the preceding paragraphs concerning the relationship among the null variable value, abnormal program termination, and abnormal termination dumps.

CAUTION: GSP uses whatever value is in the field identified by the null variable to determine whether the program is to be abnormally ter-

minated. Therefore, if a value is not assigned to the null variable, the program may be abnormally terminated at a point when abnormal termination is not desired.

PROGRAMMING CONSIDERATIONS: For the null variable to be passed to a subprogram which in turn passes it to GSP, it must be defined in one of the following ways:

1. Via the DIMENSION statement. In this case, the subprogram to which the null variable is passed must also identify the argument via the DIMENSION statement.

2. Via the COMMON statement.

3. As a "reference by location" argument (FORTRAN IV G and H only).

The null variable should be given a name (such as IGNORE) that will not be confused with a variable used in computation. To assign a value to a null variable defined via the DIMENSION statement, the null variable must be treated as a subscripted variable. Example: IGNORE(1)=3.

EXAMPLE: Refer to the detailed description of the INGDS subroutine for examples of proper calls to all the initiation subroutines.

Table 2. Relationship Among Null Variable Value, Abnormal Termination, and Abnormal Termination Dumps

When null variable absolute value is	Then
• Greater than return code	• No abnormal termination occurs.
• Less than or equal to return code	• Abnormal termination occurs.
When null variable sign is	**Then**
• Positive	• No dump is produced.
• Negative	• Dump is produced.

INDEV--Initialize a Graphic Device

The INDEV subroutine identifies a 2250 on which displays are to be produced, and establishes communication links between that 2250 and GSP. A separate call to this subroutine is necessary for each 2250 to be used.

This subroutine performs essentially the same functions as the OPEN macro instruction described in the publication IBM System/360 Operating System: Graphic Programming Services for IBM 2250 Display Unit, Form C27-6909.

General Form
CALL INDEV(gspname,unit,devicename[,gdoalength])

gspname
is an integer variable that identifies GSP. Its value must be the same as was returned as the "gspname" argument in the call to the INGSP subroutine.

unit
is an integer constant or integer variable the value of which is the data set reference number ("xx") that was specified in the name field of the data definition (DD) job control statement for the device: FTxxF001. This value must be greater than seven.

devicename
is an integer variable to which a value is assigned by the INDEV subroutine to identify the 2250 initialized by this call. This value must be supplied in calls to other GSP subroutines that refer to this 2250.

gdoalength
is an integer constant or integer variable the value of which is the length in bytes of the graphic data output area to be used for each graphic data set associated with the 2250 initialized by this call. If this argument is omitted, a value of 256 bytes is assumed for the graphic data output area. The value of the "gdoalength" argument may be one of the following:

- A value of 128. In this case, a length of 128 bytes is assigned for the graphic data output area. Six of these bytes are used by GSP. Therefore, the size of the graphic data set is limited to 122 bytes of data. This size is most useful as a message area.

- A positive value that is an integral multiple of 256. In this case, the value specified is the size of the graphic data output area. The size of the graphic data set is limited by the amount of 2250 buffer storage available to GSP minus 256 bytes. (GSP uses these additional 256 bytes for control purposes.)

PROGRAMMING CONSIDERATIONS: The "gdoalength" argument can be used to reduce the number of times input/output operations are performed to and from the 2250 buffer. This saves central processing unit time but increases the amount of main storage required for the program. Each time an image generation subroutine is called, the elements generated are placed in the graphic data output area. When this area becomes full, or when the programmer calls the EXEC subroutine, the elements in the graphic data output area are transferred to the 2250 buffer. The larger the graphic data output area, the less often it is filled and the fewer times its contents are transferred to the 2250 buffer.

EXAMPLE: Refer to the detailed description of the INGDS subroutine for examples of proper calls to all the initiation subroutines.

INGDS--Initialize a Graphic Data Set

The INGDS subroutine creates a graphic data set and associates it with a designated 2250. For a display to be produced on a 2250, one or more graphic data sets must be initialized for that 2250. A maximum of 23 calls to the INGDS subroutine may be issued before the use of a graphic data set must be terminated.

Generally, a separate call to the INGDS subroutine is necessary for each graphic data set created. However, several graphic data sets (up to a maximum of 50) may be created by a single call. Graphic data sets created in this manner are called equivalent graphic data sets.

Only one equivalent graphic data set can be displayed at a given time; a call to the EXEC subroutine for an equivalent graphic data set replaces any equivalent graphic data set on the display that was also initialized by the same call. Equivalent graphic data sets are described in more detail under "Programming Considerations" for this subroutine.

General Form
CALL INGDS(devicename,gdsname[,gdoalength][,gdslevel] [,gdsname$_1$...,gdsname$_{49}$])

devicename
is described in "Arguments Used by Many GSP Subroutines."

gdsname
is an integer variable to which a value is assigned by the INGDS subroutine to identify the graphic data set to be created by this call. This value must be supplied in calls to other GSP subroutines that refer to this graphic data set.

gdoalength
is an integer constant or integer variable, the value of which is the length in bytes of the graphic data output area to be used for the graphic data set created by this call. The method of specifying this argument is the same as described for the "gdoalength" argument in the discussion of the INDEV subroutine.

This argument normally need not be used. However, the specification of it overrides the "gdoalength" argument in the call to the INDEV subroutine that initialized the 2250 with which the specified graphic data set is associated.

If this argument is omitted, the size of the graphic data output area is assumed to be either (1) the size specified in the INDEV subroutine for the associated 2250, or (2) 256 bytes if the "gdoalength" argument was omitted from the call to the INDEV subroutine for the associated 2250.

gdslevel
is an integer constant or integer variable that indicates whether or not the standard GSP keying and correlating features will be used for the graphic data set initialized by this call. The argument may have one of the following values:

1 The standard GSP keying and correlating features will be used.

2 A programmer-defined correlation scheme will be used.

If the "gdslevel" argument is omitted, it is assumed that the standard GSP keying and correlating features will be used.

The body of this publication is primarily directed to the use of the standard GSP keying and correlating features. Use of a programmer-defined correlation scheme is discussed in Appendix F.

gdsname$_1$
is an integer variable that identifies a graphic data set that is to be considered equivalent with the graphic data set identified by the "gdsname" argument in this call. The value of this variable is assigned by GSP and must be passed to other GSP subroutines that are to refer to this graphic data set. This argument may be repeated a maximum of 49 times to identify the desired number of equivalent graphic data sets that are to be initialized by this call.

CAUTION: The number of bytes of data within an equivalent graphic data set cannot exceed the length specified by the "gdoalength" argument or 256 bytes if that argument had not been specified.

PROGRAMMING CONSIDERATIONS: Like any other graphic data set, an equivalent graphic data set can be placed in include or omit status, can be executed, and can be modified.

If an equivalent graphic data set is placed in include or omit status, all other equivalent graphic data sets created by the same call are also placed in that same status.

If an equivalent graphic data set is executed by a call to the EXEC subroutine, that graphic data set is moved to the 2250 buffer and displayed just as any other graphic data set would be.

Only one of the equivalent graphic data sets initialized by the same call can be displayed at a given time. A call to the EXEC subroutine to display an equivalent graphic data set causes any previously-displayed graphic data set in equivalence with it to be removed from the screen.

If the use of an equivalent graphic data set is terminated, the use of all other graphic data sets in equivalence with it are also terminated. In addition, all 2250 buffer storage allocated to these equivalent graphic data sets is freed.

EXAMPLE: The example below shows proper calls to the initiation subroutines where a program is to display images on a 2250 by using only one graphic data set, and depicts the use of the null variable. The subroutines called by these statements are the first three GSP subroutines called in this program.

Statement 10 allocates one word of storage for a variable named IGNORE.

Statement 11 assigns a value of -5 to IGNORE. Since Statement 12 defines IGNORE as the null variable, the value of -5 designates that the program is to be abnormally terminated and a dump is to be produced if a return code of five is made available during the processing of a GSP subroutine.

Statement 12 establishes communication links between the program and the GSP and designates the variable named IGNORE as the null variable. This statement also causes a unique value to be assigned to the variable IGSP.

Statement 13 identifies the 2250 on which images are to be displayed; it refers to the 2250 as being unit number 49. IGRAFD is the name of the variable the value of which (assigned by the INDEV subroutine) will be used in subsequent calls to refer to that 2250.

Statement 14 identifies the graphic data sets that are to be used for the display of images as equivalent graphic data sets. It refers to these graphic data sets as IGDS1, IGDS2, and IGDS3, and associates them with the 2250 (identified as IGRAFD). In this statement, the programmer did not wish to specify the third and fourth arguments, which were optional. However, he wanted to specify a fifth and a sixth argument. Therefore, he wrote IGNORE as the third and fourth arguments and followed them with the appropriate fifth and sixth arguments. This causes the third and fourth arguments to be ignored and the fifth argument to be accepted.

```
10  DIMENSION  IGNORE(1)
11  IGNORE(1)=-5
12  CALL  INGSP(IGSP,IGNORE)
13  CALL  INDEV(IGSP,49,IGRAFD)
14  CALL  INGDS(IGRAFD,IGDS1,IGNORE,IGNORE,IGDS2,IGDS3)
```

SPEC--Specify Link or Load Status

The SPEC subroutine defines whether particular GSP subroutines are to be loaded or linked to, thus overriding the link/load status previously defined for those subroutines. Its use can reduce the time required for dynamic acquisition of GSP subroutines and the amount of main storage needed for a GSP program. This is described in the section "Processing Efficiency of GSP Programs."

The SPEC subroutine may be called at any point in the user's program and as many times as desired. If the status of a subroutine is changed to link-to after a copy of that subroutine had been previously loaded into main storage, the loaded copy is removed from main storage.

General Form
CALL SPEC(gspname,code,rtnumber[,rtnumber...])

gspname
: is an integer variable that identifies GSP. Its value must be the same as was returned as the "gspname" argument in the call to the INGSP subroutine.

code
: is an integer constant or integer variable the value of which defines whether a subroutine is to be linked-to or loaded as follows:

  1 All subroutines defined in this call are to be loaded.

  2 All subroutines defined in this call are to be linked to.

rtnumber
: is an integer constant or integer variable the value of which designates a subroutine to be affected by this call. The value must be positive except when it indicates the highest value of a range of values. The values and their corresponding subroutines are listed in Table 3. This argument may be repeated as many times as necessary to define the link/load status for as many subroutines as desired.

  To reduce the number of "rtnumber" arguments necessary in a calling sequence, a pair of "rtnumber" arguments may be used to define a range of subroutines as follows (see the example at the end of this description):

  - The argument that identifies the subroutine that has the lowest corresponding value in the range must appear immediately before the argument that identifies the subroutine with the highest corresponding value in the range.

  - The argument that identifies the subroutine with the highest corresponding value in the range must be a negative value and must appear in the argument list immediately after the argument that identifies the subroutine with the lowest corresponding value in the range.

EXAMPLE: The example that follows designates that the LKSUB, INCL, OMIT, and ORGEN subroutines are to be loaded subroutines (see Table 3):

```
CALL  SPEC (IGSP,1,36,-38,51)
```

• Table 3. Identifying GSP Subroutines in Calls to SPEC Subroutine

Name	Number	Name	Number	Name	Number	Name	Number
INDEV	1	GSPRD*	17	MVPOS	31	ORGDS	44
TMDEV	2	RCURS*	18	BGSEQ	32	LOCPN	45
INGDS	3	ICURS*	19	BGSUB	33	BGTRK	46
TMGDS	4	SDATM	20	ENSEQ	34	RDTRK	47
CRATL	5	SGRAM	21	ENSUB	35	ENTRK	48
ENATL	6	SDATL	22	LKSUB	36	DFSTR	49
ENATN	7	SGDSL	23	INCL	37	PLSTR	50
DSATN	8	SSCIS	24	OMIT	38	ORGEN	51
MPATL	9	SCHAM	25	EXEC*	39	CNVRT	52
MLPEO	10	PLINE*	26	RESET	40	ITRC*	53
SLPAT	11	PPNT*	27	IDPOS	41	ITBP*	54
MLITS	12	PSGMT*	28	FSMOD	42	RTBP*	55
RQATN*	13	PTEXT*	29	STEOS	43	ITST*	56
SALRM	16	STPOS	30				

Notes:

1. An asterisk to the right of the subroutine name denotes that load is the predefined GSP status for that subroutine.
2. Subroutine numbers 14 and 15 are reserved. However, they may be included within a range specified in the call to the SPEC subroutine.
3. The status of the INGSP and TMGSP subroutines cannot be altered by a call to the SPEC subroutine and, therefore, have no associated subroutine number. These two subroutines always reside in main storage.
4. Estimates of the amount of storage required by each of these subroutines are contained in the publication IBM System/360 Operating System: Storage Estimates, Form C28-6551.

TMGDS--Terminate the Use of a Graphic Data Set

The TMGDS subroutine terminates the use of a particular graphic data set and frees all main and buffer storage associated with that data set.

If the graphic data set terminated is an equivalent graphic data set, all other graphic data sets in equivalence with the terminated graphic data set are also terminated. Equivalent graphic data sets are discussed in the detailed description of the INGDS subroutine.

General Form
CALL TMGDS(gdsname)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

PROGRAMMING CONSIDERATIONS: The programmer need not specifically call the TMGDS subroutine to terminate the use of every graphic data set in his program. This is because (1) a call to the TMDEV subroutine also terminates all graphic data sets associated with the specified 2250, and (2) a call to the TMGSP subroutine automatically terminates all graphic data sets. However, when main storage space is a consideration, it is advisable to terminate the use of graphic data sets when they are no longer needed. This allows storage occupied by those graphic data sets to be used for other purposes.

EXAMPLE: The example below shows the use of the TMGDS subroutine. Statement 15 initializes graphic data set IGDS1. Assuming that the value of IGDS1 is the same as returned in statement 15, statement 70 terminates the use of that graphic data set and frees all storage associated with it.

```
15  CALL  INGDS(IGRAFD,IGDS1)
        .
        .
        .
70  CALL  TMGDS(IGDS1)
```

TMDEV--Terminate the Use of a Graphic Device

The TMDEV subroutine terminates the use of a particular 2250 and all graphic data sets associated with that 2250. All main storage occupied by the terminated graphic data sets is freed.

General Form
CALL TMDEV(devicename)

devicename
is described in "Arguments Used by Many GSP Subroutines."

EXAMPLE: The example below shows the use of the TMDEV subroutine. Statement 15 initializes a 2250 identified as IGRAFD. Statements 16 and 17 create graphic data sets that are to be associated with IGRAFD. Assuming that the value of IGRAFD is the same as was returned in statement 15, statement 71 terminates the use of IGRAFD. If graphic data sets IGDS1 and IGDS2 have not been terminated previously, they are terminated when IGRAFD is terminated.

```
15  CALL  INDEV(IGSP,49,IGRAFD)
16  CALL  INGDS(IGRAFD,IGDS1)
17  CALL  INGDS(IGRAFD,IGDS2)
        .
        .
        .
71  CALL  TMDEV(IGRAFD)
```

TMGSP--Terminate Use of the Graphic Subroutine Package

The TMGSP subroutine terminates the use of the GSP and frees all main storage and buffer storage associated with it. This subroutine terminates all 2250's and graphic data sets that had not been previously terminated. The TMGSP subroutine should be called when the graphic processing portion of the user's program has been completed.

General Form
CALL TMGSP(gspname)

gspname
is an integer variable that identifies the GSP. Its value must be the same as was returned as the "gspname" argument in the call to the INGSP subroutine.

OPTION DEFINITION SUBROUTINES

This section describes the subroutines that define the characteristics of the data to be supplied to image generation subroutines and of the display to be produced. Use of these option definition subroutines in a graphic program is optional. For each subroutine not called, pre-established default conditions are assumed. The option definition subroutines and their default conditions are listed in Table 4.

Each option definition subroutine may be called as often as desired. Options selected affect subsequent calls only; data produced by previous calls is not affected. Also, the options selected apply only to the specified graphic data set.

Table 4. Default Conditions for Option Definition Subroutines

Mnemonic	Name	Default Conditions
SDATM	Set Data Mode	Real, absolute input data.
SGRAM	Set Graphic Mode	Optimized output.
SCHAM	Set Character Mode	Basic size, protected characters.
SGDSL	Set Graphic Data Set Limits	Graphic data set boundaries are the same as the screen boundaries.
SDATL	Set Data Limits	Input data limits are the same as the graphic data set boundaries and no scaling is to be performed.
SSCIS	Set Scissoring Option	Images are to be scissored at the screen boundaries and image generation is to continue after scissoring.

SDATM--Set Data Mode

The SDATM subroutine defines the type and form of the data that the user will provide as x- and y-coordinates for the image generation subroutines that refer to the specified graphic data set. This input data may be of real or integer type, and of absolute or incremental form.

Absolute form consists of input data the values of which are actual coordinates. Incremental form consists of input data the values of which represent displacements from a previous point. If the subroutine is not called for a graphic data set, it is assumed that real, absolute data is provided as the x- and y-coordinates.

General Form
CALL SDATM(gdsname,xmode[,ymode])

gdsname
is described in "Arguments Used by Many GSP Subroutines."

xmode,ymode
are integer constants or integer variables that define the type and form of the data that will be provided as x- and y-coordinates. The constants or variables must represent the following values:

1 for real, absolute
2 for real, incremental
3 for integer, absolute
4 for integer, incremental

If the "ymode" argument is omitted, the data mode for y-coordinates is assumed to be the same as specified in the "xmode" argument.

CAUTION: Input data for a particular graphic data set will always be interpreted as being of the type and form defined by the SDATM subroutine for that graphic data set. If the data is not of the type and form defined by the SDATM subroutine, unpredictable results will occur except in regard to scaling. For scaling, if the SDATM subroutine is called to change the input mode from real to integer or vice versa, input data limits previously defined via the SDATL subroutine need not be redefined. Instead, these data limits are automatically converted by GSP to their equivalent real or integer values and scaling is performed in the new mode.

EXAMPLE: The example sets the data mode for the graphic data set identified as IGDS1 so that the programmer can provide integer, absolute data as x- and y-coordinates in subsequent calls to image generation subroutines for that graphic data set. The "xmode" argument is specified as an integer variable. The "ymode" argument is omitted and, therefore, is assumed to be the same as the "xmode" argument.

```
15  IC=3
20  CALL  SDATM(IGDS1,IC)
```

SGRAM--Set Graphic Mode

The SGRAM subroutine defines the form of output graphic orders and data to be produced by image generation subroutines for a specified graphic data set. These output orders and data may be absolute, incremental, or optimized. (Incremental output is produced only on the IBM 2250, Model 1, with graphic design feature, and the IBM 2250, Model 3.)

Optimized data represents input data that has been transformed by image generation subroutines into a form that minimizes the amount of buffer storage required to display an element. This form may consist of absolute data, incremental data, or a combination of the two.

If the SGRAM subroutine is not called for a particular graphic data set, optimized form is assumed. However, only absolute mode is available on the 2250, Model 1 without graphic design feature. In these cases, optimized data is absolute.

General Form
CALL SGRAM(gdsname,gmode)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

gmode
is an integer constant or integer variable that defines the type of output that is to be produced by image generation subroutines. The constant or variable must represent the following values:

1 for optimized
2 for absolute
3 for incremental

PROGRAMMING CONSIDERATIONS: Setting a particular graphic mode has no effect on the data mode set by a call to the SDATM subroutine. That is, absolute output can be produced using incremental input, and vice versa.

When incremental or optimized output is to be produced within a graphic data set, the STPOS subroutine should be the first image generation subroutine called within that graphic data set. This establishes a beam position to which subsequent beam positions can be related.

EXAMPLE: The example below sets the graphic mode for the graphic data set identified as IGDS1 so that image generation subroutines will generate absolute output data.

```
30  CALL  SGRAM(IGDS1,2)
```

SCHAM--Set Character Mode

The SCHAM subroutine defines the size (basic or large) of characters to be produced when the PTEXT subroutine is called for a particular graphic data set. This subroutine also defines whether the characters are to be protected or unprotected.

A protected character cannot be overlaid (replaced) by a character entered from the alphameric keyboard. However, an unprotected character is replaced by a character entered from the alphameric keyboard when the cursor appears beneath its position on the screen.

(The cursor is a symbol displayed on the screen that marks the position at which the next character entered from the alphameric keyboard is to appear. A cursor may be inserted into a graphic data set by means of the ICURS subroutine.)

If the SCHAM subroutine is not called for a graphic data set, it is assumed that basic size, protected characters are to be produced for that graphic data set.

General Form
CALL SCHAM(gdsname,mode)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

mode
is an integer constant or integer variable that defines the type of

characters to be produced. The constant or variable must represent the following values:

1 for basic size, protected.
2 for large size, protected.
3 for basic size, unprotected.
4 for large size, unprotected.

EXAMPLE: The example below sets the character mode for the graphic data set identified as IGDS1 so that large size, protected characters will be produced when the PTEXT subroutine is called for that graphic data set.

```
20  CALL  SCHAM(IGDS1,2)
```

SGDSL--Set Graphic Data Set Limits

The SGDSL subroutine defines the size and position of a particular graphic data set with respect to the boundaries of the 2250 screen.

The screen is the total surface of the 2250 on which images can be displayed. A graphic data set is a rectangular area that may overlap the screen. It may be entirely on the screen or some portion of it may extend beyond the screen boundaries. Figure 1, which appears earlier in this manual, shows the relationship of the graphic data set and images.

If the SGDSL subroutine is not called for a particular graphic data set, the boundaries of the graphic data set are assumed to be the same as the boundaries of the screen.

General Form
CALL SGDSL(gdsname,gllx,glly,gurx,gury[,sllx,slly,surx,sury])

gdsname
is described in "Arguments Used by Many GSP Subroutines."

gllx,glly
are constants or variables representing x- and y-coordinates of the lower left corner of the graphic data set. These coordinate values must be less than the coordinate values used to represent the upper right corner of the graphic data set.

gurx,gury
are constants or variables representing x- and y-coordinates of the upper right corner of the graphic data set. These coordinate values must be greater than the coordinate values used to represent the lower left corner of the graphic data set.

sllx,slly
are constants or variables representing x- and y-coordinates that correspond to the lower left corner of the screen. These coordinate values must be less than the coordinate values used to represent the upper right corner of the screen.

surx,sury
are constants or variables representing x- and y-coordinates that correspond to the upper right corner of the screen. These coordinate values must be greater than the coordinate values used to represent the lower left corner of the screen.

DEFAULT ARGUMENTS: If the "sllx,slly,surx,sury" arguments are omitted, the screen boundaries are assumed to be 0,0,4095,4095. These default coordinates are absolute values of the same type (real or integer)

specified by the SDATM subroutine, or by default, for the designated graphic data set.

CAUTION: The constants or variables used as arguments to represent screen and graphic data set coordinates must be absolute values of the type (real or integer) specified by the call to the SDATM subroutine for the specified graphic data set. If the SDATM subroutine has not been called for this graphic data set, the data must be real and absolute.

EXAMPLE: Refer to the example in the detailed description of the SDATL subroutine for an example of a call to the SGDSL subroutine.

SDATL--Set Data Limits

The SDATL subroutine allows the programmer to define a rectangular coordinate system for his input data to image generation subroutines that differs from the coordinate system established for the graphic data set by the SGDSL subroutine.

The upper and lower limits specified for input data are mapped so that they correspond to the upper right and lower left corners of the designated graphic data set.

Subsequent input data is appropriately scaled so that it relates to this newly defined coordinate system. Images associated with data within the designated range are displayed within the graphic data set. Images associated with data that lies outside the designated range are scissored as defined by the scissoring options specified by the SSCIS subroutine.

If the SDATL subroutine is not called for a graphic data set, data limits are assumed to be the same values specified for the graphic data set boundaries by the SGDSL subroutine, and no scaling is performed.

General Form
CALL SDATL(gdsname,$xlim_1$,$ylim_1$,$xlim_2$,$ylim_2$)

gdsname
: is described in "Arguments Used by Many GSP Subroutines."

$xlim_1$,$ylim_1$
: are constants or variables representing the x- and y-coordinate values that correspond to the lower left corner of the graphic data set. These values may be greater or less than the values specified as the "$xlim_2$" and "$ylim_2$" arguments.

$xlim_2$,$ylim_2$
: are constants or variables representing the x- and y-coordinate values that correspond to the upper right corner of the graphic data set.

CAUTION: If the SDATM subroutine has not been called for the specified graphic data set, real and absolute values must be supplied to the SDATL subroutine.

EXAMPLE: The example that follows defines a data scaling factor that is to be used when an image generation subroutine is called for the graphic data set identified as IGDS1.

```
10  CALL  SGDSL(IGDS1,0,5,5,10,0,0,10,10)
15  CALL  SDATL(IGDS1,0,0,7000,7000)
```

Statement 10 establishes the size and position of graphic data set IGDS1 relative to the screen. It assigns coordinates of 0,0 and 10,10 to the lower left and upper right corners of the screen, and coordinates of 0,5 and 5,10 to the lower left and upper right corners of IGDS1. Therefore, IGDS1 occupies the top left quarter of the screen as shown in Figure 2.

Statement 15 designates the range for input data to image generation subroutines for graphic data set IGDS1 as from 0,0 to 7000,7000. These data limits are mapped so that 0,0 corresponds to the lower left corner of IGDS1 and 7000,7000 corresponds to the upper right corner of IGDS1 (see Figure 3). All subsequent input data to image generation subroutines associated with IGDS1 will now be scaled so that it corresponds to this new coordinate system. If the SDATL subroutine were not called, the input data would not be scaled and the images associated with it would be displayed within IGDS1 only if they were within the limits 0,2048 and 2048,4095.

SSCIS--Set Scissoring Option

The SSCIS subroutine defines the image scissoring option for a graphic data set. Image scissoring involves truncating those portions of a display that extend beyond the boundaries of the graphic data set or the screen. When a portion of the graphic data set extends beyond one or more of the screen boundaries, the screen boundaries intersecting the graphic data set are considered to be the graphic data set boundaries.

If the SSCIS subroutine is not specified for a particular graphic data set, images are scissored at the screen boundaries and image generation continues after the scissoring.

General Form
CALL SSCIS(gdsname,scissoring)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

scissoring
is an integer constant or integer variable that designates the scissoring option selected for the specified graphic data set. Its value must be one of the following:

+1 Scissoring occurs at screen boundaries, and image generation continues after the scissoring.

-1 Scissoring occurs at screen boundaries, and image generation does not continue after the scissoring.

+2 Scissoring occurs at graphic data set boundaries, and image generation continues after the scissoring.

-2 Scissoring occurs at graphic data set boundaries, and image generation does not continue after the scissoring.

+3 No scissoring is to be done.

CAUTION: If a value of three is specified as the "scissoring" argument, the value of subsequent data is not checked to see if it lies within the screen or graphic data set boundaries. When the data exceeds the screen boundaries, the display is affected as described for absolute and incremental data in the appropriate 2250 component description manual. (Refer to "The 2250 Display Unit" for a listing of these manuals.)

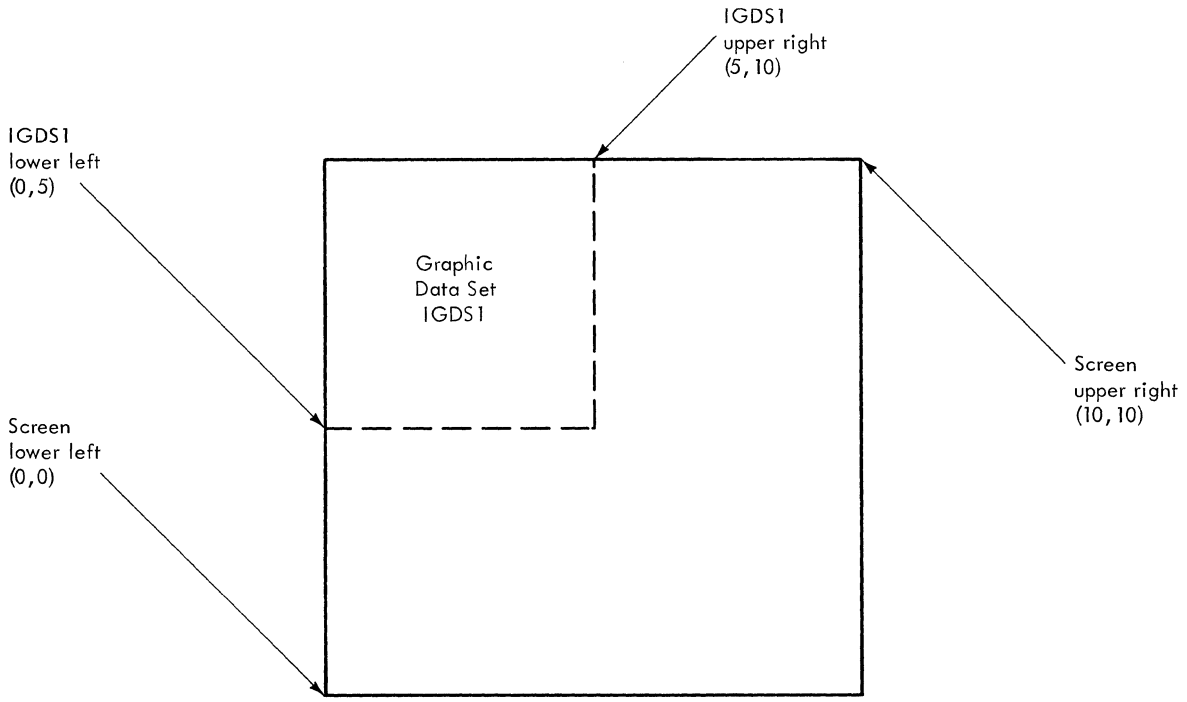


Figure 2. Defining the Size and Position of a Graphic Data Set

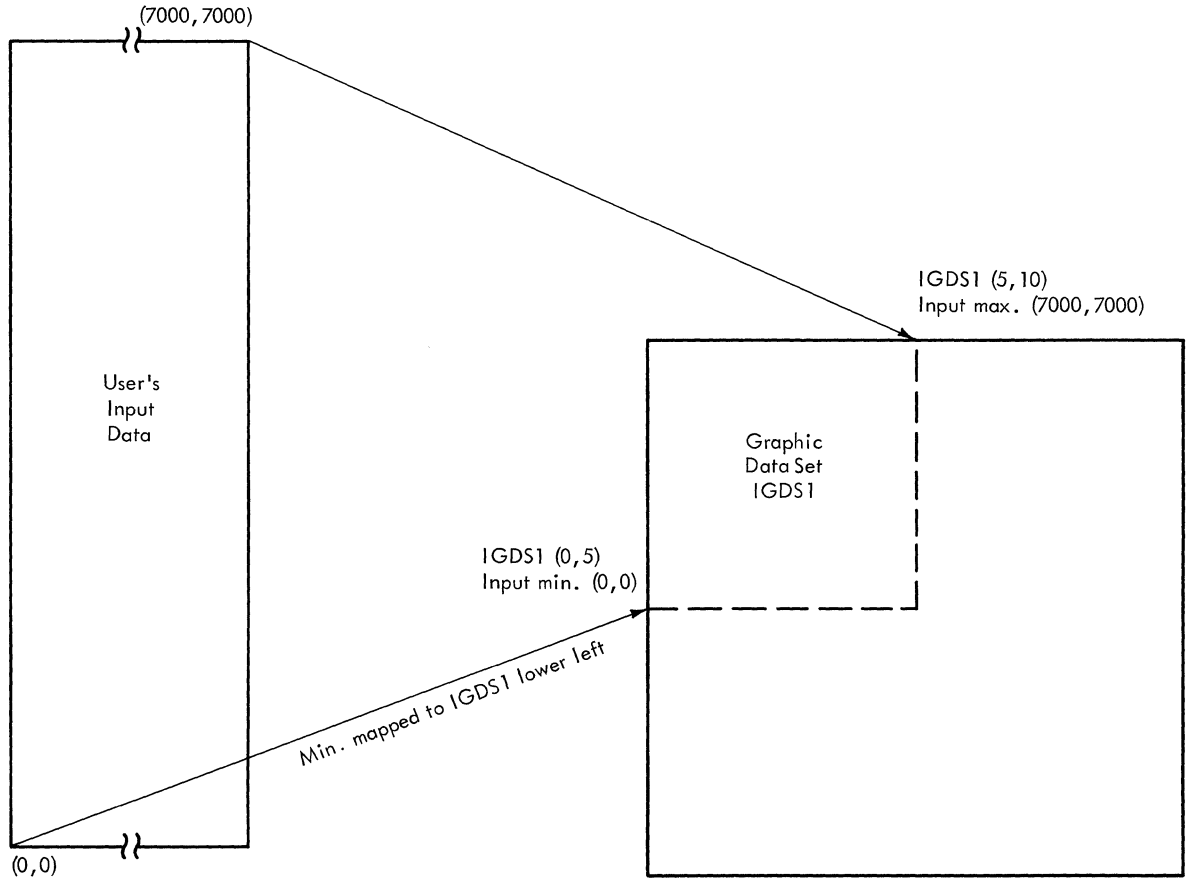


Figure 3. Defining Limits of the User's Input Data

IMAGE GENERATION SUBROUTINES

This section describes the subroutines that generate the elements necessary for displaying images associated with a graphic data set. These subroutines, called the image generation subroutines, create elements that identify the position on the screen where an image is to begin, and that cause the display of lines, points, and characters. Images are only displayed after a call is issued to the EXEC subroutine for the graphic data set that contains the elements that produce the images. The image generation subroutines are:

- Move Beam to a Position (MVPOS)
- Set Beam at an Absolute Position (STPOS)
- Plot Lines (PLINE)
- Plot Line Segments (PSGMT)
- Plot Text (PTEXT)
- Set an End-Order-Sequence Order (STEOS)

Elements generated by the MVPOS and STPOS subroutines are called positioning elements. Those generated by the PLINE, PSGMT, and PPNT subroutines are called graphic elements. Those generated by the PTEXT subroutine are called text elements. Those generated by the STEOS subroutine are called end-order-sequence order elements.

If an element is keyed or correlated, it may be generated in either include status or omit status. Elements not keyed or correlated are always generated in include status.

Images associated with elements in include status are displayed. Images associated with elements in omit status are not displayed. Further effect of include or omit status on an element depends on the element type as follows:

- A graphic element causes the 2250 beam to be moved as directed by the element. When the element is in omit status, the beam is moved to the end point of the element but no image is displayed as a result of that movement.
- A positioning element or text element in omit status does not cause the 2250 beam to be moved as directed by the element; i.e., the beam remains at the screen location at which it was before the element was executed. Subsequently, if the element following a positioning or text element in omit status does not absolutely position itself, that succeeding element will have a different starting position than it did previously.
- An end-order-sequence order element in omit status does not cause an attention to occur and display regeneration continues as normal. An end-order-sequence order element in include status causes an attention to occur and stops display regeneration.

The status of an element that has not been keyed or correlated may not be modified from its original include status. However, the status of a keyed or correlated element may be modified as desired from include status to omit status, and vice versa, by a call to either the Place in Include Status (INCL) or the Place in Omit Status (OMIT) subroutine. For detailed descriptions of these subroutines, refer to the section "Controlling When Images are Displayed."

A keyed or correlated element can be updated by a call to the same subroutine that generated it. The effect of updating an element is a replacement of the element. Updating is described in detail in "The Update Facility."

As elements are created by image generation subroutines, they are sequentially placed in the graphic data sets identified by the calls to those subroutines. Within each graphic data set, the coordinates that represent the end point of the most recent element generated within that graphic data set are retained.

These retained coordinates are updated each time a new element is created. They are used:

- To apply image scissoring options as designated by the SSCIS subroutine.

- As the starting position from which the end point of the next element to be created within that graphic data set is computed. (Thus, the end point of one element is the starting position of the succeeding element.)

Subsequently, any time the sequence in which elements are created within a graphic data set is interrupted (such as in updating and resetting), the desired starting position must be reestablished prior to calling an image generation subroutine for creating the next element to be placed within that graphic data set. In the case of updating, this must be done prior to the update call and prior to adding a new element to the graphic data set after the update call. In the case of resetting, this must be done prior to generating a new element after the reset operation has been performed. If this is not done, images associated with the newly created elements may appear in different locations than desired and image scissoring may occur at an undesirable time during execution of the program.

Reestablishing the desired starting position for new elements is accomplished in one of the following ways:

- By a call to the STPOS subroutine, or

- By a call to the Indicate Beam Position (IDPOS) subroutine if the beam position is known and that position is to be the starting position of the new element.

Furthermore, when incremental or optimized data is to be produced within a graphic data set, the STPOS subroutine should be the first image generation subroutine called within that graphic data set. This establishes an initial beam position for that graphic data set to which subsequent beam positions can be related.

CAUTION: All input data to image generation subroutines must be of the type (integer or real) and form (absolute or incremental) designated by the most recent call to the SDATM subroutine. All output data will be of the form (absolute, incremental, or optimized) defined by the most recent call to the SGRAM subroutine. Any input increment less than one raster unit (0.0117 inch) from the previous increment is ignored.

Input coordinates must be:	• Real or integer, and absolute or incremental as defined by the most recent call to the SDATM subroutine, or its default condition, for the specified graphic data set.
Input coordinates are:	• Appropriately scaled as defined by the most recent call to the SDATL subroutine (in combination with the most recent call to the SGDSL subroutine), or its default condition, for the specified graphic data set.
Output graphic element will be:	• Absolute, incremental, or optimized as defined by the most recent call to the SGRAM subroutine, or its default condition, for the specified graphic data set.
Output graphic element will be:	• In include or omit status as designated by the "gencode" argument.
PTEXT Subroutine Only	
Input text string must be:	• Packed four characters to a word and left-aligned.
When a text element in include status is executed, the characters will be:	• Basic or large, and protected or unprotected, as defined by the SCHAM subroutine, or its default condition, for the specified graphic data set.

Figure 4. Input/Output Requirements and Programming Considerations for the MVPOS, PLINE, PPNT, PSGMT, and PTEXT Subroutines.

MVPOS--Move Beam to a Position

The MVPOS subroutine creates a positioning element that moves the 2250 beam to a specified screen location without producing an image. Programming considerations and input/output requirements for the MVPOS subroutine are listed in Figure 4.

General Form
CALL MVPOS(gdsname,xcoor,ycoor[,corrval][,key][,gencode])

gdsname,corrval,key,gencode
are described in "Arguments Used by Many GSP Subroutines."

xcoor,ycoor
are constants or variables representing x- and y-coordinates that define where the beam is to be moved on the screen.

STPOS--Set Beam at Absolute Position

The STPOS subroutine performs the same function as the MVPOS subroutine except for the following:

- Absolute input data must always be supplied to the STPOS subroutine regardless of the mode specified by the SDATM subroutine. This input data may be either real or integer as defined by the SDATM

subroutine. (Note: If the input data happens to be incremental, the STPOS subroutine uses this data as if it were absolute.)

- Absolute output data will always be produced by the STPOS subroutine regardless of the mode specified by the SGRAM subroutine.

When incremental or optimized output is to be produced within a graphic data set, the STPOS subroutine should be the first image generation subroutine called for that graphic data set. This establishes a beam position to which subsequent beam positions can be related.

General Form
CALL STPOS(gdsname,xcoor,ycoor[,corrval][,key][,gencode])

gdsname,corrval,key,gencode
are described in "Arguments Used by Many GSP Subroutines."

xcoor,ycoor
are constants or variables representing x- and y-coordinates that define where the beam is to be positioned on the screen. These input coordinates must be of absolute form.

PLINE--Plot Line(s)

The PLINE subroutine creates a graphic element that displays a line or a series of connected lines. Programming considerations and input/output requirements for the PLINE subroutine are listed in Figure 4.

To cause a line to be drawn, the programmer need only specify the screen location at which the line is to end. This location is called the end point of the line. When displaying a series of connected lines, the end point of each line is the starting point of its succeeding line.

General Form
CALL PLINE(gdsname,xcoor,ycoor[,corrval][,key][,gencode][,count] [,xindex][,yindex][,xincr][,yincr])

gdsname, corrval, key, gencode
are described in "Arguments Used by Many GSP Subroutines."

xcoor,ycoor
are constants, variables, or array names representing the respective x- and y-coordinates of the end point(s) of the line(s) to be produced.

count
is an integer constant or integer variable the value of which specifies the number of lines to be produced by this call. This argument must be specified if any of the index or increment arguments are used. If the "count" argument is omitted, one line is produced. If the "count" argument has a value greater than one and neither the index nor increment argument is specified for a particular type of coordinates (x or y), the input representing that type of coordinates is assumed to be in an array with an index value of one.

xindex,yindex
are integer constants or integer variables the values of which are used in indexing corresponding arrays identified by the "xcoor" and "ycoor" arguments. If these and the "xincr" and "yincr" arguments are omitted, a value of one is used in indexing corresponding arrays identified by the "xcoor" or "ycoor" argument. For further details, refer to "Programming Considerations."

xincr,yincr
are constants or variables the values of which designate increments that are added to the preceding corresponding x- and y-coordinates to form the end points of each succeeding line to be produced. The values specified by the respective "xcoor" and "ycoor" arguments are used to form the first line. The number of lines to be produced must be specified in the "count" argument. For further details, refer to "Programming Considerations."

PROGRAMMING CONSIDERATIONS: The "xindex" and "yindex" arguments permit the use of one or two arrays of coordinates to define the end points of a series of connected lines. The number of lines to be produced is defined by the "count" argument. The locations of the first x- and y-coordinates to be used are defined by the "xcoor" and "ycoor" arguments. The number of entries from the location of one x-coordinate to the location of the next x-coordinate is defined by the "xindex" argument. The number of entries from the location of one y-coordinate to the location of the next y-coordinate is defined by the "yindex" argument. From these arguments, the PLINE subroutine proceeds through the array, creating the element for the lines to be drawn.

The "xincr" and "yincr" arguments allow the programmer to increment either the x- or y-coordinates, or both, by a constant amount for each of a series of lines to be drawn without using an array. When one of the arguments is specified (e.g., "xincr"), the value of the increment is added to the previous value for that type of coordinate (e.g., x-coordinate) and this sum becomes the next coordinate. For absolute input, the first coordinate to which the increment value is added is defined by means of either the "xcoor" or "ycoor" argument, whichever is appropriate. For incremental input, the first coordinate to which the increment value is added is determined by adding the value of "xcoor" or "ycoor" as appropriate to the current beam position to form a new beam position. The number of lines to be drawn is defined by the "count" argument.

The index and increment arguments cannot both be specified for a single set of coordinates. However, an index argument may be specified for a set of x-coordinates and an increment argument for a set of y-coordinates, and vice versa.

EXAMPLE: Figure 5 shows the use of the PLINE subroutine to generate eight lines from a point at the center of the screen to the circumference of a circle. The circle has a radius of 100 units. The eight lines are to be displayed 45 degrees apart. The display produced is shown in Figure 6.

```
10    DIMENSION NULL(1)
20    NULL(1) = -5
30    CALL INGSP(IGSP,NULL)
40    CALL INDEV(IGSP,40,IGRAFD)
50    CALL INGDS(IGRAFD,IGDS1)
60    CALL SDATL(IGDS1,0.0,0.0,1000.0,1000.0)
70    R = 100.0
80    C = 3.141593/180.0
90    THETA = 45.0
100   DO 150 I = 1,8
110   RADIAN=THETA*FLOAT(I)*C
120   X = 500.0+R*COS(RADIAN)
130   Y = 500.0+R*SIN(RADIAN)
140   CALL STPOS(IGDS1,500.0,500.0)
150   CALL PLINE(IGDS1,X,Y)
160   CALL EXEC(IGDS1)
        .
        .
        .
```

10,20	Designate storage for a variable named NULL and assign a value of -5 to that variable. Since NULL is later identified as the null variable, the value of -5 indicates that the program is to be abnormally terminated and a dump is to be produced only when a return code of five is made available by a GSP subroutine.
30,40,50	Establish communication links among the program, GSP, a 2250, and a graphic data set. Designate NULL as the null variable.
60	Defines the input data limits of graphic data set IGDS1. No other option definition subroutines are called for IGDS1. Subsequently, the default conditions of all other option definition subroutines are assumed.
70	Defines the radius of the circle as 100 units.
80	Defines a constant (C) used to convert degrees to radians.
90	Specifies that the lines are to be 45 degrees apart.
100	Specifies that eight lines are to be displayed and establishes a DO loop with a range through statement 150.
110	Converts degrees to radians.
120,130	Establish the x- and y-coordinates for the end point of one of the lines to be displayed.
140	Creates an element that moves the 2250 beam (in unblanked mode) to the center of the screen. Since the "gencode" argument is not specified, this element is generated in include status.
150	Creates an element that draws a line to the x- and y-coordinates established by statements 120 and 130. Since the "gencode" argument is not specified, this element is generated in include status.
160	Causes the lines to be displayed as shown in Figure 6.

Figure 5. Example of Use of PLINE Subroutine

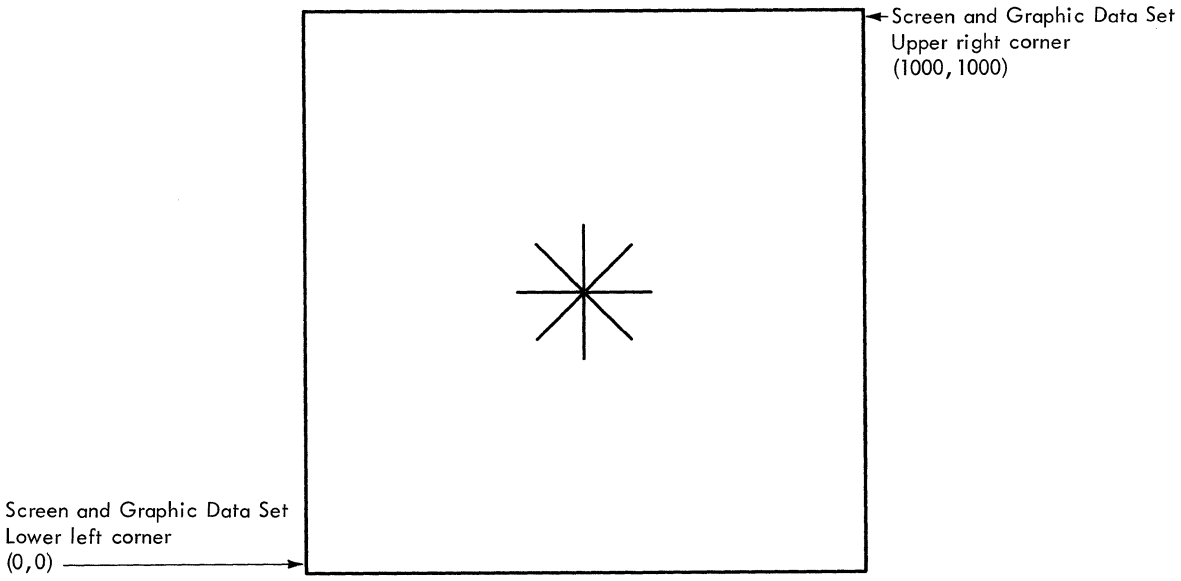


Figure 6. Display Produced by PLINE and PSGMT Examples

PPNT--Plot Point(s)

The PPNT subroutine creates a graphic element that displays one or several points. It performs in the same manner as the PLINE subroutine, except that only end points are displayed.

Programming considerations and input/output requirements for the PPNT subroutine are listed in Figure 4 included earlier in this section. For detailed descriptions of the available arguments, refer to the description of the PLINE subroutine.

General Form
CALL PPNT(gdsname,xcoor,ycoor[,corrval][,key][,gencode] [,count][,xindex][,yindex][,xincr][,yincr])

EXAMPLE: Figure 7 shows the use of the PPNT subroutine to produce a display and then modify that display by means of the update facility. The update facility is described in the section "Replacing and Eliminating Elements Within a Graphic Data Set."

The first display contains 72 points that are plotted on the circumference of a circle that has a radius of 100 units. These points are five degrees apart. The updated display also contains 72 points that are five degrees apart. However, these points are plotted on the circumference of a circle that has a radius of 200 units. Both the initial display and the updated display are shown in Figure 8.

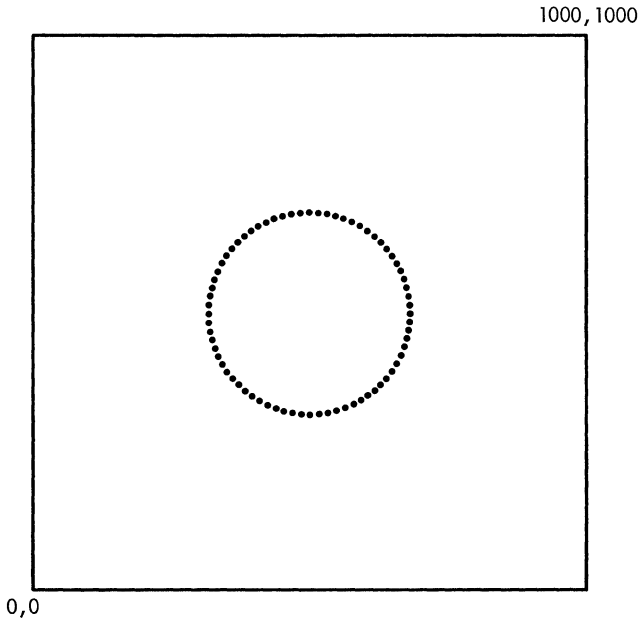
```
10    DIMENSION X(72),Y(72),NO(1)
20    NO(1) = -5
30    CALL INGSP(IGSP,NO)
40    CALL INDEV(IGSP,40,IGRAFD)
50    CALL INGDS(IGRAFD,IGDS1)
60    CALL SDATL(IGDS1,0.0,0.0,1000.0,1000.0)
70    CALL SGRAM(IGDS1,2)
80    R = 100.0
90    C =  3.141593/180.0
100   THETA = 5.0
110   IGEN = 1
120   DO 150 I = 1,72
130   RADIAN=THETA*FLOAT(I)*C
140   X(I) = 500.0 + R*COS(RADIAN)
150   Y(I) = 500.0 + R*SIN(RADIAN)
160   CALL PPNT(IGDS1,X,Y,NO,IKEY,IGEN,72)
170   CALL EXEC(IGDS1)
       .
       .
       .
500   SAVEX = RTBP(IGDS1,2)
510   SAVEY = RTBP(IGDS1,4)
520   IGEN = 3
530   R = 200.0
540   DO 570 I = 1,72
550   RADIAN=THETA*FLOAT(I)*C
560   X(I) = 500.0 + R*COS(RADIAN)
570   Y(I) = 500.0 + R*SIN(RADIAN)
580   CALL PPNT(IGDS1,X,Y,NO,IKEY,IGEN,72)
590   CALL IDPOS(IGDS1,SAVEX,SAVEY)
       .
       .
       .
```

10	Provides dimensions of two arrays, X and Y, that will contain the x- and y-coordinates where points are to be plotted on the screen. Also names the variable (NO).
20	Assigns a value of -5 to the variable NO. Since this variable is later identified as the null variable, the value of -5 indicates that the program is to be abnormally terminated and a dump is to be produced only when a return code of five is made available by a GSP subroutine.
30,40,50	Establish communication links among the program, GSP, a 2250, and a graphic data set. Designate NO as the null variable.
60,70	Define the input data limits of graphic data set IGDS1 (statement 60), and designate that absolute output is to be produced for that graphic data set (statement 70). No other option definition subroutines are called. Subsequently, the default conditions of all other option definition subroutines are assumed.
80	Defines the radius of the circle as 100 units.
90	Defines a constant (C) used to convert degrees to radians.
100	Specifies that the points are to be five degrees apart.

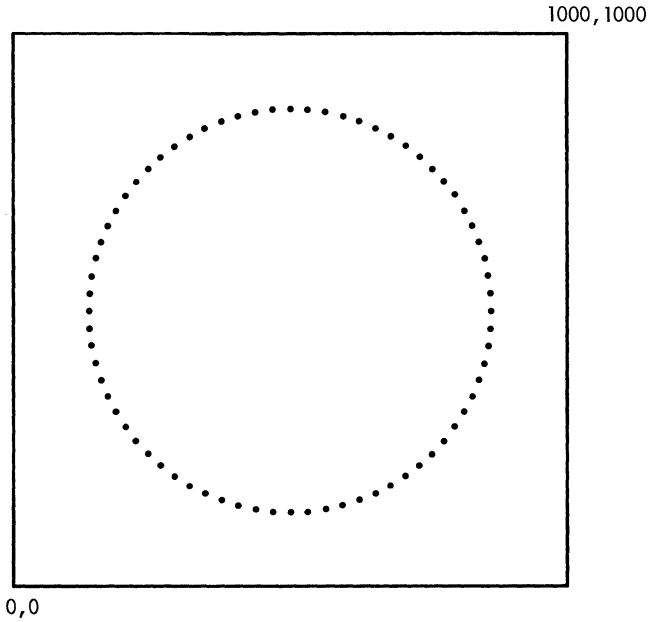
Figure 7. Example of Use of the PPNT Subroutine (Part 1 of 2)

110	Assigns a value of one to IGEN that will be used as the "gencode" argument in the call to the PPNT subroutine (statement 160). This causes the element for plotting points to be generated in include status. The "gencode" argument is defined as a variable by this statement so that it can be changed if the element produced by the PPNT subroutine is to be updated. The "key" argument is defined as the variable IKEY.
120	Specifies that 72 points are to be displayed and establishes a DO loop with a range through statement 150.
130	Converts degrees to radians.
140,150	Establish the coordinates for one of the points to be produced. The x-coordinate is stored in array X. The y-coordinate is stored in array Y.
160	Creates the element for displaying the 72 points.
170	Causes the points to be displayed on the screen as shown in Figure 8.
500,510	Save the current x- and y-positions of the 2250 beam so that the beam can be positioned at the same location after the update call (statement 580) as it was prior to issuance of that call, thus enabling new elements to be added to the graphic data set after the update call.
520	Changes the value of IGEN to three. This causes the next call that contains IGEN as the "gencode" argument to be considered an update call.
530	Defines the radius of the new circle to be displayed as 200 units.
540,550, 560,570	Establish a new array of coordinates for 72 points to be displayed in the same manner as the coordinates were established in statements 120 through 150.
580	Creates the update element and substitutes it for the element previously created by statement 160. The new element is automatically placed in the buffer and its associated image is displayed. A call to the EXEC subroutine is unnecessary. This updated image is shown in Figure 8. Note that the updated element is still identifiable by the value of IKEY.
590	Restores the beam to the position it was in prior to processing of the update call.

Figure 7. Example of Use of the PPNT Subroutine (Part 2 of 2)



First Display



Updated Display

Figure 8. Displays Produced by PPNT Example

PSGMT--Plot Line Segment(s)

The PSGMT subroutine creates a graphic element that displays one or several line segments. It differs from the PLINE subroutine in that segments produced by the PSGMT subroutine need not be connected.

Prior to the generation of a line segment, the beam is moved to the starting location of that segment. This beam movement does not result in an image being displayed. Programming considerations and input/output requirements for the PSGMT subroutine are listed in Figure 4.

```
┌──────────────────────────────────────────────────────────────────────┐
│General Form                                                          │
├──────────────────────────────────────────────────────────────────────┤
│CALL  PSGMT(gdsname,xstart,ystart,xend,yend[,corrval][,key][,gencode] │
│            [,count][,xstartindex][,ystartindex][,xendindex]          │
│            [,yendindex][,xstartincr][,ystartincr][,xendincr]         │
│            [,yendincr])                                              │
└──────────────────────────────────────────────────────────────────────┘
```

gdsname,corrval,key,gencode
are described in "Arguments Used by Many GSP Subroutines."

xstart,ystart
are constants, variables, or array names representing the respective x- and y-coordinates of the starting point of each line segment to be produced by this call. If incremental input mode is specified, the first "xstart" value specified is added to the current x-coordinate beam position to determine the starting point of the first line segment to be produced. If array input is used, the second "xstart" value is added to the computed starting point of the first line segment to determine the starting point of the second line segment; and so forth. The y-coordinate values are determined in the same manner.

xend,yend
are constants, variables, or array names representing the respective x- and y-coordinates of the end point of each line segment to be produced by this call. If incremental input mode is specified, the first "xend" value specified is added to the computed starting point of the first line segment to determine the end point of that first line segment. If array input is used, the second "xend" value specified is added to the computed end point of the first line segment to determine the end point of the second line segment; the third "xend" value specified is added to the computed end point of the second line segment to determine the end point of the third line segment; and so forth. The y-coordinate end values are determined in the same manner.

count
is an integer constant or integer variable the value of which specifies the number of line segments to be produced by this call. This argument must be specified if any of the index or increment arguments are specified. If the "count" argument is omitted, one line segment is produced. If the "count" argument has a value greater than one and neither an index nor increment argument is specified for a particular type of coordinates (x or y), the input representing that type of coordinates is assumed to be in an array with an index value of one.

xstartindex,ystartindex,xendindex,yendindex
are integer constants or integer variables the values of which are used in indexing corresponding arrays identified by the "xstart", "ystart", "xend", and "yend" arguments. If any of these arguments are omitted, a value of one is used in indexing its corresponding

array. Refer to "Programming Considerations" for a further discussion of how these arguments may be used.

xstartincr
: is a constant or variable the value of which designates an increment that is to be added to the x-coordinate starting point of the preceding line segment produced by this call to determine the x-coordinate starting point of the next line segment to be produced. The value specified by the "xstart" argument is used to form the x-coordinate starting point of the first line segment to be produced. The number of line segments to be produced must be specified by the "count" argument. Refer to "Programming Considerations" for a further discussion of how this argument may be used.

ystartincr,xendincr,yendincr
: are specified in the same manner as the "xstartincr" argument to determine y-coordinate starting points and x- and y-coordinate end points of line segments to be produced by this call. The values specified by corresponding "ystart", "xend", and "yend" arguments are used to determine the respective y-coordinate starting point and x- and y-coordinate end point of the first line segment to be produced. Coordinates for subsequent line segments are determined as follows: (1) the value specified as the "ystartincr" argument is added to the y-coordinate starting point of the preceding line segment produced; (2) the value specified as the "xendincr" argument is added to the x-coordinate end point of the preceding line segment produced; (3) the value specified as the "yendincr" argument is added to the y-coordinate end point of the preceding line segment produced.

PROGRAMMING CONSIDERATIONS: The index arguments permit the use of one or more arrays of coordinates to define the starting and end points of a series of line segments. The number of line segments to be produced is defined by the "count" argument. The locations of the first of the coordinates to be used are defined by the respective "xstart", "ystart", "xend", and "yend" arguments. The number of entries from the location of one coordinate to the location of the next coordinate of the same type is defined by the respective index argument. From these arguments, the PSGMT subroutine proceeds through the array(s), creating the element for the line segments to be displayed.

The increment arguments allow the programmer to increment by a constant amount the starting or end point coordinates (or all four) of each of a series of line segments to be produced without using an array.

Both the index and increment arguments may be used with any input mode designated by the SDATM subroutine. An index argument may be specified for one set of coordinates (e.g., starting point x-coordinates) and an increment argument for another set of coordinates (e.g., end point y-coordinates). However, both an index and an increment argument cannot be specified for a single set of coordinates.

EXAMPLE: Figure 9 shows the use of the PSGMT subroutine to generate eight lines from a point at the center of the screen to the circumference of a circle. The circle has a radius of 100 units and the eight lines are to be displayed 45 degrees apart.

The image displayed by the example shown in Figure 9 is the same as displayed by the example for the PLINE subroutine. However, this PSGMT example executes more efficiently than the PLINE example because it contains fewer calls to image generation subroutines. The PLINE example calls two image generation subroutines (STPOS and PLINE) to be executed

```
10    DIMENSION X(8),Y(8),NO(1)
20    NO(1) = -5
30    CALL INGSP(IGSP,NO)
40    CALL INDEV(IGSP,40,IGRAFD)
50    CALL INGDS(IGRAFD,IGDS1)
60    CALL SDATL(IGDS1,0.0,0.0,1000.0,1000.0)
70    R = 100.0
80    C = 3.141593/180.0
90    THETA = 45.0
100   DO 130 I = 1,8
110   RADIAN=THETA*FLOAT(I)*C
120   X(I) = 500.0 +R*COS(RADIAN)
130   Y(I) = 500.0 +R*SIN(RADIAN)
140   CALL PSGMT(IGDS1,500.0,500.0,X,Y,NO,NO,NO,8,0,0)
150   CALL EXEC(IGDS1)
```

10	Provides dimensions of two arrays, X and Y, that will contain the x- and y-coordinates for the end points of line segments to be displayed. Also names the variable NO.
20	Assigns a value of -5 to the variable NO. Since this variable is later identified as the null variable, the value of -5 indicates that the program is to be abnormally terminated and a dump is to be produced only when a return code of five is made available by a GSP subroutine.
30,40,50	Establish communication links among the program, GSP, a 2250, and a graphic data set. Specify NO as null variable.
60	Defines the input data limits of graphic data set IGDS1. No other option definition subroutines are called. Subse quently, the default conditions of all other option definition subroutines are assumed.
70	Defines the radius of the circle as 100 units.
80	Defines a constant (C) used to convert degrees to radians.
90	Specifies that line segments are to be 45 degrees apart.
100	Specifies that eight line segments are to be displayed and establishes a DO loop with a range through statement 130.
110	Converts degrees to radians.
120,130	Establish coordinates for the end point of one of the line segments to be produced. The x-coordinate is stored in array X. The y-coordinate is stored in array Y.
140	Creates the element for producing the eight line segments. The arguments specified indicate the following: (1) output from this subroutine is associated with graphic data set IGDS1, (2 & 3) the first line segment is to start at coordinates 500,500, (4 & 5) arrays that contain the coordinates for the end points of the line segments to be produced are respectively named X and Y, (6 & 7) no correlation value or key is to be associated with this element, (8) this element is to be generated in include status, (9) eight line segments are to be produced, (10 & 11) by being set to zero, that the same starting point is to be used for each of the line segments to be produced, and (12 & 13) by their omission, that each of the arrays is to be indexed by one in order to find the end point of each succeeding line segment.
150	Causes line segments to be displayed as shown in Figure 6.

Figure 9. Example of Use of the PSGMT Subroutine

eight times each for a total of 16 calls. The PSGMT example calls one image generation subroutine (PSGMT) to be executed only one time. The display produced is shown in Figure 6.

PTEXT--Plot Text

The PTEXT subroutine creates a text element that displays one or more characters using the character generator feature of the 2250. The character data is displayed in the same format as it appears in main storage. Characters produced may be either of two sizes (basic or large) and of one orientation (vertical). Character size is determined by the most recent call to the SCHAM subroutine, or by default.

Programming considerations and input/output requirements for the PTEXT subroutine are listed in Figure 4. Scissoring options set by the SSCIS subroutine apply to text in the same manner as they apply to images resulting from the execution of the other image generation subroutines.

General Form
CALL PTEXT(gdsname,text,count[,corrval][,key][,gencode] [,xcoor,ycoor])

gdsname,corrval,key,gencode
are described in "Arguments Used by Many GSP Subroutines."

text
is a variable or array name that specifies the storage location of the first character of text to be displayed. The characters in the text string must be packed four to a word and must be left aligned at the location specified.

count
is an integer constant or integer variable the value of which specifies the number of characters to be displayed.

xcoor,ycoor
are constants or variables representing the x- and y-coordinates of the location where the first character is to be displayed. If these arguments are not specified, the first character is displayed wherever the 2250 beam is positioned at the time the subroutine is called.

EXAMPLE: Figure 10 shows the use of the PTEXT subroutine. This example assumes that the text to be displayed will be read from the first forty columns of a card, a tape record, or a disk record. This is accomplished in the example by a combination of the FORMAT statement and the READ statement.

```
10    DIMENSION TEXT(10),NO(1)
20    NO(1) = -5
30    CALL INGSP(IGSP,NO)
40    CALL INDEV(IGSP,40,IGRAFD)
50    CALL INGDS(IGRAFD,IGDS1)
60    CALL SDATL(IGDS1,0.0,0.0,1000.0,1000.0)
70    CALL SCHAM(IGDS1,2)
80    FORMAT (10A4)
       .
       .
       .
160   READ (5,80) (TEXT(I),I=1,10)
170   CALL PTEXT(IGDS1,TEXT,40,NO,IKEY,1,20.0,975.0)
180   CALL EXEC(IGDS1)
       .
       .
       .
```

10	Defines the size of an array called TEXT where the forty characters to be displayed will be placed by the READ statement so that they can be located by the PTEXT subroutine. Also names the variable NO.
20	Assigns a value of -5 to the variable NO. Since this variable is later identified as the null variable, the value of -5 indicates that the program is to be abnormally terminated and a dump is to be produced only when a return code of five is made available by a GSP subroutine.
30,40,50	Establish communication links among the program, GSP, a 2250, and a graphic data set. Designate NO as the null variable.
60,70	Define the input data limits of graphic data set IGDS1 (statement 60), and specify that large size, protected-characters are to be displayed (statement 70). No other option definition subroutines are called. Subsequently, the default conditions of other option definition subroutines are assumed.
80	Defines the type and amount of characters to be read into the array called TEXT.
160	Reads the text to be displayed into the main storage array TEXT from a card, a tape record, or a disk record.
170	Creates the element for displaying the contents of array TEXT on the screen. The arguments specified indicate the following: (1) output from this subroutine is associated with graphic data set IGDS1, (2) the characters to be displayed are contained in the array TEXT, (3) 40 characters will be displayed, (4) no correlation value is assigned to the text element created by this call, (5) a key is to be assigned to the variable IKEY so that the text element created by this call can be identified later in the program, (6) the text element is to be generated in include status, and (7) the first character of text is to be displayed on the screen at coordinate 20.0, 975.0.
180	Causes the text to be displayed on the screen.

Figure 10. Example of Use of the PTEXT Subroutine

STEOS--Set an End-Order-Sequence Order

The STEOS subroutine places an end-order-sequence order in the next available location within a graphic data set. This order is used in attention handling for controlling display regeneration (see "Communicating With the 2250 Operator").

The element produced by a call to this subroutine is called an end-order-sequence order element. It has no data associated with it and cannot be updated. However, it may be placed in include or omit status.

When the element is executed in include status, an attention occurs and automatic regeneration of the display is stopped. When the element is executed in omit status, no attention occurs and automatic regeneration of the display proceeds as normal.

General Form
CALL STEOS(gdsname[,corrval][,key][,gencode])

gdsname,corrval,key,gencode
: are the same as described in "Arguments Used by Many GSP Subroutines" except that a value of three cannot be assigned as the "gencode" argument.

PROGRAMMING CONSIDERATIONS: The STEOS subroutine is useful in preventing the display regeneration cycle from being interrupted before a desired display completely appears on the screen. Each time new elements are added to the buffer via the EXEC subroutine, display regeneration is stopped, the new elements are placed in the buffer, and display regeneration is restarted. Occasionally, the process of creating new elements, placing them in the GDOA, and transferring them to the buffer is completed by the operating system before the buffer can complete regenerating all the elements previously transferred to it. Thus, only a portion of a desired display (rather than the entire display) will appear on the screen at a given time.

To prevent this, the following should be performed:

1. Call the STEOS subroutine before calling the EXEC subroutine that transfers newly created elements to the buffer.

2. Call the RQATN subroutine to wait for an end-order-sequence attention. The order that causes this attention was placed in the buffer via the call to the STEOS subroutine noted in Step 1.

3. Call the RESET subroutine to remove the end-order-sequence order as soon as the end-order-sequence attention occurs.

4. Call desired image generation subroutines for creating new elements.

An example of the use of the preceding sequence is contained in the detailed description of the RDTRK subroutine.

IDENTIFICATION SUBROUTINES

Elements can be identified as a unit by grouping them into a sequence or a buffer subroutine. The subroutines available for this purpose, called identification subroutines, are described in the sections that follow.

GROUPING ELEMENTS INTO A SEQUENCE

Elements within a graphic data set may be grouped into a sequence by calls to the Begin a Sequence of Elements (BGSEQ) and the End a Sequence of Elements (ENSEQ) subroutines. A sequence consists of all elements between calls to the BGSEQ and ENSEQ subroutines that are associated with the same graphic data set named in those calls.

A sequence must be keyed or correlated. All elements within a sequence are treated as an entity identifiable by the key or correlation value assigned to that sequence. In addition, each element within a sequence may or may not be keyed or correlated independently of the keying or correlation of the sequence itself.

Since it is treated as an entity, a sequence can be manipulated as if it were a single element. It can be placed in include or omit status. It can have a cursor inserted into it. It can be removed by the RESET subroutine. However, it cannot be updated. The manipulation of a sequence does not restrict the manipulation of elements within it.

When a light pen is pointed at an image produced by an element that is part of a sequence, GSP returns to the user's program a key or correlation value for the element on which the light pen detect occurred, and any key or correlation value for the sequence within which the element is contained. This enables the programmer to use an attention-handling routine to identify the entire sequence rather than just the element detected by the light pen. For additional information, refer to the description of the Request Attention Information (RQATN) subroutine.

Use of the BGSEQ and ENSEQ subroutines in a program is optional. They are described in detail in the paragraphs that follow.

BGSEQ--Begin a Sequence of Elements

The BGSEQ subroutine designates the beginning of a sequence of one or more elements within a particular graphic data set. Once this subroutine is called, all elements within the specified graphic data set that are generated between this call and a call to the ENSEQ subroutine for the same graphic data set are included as part of the sequence.

General Form
CALL BGSEQ(gdsname[,corrval][,key][,gencode])

gdsname,corrval,gencode
: are the same as described in "Arguments Used by Many GSP Subroutines," except that a value of three cannot be assigned as the "gencode" argument.

key
: is an integer variable that identifies the sequence begun by this call. Its value, or the key, is assigned by GSP. This key can only be used in a call to the RESET subroutine that is issued before the ENSEQ subroutine is called for this sequence. A new key

is assigned when the ENSEQ subroutine is called and must be used from that point on.

CAUTION: No sequence associated with a particular graphic data set may be defined within another sequence or within a buffer subroutine associated with that same graphic data set. This means that the ENSEQ subroutine must be called for a graphic data set before a second call for that same graphic data set can be issued to the BGSEQ subroutine.

EXAMPLE: For an example of the use of the BGSEQ subroutine, refer to "Example of Creating a Sequence."

ENSEQ--End a Sequence of Elements

The ENSEQ subroutine designates the end of a sequence of elements associated with a graphic data set. Once this subroutine is called, the sequence created consists of all elements within the specified graphic data set that have been generated prior to this call and after a preceding call to the BGSEQ subroutine for the specified graphic data set.

General Form
CALL ENSEQ(gdsname[,key])

gdsname,key
: are described in "Arguments Used by Many GSP Subroutines," except that the key specified in this call replaces any key assigned to this sequence in the call to the BGSEQ Subroutine that created the sequence (see "Programming Considerations" below).

CAUTION: If a correlation value had not been specified in the call to the BGSEQ subroutine for this sequence, the "key" argument must be included in the call to the ENSEQ subroutine.

PROGRAMMING CONSIDERATIONS: A correlation value assigned in the call to the BGSEQ subroutine is valid as long as the sequence exists. Unless the graphic data set is reset to a point within the sequence, a key assigned in the call to the BGSEQ subroutine is valid only until the ENSEQ subroutine is called to designate the end of the sequence. If a key is to be used for identifying the sequence after the ENSEQ subroutine has been called, the "key" argument must be included in the call to the ENSEQ subroutine. This argument may or may not refer to the same variable specified as the "key" argument in the call to the BGSEQ subroutine.

Example of Creating a Sequence

The example below establishes a sequence of three elements within the graphic data set identified by the value of IGDS1. In the example, NO is used as the null variable.

Statement 50 begins a sequence within graphic data set IGDS1 and correlates that sequence with a value of 25. Statement 54 ends the sequence and requests that a key be assigned for the variable IKEY1. Since the elements generated by statements 51, 52, and 53 are associated with graphic data set IGDS1, these three elements make up the sequence just begun. Note that the elements generated by statements 51 and 53 are keyed within the sequence and the element generated by statement 52 is not keyed. Statement 70 transfers the elements making up the sequence to the buffer for execution.

```
50  CALL  BGSEQ(IGDS1,25)
51  CALL  PPNT(IGDS1,X1,Y1,NO,IKEY4)
52  CALL  PTEXT(IGDS1,TEXT1,15)
53  CALL  PPNT(IGDS1,X2,Y2,NO,IKEY5)
54  CALL  ENSEQ(IGDS1,IKEY1)
      .
      .
      .
70  CALL  EXEC(IGDS1)
```

GROUPING ELEMENTS INTO A BUFFER SUBROUTINE (2250 MODEL 3 ONLY)

Occasionally, it is necessary to display the same image at several different screen locations. GSP provides for this to be easily done on the 2250 Model 3 through the use of buffer subroutines. (Buffer subroutines cannot be used with the 2250 Model 1.)

A buffer subroutine consists of one copy of all the elements necessary to display an image. It is placed at a fixed location in the 2250 buffer and can be repeatedly invoked to display its associated image at different screen locations. The starting position of the image is the location of the 2250 beam at the time the buffer subroutine is invoked.

A buffer subroutine is created by a call to the Begin a Buffer Subroutine (BGSUB) subroutine within a particular graphic data set and is terminated by a call to the End a Buffer Subroutine (ENSUB) subroutine for the same graphic data set. All elements created for that graphic data set between these two calls are included in the buffer subroutine. The BGSUB and ENSUB subroutines perform in pairs similar to the BGSEQ and ENSEQ subroutines.

Once a buffer subroutine has been created, the execution of it can be requested at any time (and as many times as desired) by a call to the Link to a Buffer Subroutine (LKSUB) subroutine followed by a call to the EXEC subroutine. The position of the 2250 beam at the time the LKSUB subroutine is called designates the starting position of the image produced by the buffer subroutine being called.

Once an image is displayed by a buffer subroutine, that image is identified by the key or correlation value assigned to the linkage produced by the call to the LKSUB subroutine that caused that image to be displayed, rather than by the key or correlation value assigned to the buffer subroutine itself. Therefore, if the 2250 operator points the light pen at one of several images created by different executions of the buffer subroutine, that light pen attention (if enabled) is associated with the linkage that caused the designated image to be displayed. A subsequent call to the Request Attention Information (RQATN) subroutine makes available the key or correlation value of that linkage. The key or correlation value of the element pointed at by the light pen is not returned. For a description of light pen attention handling, see "Communicating With the 2250 Operator."

All input data to image generation subroutines within a buffer subroutine is assumed to be incremental, regardless of the mode specified by the SDATM subroutine. This input data is appropriately scaled. Howev er, images produced by buffer subroutines are not scissored.

Elements making up a buffer subroutine are always generated in incremental form, regardless of the type of output specified by the SGRAM subroutine. This allows the image to be displayed anywhere on the screen by the programmer specifying an absolute beam position prior to a call for linking to the buffer subroutine.

Since incremental input and output are assumed for a buffer subroutine, the STPOS subroutine should not be called within a buffer subroutine. This is because the STPOS subroutine always assumes absolute input and produces absolute output. If the STPOS subroutine is called, the call will be treated as if it were a call to the MVPOS subroutine that expects incremental input and produces incremental output.

Descriptions of the BGSUB, ENSUB, and LKSUB subroutines and an example of their use are contained in the paragraphs that follow.

BGSUB--Begin a Buffer Subroutine

The BGSUB subroutine designates the beginning of a buffer subroutine within a graphic data set. Once this subroutine is called, all elements within the specified graphic data set that are generated between this call and a call to the ENSUB subroutine for the same graphic data set are included as part of the buffer subroutine.

General Form
CALL BGSUB(gdsname[,corrval][,key][,gencode])

gdsname,gencode
: are the same as described in "Arguments Used by Many GSP Subroutines" except that a value of three cannot be assigned as the "gencode" argument.

corrval
: is a constant or variable the value of which identifies the buffer subroutine begun by this call. This value is assigned by the programmer.

key
: is an integer variable that identifies the buffer subroutine begun by this call. Its value, or the key, is assigned by GSP. This key may only be used in a call to the RESET subroutine. A new key is assigned when the ENSUB subroutine is called and must be used from that point on.

CAUTION: No buffer subroutine associated with a particular graphic data set may be defined within another buffer subroutine or sequence associated with the same graphic data set. This means that the ENSUB subroutine must be called within a graphic data set before a second call within that same graphic data set can be issued to the BGSUB subroutine.

ENSUB--End a Buffer Subroutine

The ENSUB subroutine designates that the buffer subroutine begun by a call to the BGSUB subroutine is complete. Once the ENSUB subroutine is called, the buffer subroutine created consists of all elements within the specified graphic data set that have been generated prior to this call and after a preceding call to the BGSUB subroutine for the specified graphic data set.

General Form
CALL ENSUB(gdsname[,key])

gdsname
: is described in "Arguments Used by Many GSP Subroutines."

key
is an integer variable the value of which identifies the buffer subroutine terminated by this call. This value (called the key) is assigned by GSP. This key replaces any key assigned to this buffer subroutine in the call to the BGSUB subroutine that began the buffer subroutine. For additional information on the specification of this argument, refer to "Caution" and "Programming Considerations" in the description of the ENSEQ subroutine.

LKSUB--Link to a Buffer Subroutine

The LKSUB subroutine generates linkage to a buffer subroutine. The buffer subroutine must have been generated prior to the time this subroutine is called. A light pen detect on any portion of the images produced by a buffer subroutine makes available the key or correlation value of the linkage. The element that produced the detected image is not identified.

General Form
CALL LKSUB(gdsname[,bufcorrval][,bufkey][,linkcorrval][,linkkey] [,gencode])

gdsname,gencode
are as described in "Arguments Used by Many GSP Subroutines" except that a value of three cannot be assigned as the "gencode" argument.

bufcorrval
is a constant or variable the value of which identifies the buffer subroutine to be executed. This value must be the same as was assigned by the programmer as the correlation value in the call to the BGSUB subroutine that began the buffer subroutine. If this argument is not specified, the "bufkey" argument must be specified.

bufkey
is an integer variable the value of which identifies the buffer subroutine to be executed. This value must be the same as was assigned by GSP as the key value in the call to the ENSUB subroutine that terminated the subroutine. If this argument is not specified, the "bufcorrval" argument must be specified.

linkcorrval
is a constant or variable the value of which identifies the linkage generated by this call. This value is assigned by the programmer.

linkkey
is an integer variable the value of which identifies the linkage generated by this call. This value is assigned by GSP.

Example of Use of a Buffer Subroutine

Figure 11 shows the use of the BGSUB, ENSUB, and LKSUB subroutines to display six squares on a diagonal line across the screen. Each of the squares is the same size. The orders and data for displaying one square is generated within a buffer subroutine identified as NSQURE. This subroutine is then called six times to produce the six squares.

```
10   DIMENSION NULL(1)
20   NULL(1) = -5
30   CALL INGSP(IGSP,NULL)
40   CALL INDEV(IGSP,40,IGRAFD)
50   CALL INGDS(IGRAFD,IGDS1)
60   CALL SDATL(IGDS1,0.0,0.0,1000.0,1000.0)
70   NSQURE = 10
80   CALL BGSUB(IGDS1,NSQURE)
90   CALL MVPOS(IGDS1,25.0,25.0)
100  CALL PLINE(IGDS1,0.0,-50.0)
110  CALL PLINE(IGDS1,-50.0,0.0)
120  CALL PLINE(IGDS1,0.0,50.0)
130  CALL PLINE(IGDS1,50.0,0.0)
140  CALL MVPOS(IGDS1,-25.0,-25.0)
150  CALL ENSUB(IGDS1)
160  X1 = 0
170  DELTA = 150.0
180  DO 210 I = 1,6
190  X1 = X1 + DELTA
200  CALL STPOS(IGDS1,X1,X1)
210  CALL LKSUB(IGDS1,NSQURE,NULL,I)
220  CALL EXEC(IGDS1)
       .
       .
       .
```

10,20	Designate storage for a variable named NULL and assign a value of -5 to that variable. Since NULL is later identified as the null variable, the value of -5 indicates that the program is to be abnormally terminated and a dump is to be produced only when a return code of five is made available by a GSP subroutine.
30,40,50	Establish communication links among the program, GSP, a 2250, and a graphic data set. Designate NULL as the null variable.
60	Defines the input data limits of graphic data set IGDS1. No other option definition subroutines are called for IGDS1. Subsequently, the default conditions of all other option definition subroutines are assumed.
70	Assigns a correlation value of 10 to the square.
80	Specifies the start of buffer subroutine NSQURE.
90	Moves the 2250 beam to what will be the upper right corner of the square. Subsequent repositioning of the square will be defined in terms of the center of the square.
100,110, 120,130	Create the elements to draw the four sides of the square. Each side is 50 units long.
140	Moves the 2250 beam to the center of the square. This facilitates maintenance of the beam position during program execution. (Note: This positioning is not required if the STPOS subroutine is used between calls to the LKSUB subroutine.)
150	Ends buffer subroutine NSQURE.

Figure 11. Example of Use of a Buffer Subroutine (Part 1 of 2)

160,170, 180	Establish a DO loop with a range through statement 210.
190	Specifies the position of the center of the square.
200	Moves the 2250 beam to the position where the next square is to be displayed. This position is the center of the square. Since the same value is used for both the x- and y-coordinate, the squares displayed after six executions of this DO loop appear on a diagonal line from 0,0 to 1000,1000.
210	Creates linkage to buffer subroutine NSQURE. Each successive execution of this DO loop assigns a correlation value of 1 through 6 respectively to the elements that generate the six squares.
220	Causes the six squares to be displayed.

Figure 11. Example of Use of a Buffer Subroutine (Part 2 of 2)

IMAGE CONTROL SUBROUTINES

Controlling the display of images involves designating when an image is to be displayed and what is to be displayed. The subroutines available for this purpose, called the image control subroutines, are described in the sections that follow.

REPLACING AND ELIMINATING ELEMENTS WITHIN A GRAPHIC DATA SET

At any point within a program, the programmer may choose to replace elements within a graphic data set with new elements, or may choose to eliminate elements from a graphic data set. This section describes the facilities and subroutines available for this purpose. They are:

- The Update Facility.
- The Reset a Graphic Data Set (RESET) subroutine.
- The Indicate Beam Position (IDPOS) subroutine.

If the data to be modified constitutes an entire graphic data set or is at the end of a graphic data set, the use of the RESET subroutine followed by appropriate calls to image generation subroutines is an efficient method of modifying the data. In this regard, a portion of a graphic data set that is to be frequently modified should be generated so that it is placed at the end of the graphic data set.

However, this is not always conveniently possible. For example, the first line of more than one hundred lines in a graphic data set may require modification. In this case, it is inefficient to eliminate the element used to display that line and all elements thereafter within the graphic data set and then generate all the removed elements again. It would be more efficient to use the update facility and merely replace the desired element by issuing one call to an image generation subroutine.

CAUTION: Any time the sequence in which elements are created within a graphic data set is interrupted (such as in updating and resetting), the desired starting position must be reestablished prior to calling an image generation subroutine for creating the next element to be placed within that graphic data set. In the case of updating, this must be

done prior to the update call and prior to adding a new element to the graphic data set after the update call. In the case of resetting, this must be done prior to generating a new element after the reset operation has been performed. If this is not done, images associated with the newly created elements may appear in different locations than desired and scissoring may occur at an undesirable time during execution of the program.

Reestablishing the desired starting position for new elements is accomplished in one of the following ways:

- By a call to the STPOS subroutine, or
- By a call to the IDPOS subroutine if the beam position is known and that position is to be the starting position of the new element.

The Update Facility

An element that has been keyed or correlated can be modified at any time by means of the update facility. This facility involves calling the subroutine that originally created a particular element to generate a new element and substitute it for that previously created element. The element being updated must have already been transferred to the 2250 buffer via the EXEC subroutine. The update element is automatically transferred to the buffer after it is created; therefore, the EXEC subroutine need not be recalled.

A key or correlation value must be used to identify the element to be updated. An element modified by an update call will be in the same status (include or omit) as the previously created element it has replaced.

To insure successful performance of the update facility, the following conditions must be adhered to:

1. The update element must be created by a call to the same image generation subroutine that created the element being replaced. That is, the PPNT subroutine must be called to update an element created by a prior call to the PPNT subroutine; the PLINE subroutine must be called to update an element created by a prior call to the PLINE subroutine; and so forth.

2. The output data making up the update element must be of the same form as the output data comprising the element being replaced. That is, an element consisting of absolute output can only be updated by an element consisting of absolute output; a text element that produces basic size, protected characters can only be updated by a text element that will produce basic size, protected characters; and so forth.

3. The amount of data generated by the update call must not exceed the amount of data originally generated for the element being updated. If it does, an appropriate return code is made available and the element to be updated is left unchanged. This condition, called update data overflow, is likely to occur when scissoring has been performed on the element to be updated, or when scissoring will be required on the update element.

   If an update element consists of less data than the element it is replacing, the entire first element is replaced. However, the amount of data that can be substituted by future updates of that same element is based on the amount of data that was generated by the call that originally created the element. For example, if an element consisting of 39 words of data were replaced by an update element of 21 words of data, that update element may in turn be

replaced by another update element consisting of from 1 through 39 words of data.

4. Prior to issuing the update call, the programmer must inform GSP of the absolute starting location to be used for creating the update element. Prior to adding a new element to the graphic data set after an update call, the programmer must inform GSP of the absolute starting location to be used for creating the new element. These operations are accomplished by calling the STPOS or IDPOS subroutine.

To use the update facility, the programmer must include the following in his program (see Figure 7 for an example):

1. The "key" or "corrval" argument, or both, in the call to the subroutine that generates the element that will later be updated. In this case, the "key" or "corrval" argument identifies the element that is generated by the call. If the "gencode" argument is specified in this call, it must not have a value of three.

2. The "key" or "corrval" argument, and a value of three as the "gencode" argument in the call to the subroutine that generates the element to be substituted for a previously generated element. In this case, the "key" or "corrval" argument identifies the previously generated element that is to be replaced.

If both the "key" and "corrval" arguments are specified in an update call, the "key" argument is used to identify the element to be updated and the value of the "corrval" argument is assigned as a new correlation value to the updated element. A value of zero assigned to the "corrval" argument causes removal of the correlation value previously associated with the element being updated.

If only the "key" argument is specified in an update call, the value of that argument is used to identify the element to be updated. The updated element remains associated with the correlation value (if any) with which it had been previously associated.

If only the "corrval" argument is specified in an update call, the value of that argument is used to identify the element to be updated and remains as the correlation value of the updated element. If several elements have been assigned the same correlation value, the first element that was assigned that correlation value is assumed to be the element that is to be updated.

CAUTION: When using the update facility, the programmer must make certain that the updating is performed successfully (i.e., it does not violate any of the conditions described in "The Update Facility") or must include an alternate routine in his program that uses the RESET subroutine to perform the same modification and that assumes control if the update attempt is unsuccessful.

PROGRAMMING CONSIDERATION: If a text element containing a cursor is updated, that cursor is overlaid by the update element.

RESET--Reset a Graphic Data Set

The RESET subroutine removes a particular sequence, buffer subroutine, or element and all elements that follow it from a graphic data set, or removes all elements from a graphic data set. When this subroutine is called, the display of the image produced by the removed elements is terminated and all 2250 buffer storage occupied by the removed elements is freed. All keys or correlation values associated with the removed elements are also permanently erased.

If only part of a graphic data set is to be removed by the RESET subroutine, the first sequence, buffer subroutine, or element to be removed must be identified by a key or correlation value. If neither a key nor correlation value is specified, all elements within the specified graphic data set are removed.

The removal of all elements in a graphic data set does not terminate the use of that graphic data set. New elements can be generated for that graphic data set until the use of the data set is terminated by one of the termination subroutines. However, prior to adding a new element to the graphic data set after the reset operation has been performed, GSP must be informed of the desired starting position of that new element. This is accomplished via the STPOS or IDPOS subroutine.

General Form
CALL RESET(gdsname[,corrval][,key])

gdsname
is described in "Arguments Used by Many GSP Subroutines."

corrval
is a constant or variable the value of which has been correlated with the first sequence, buffer subroutine, or element that is to be removed from the specified graphic data set.

key
is an integer variable the value of which is the key that identifies the first sequence, buffer subroutine, or element that is to be removed from the specified graphic data set.

PROGRAMMING CONSIDERATIONS: If it is desired to redisplay images associated with elements that have been removed from a graphic data set by the RESET subroutine, those elements must be recreated by calls to the image generation subroutines.

IDPOS--Indicate Beam Position

The IDPOS subroutine provides GSP with a starting location from which other x- and y-coordinates are to be computed. This starting location is also used as a basis for applying scissoring options as specified by the SSCIS subroutine. The IDPOS subroutine does not create orders and data. It, or the STPOS subroutine in lieu of it, must be called prior to issuance of an update call and before new elements are created within a graphic data set after an element has been updated or a reset within that graphic data set.

General Form
CALL IDPOS(gdsname,xlast,ylast[,xcurr,ycurr])

gdsname
is described in "Arguments Used by Many GSP Subroutines."

xlast,ylast
are constants or variables the values of which designate the positions that are to be used as the absolute starting points from which other x- and y-coordinates are to be computed. These values must be absolute no matter what has been specified by the SDATM subroutine. They may be either integer or real as defined by the

SDATM subroutine. (Note: If the values happen to be incremental the IDPOS subroutine uses them as if they were absolute.)

xcurr,ycurr
are constants or variables the values of which designate the current beam position after the "xlast" and "ylast" arguments have been scissored. These values must be absolute and may be real or integer as defined by the SDATM subroutine. These arguments must be specified if the "xlast" and "ylast" arguments represent values that have been scissored. If these arguments are omitted, their values are assumed to be the same as the values of the "xlast" and "ylast" arguments respectively.

CONTROLLING WHEN IMAGES ARE DISPLAYED

This section describes the subroutines that cause images to be displayed or removed from the 2250 screen. These subroutines initiate the processing of elements, thus producing a display; and place elements, sequences, buffer subroutines, or graphic data sets in include or omit status, thus altering a display. The subroutines described are as follows:

- Execute (EXEC)
- Place in Include Status (INCL)
- Place in Omit Status (OMIT)
- Order Graphic Data Sets (ORGDS)

The status of a graphic data set, sequence, element, buffer subroutine, or buffer subroutine linkage may be altered from include to omit status, and vice versa, as often as desired. Altering the status of one of the above does not affect the status of the others. However, it does determine whether or not images associated with the others are displayed in the following hierarchy: (1) graphic data set, (2) sequence or buffer subroutine, and (3) element.

If a graphic data set is placed in omit status, images associated with sequences, buffer subroutines, or elements within that graphic data set are not displayed regardless of their status. An image associated with an element is displayed only if the graphic data set and sequence or buffer subroutine which contain it, and the element itself, are in include status. Images associated with a sequence or buffer subroutine are displayed only if the graphic data set and the sequence or buffer subroutine itself is in include status. For a discussion of what occurs when various types of elements (graphic, positioning, text, and end-order-sequence) are in include status, refer to the introduction to the section "Image Generation Subroutines."

EXEC--Execute

The EXEC subroutine must be called to display an image. It causes all elements within a particular graphic data set that have been generated since a previous call to the EXEC subroutine for that graphic data set to be transferred to the 2250 buffer. Images associated with any of these transferred elements in include status are displayed at this time. The first call to the EXEC subroutine causes the display of all elements in include status within the specified graphic data set that have been generated since the data set was initialized.

The EXEC subroutine does not affect the include or omit status of elements, sequences, buffer subroutines, buffer subroutine linkages, or graphic data sets. These items may be changed from include status to

omit status, and vice versa, merely by calling the INCL or OMIT subroutine. The EXEC subroutine need not be reissued.

A call to the EXEC subroutine also is not necessary in the case of update elements. Such elements are automatically transferred to the buffer and executed immediately after they are created.

General Form
CALL EXEC(gdsname)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

INCL--Place in Include Status

The INCL subroutine places a graphic data set, a keyed or correlated sequence, a keyed or correlated buffer subroutine, keyed or correlated buffer subroutine linkage, or a keyed or correlated element in the include status. Elements that have not been keyed or correlated are always placed in include status when they are created. If the item referred to in a call to this subroutine is already in include status, the call is ignored.

When a graphic or text element in include status is executed, the image associated with that element is displayed on the screen. When a positioning element in include status is executed, the 2250 beam is moved as directed by that element; the beam is blanked so no image appears.

General Form
CALL INCL(gdsname[,corrval][,key])

gdsname,corrval,key
are described in "Arguments Used by Many GSP Subroutines."

CAUTION: Before a graphic data set can be placed in include status by this subroutine, at least one element must have been created for it.

OMIT--Place in Omit Status

The OMIT subroutine places a graphic data set, a keyed or correlated sequence, a keyed or correlated buffer subroutine, keyed or correlated buffer subroutine linkage, or a keyed or correlated element in omit status. Elements not previously keyed or correlated cannot be placed in omit status. If the item referred to in a call to this subroutine is already in omit status, the call is ignored.

When a graphic or text element in omit status is executed, the image associated with that element is not displayed on the screen. However, for a graphic element, the 2250 beam is moved as designated by that element; for a text element, the 2250 beam is not moved. When a positioning element in omit status is executed, the 2250 beam also is not moved.

General Form
CALL OMIT(gdsname[,corrval][,key])

gdsname,corrval,key
are described in "Arguments Used by Many GSP Subroutines."

CAUTION: Before a graphic data set can be placed in omit status by this subroutine, at least one element must have been created for it.

ORGDS--Order Graphic Data Sets

The ORGDS subroutine modifies the sequence in which graphic data sets associated with a particular 2250 are regenerated within the buffer. This ordering has no effect on the status (include or omit) of each graphic data set in the sequence. Normally, graphic data sets are regenerated in the sequence in which elements were first created for them.

General Form
CALL ORGDS($gdsname_1$[,$gdsname_2$...,$gdsname_n$])

$gdsname_1$ through $gdsname_n$
are integer variables that identify graphic data sets involved in the ordering. A particular graphic data set must be identified only once in a given call. The "gdsname" arguments indicate the following:

- "$gdsname_2$" is to be regenerated immediately after "$gdsname_1$", "$gdsname_3$" is to be regenerated immediately after "$gdsname_2$", and so forth through "$gdsname_n$" (see Statement 200 in the example that follows).

- All graphic data sets not specified in this call that are associated with the same 2250 as the specified graphic data sets are regenerated after the last graphic data set specified in this call. These graphic data sets are regenerated in the same relative order to one another as they were prior to this call (see Statements 300 and 400 in the example that follows).

PROGRAMMING CONSIDERATIONS: The ordering of graphic data sets is useful in light pen attention-handling. For example, if a light pen were directed at a point where two lines intersect and if each of these lines are within a different graphic data set (e.g., IGDS1 and IGDS2 in that order) associated with the same 2250, the attention is queued for the graphic data set that appears earliest in the sequence of graphic data sets (in this case, IGDS1). If the sequence were ordered so that IGDS2 appears before IGDS1, the attention is then queued for IGDS2 which is now the earliest appearing graphic data set in the sequence.

EXAMPLE: The following example depicts the ordering of graphic data sets by means of the ORGDS subroutine:

```
70  CALL  STPOS(IGDS1,200,100)  ----------> Places these four graphic
80  CALL  STPOS(IGDS2,100,100)              data sets in the following
90  CALL  STPOS(IGDS3,50,50)                sequence: IGDS1,IGDS2,IGDS3,
100 CALL  STPOS(IGDS4,200,200)              IGDS4. (Assumes that these
      .                                     are the first calls to image
      .                                     generation subroutines for
      .                                     each graphic data set.)
      .
200 CALL  ORGDS(IGDS4,IGDS3,IGDS2,IGDS1)--> Orders the four graphic data
      .                                     sets so that they are in the
      .                                     following sequence: IGDS4,
      .                                     IGDS3,IGDS2,IGDS1.
      .
```

```
300 CALL  ORGDS(IGDS2,IGDS3)----------> Orders the sequence estab-
     .                                  lished in Statement 200 so
     .                                  that the graphic data sets
     .                                  appear as follows: IGDS2,
     .                                  IGDS3,IGDS4,IGDS1.
     .
400 CALL  ORGDS(IGDS4)----------------> Orders the sequence estab-
                                        lished in Statement 300 so
                                        that the graphic data sets
                                        appear as follows: IGDS4,
                                        IGDS2,IGDS3,IGDS1.
```

KEYBOARD INPUT AND BUFFER DATA ANALYSIS SUBROUTINES

This section describes the subroutines that allow text data to be entered from the alphameric keyboard and then read into main storage for analysis. The subroutines described are Insert Cursor (ICURS), Remove Cursor (RCURS), and Read Data (GSPRD). The GSPRD subroutine may also be used to read into main storage graphic orders and data other than that entered from the alphameric keyboard.

ICURS--Insert Cursor

The Insert Cursor (ICURS) subroutine inserts a cursor into a graphic data set. The cursor is a symbol that is represented on the screen as a dash beneath a character position. Only one cursor may be associated with a 2250 at any time. An attempt to insert a second cursor causes the previously displayed cursor to be repositioned to the location specified for the second cursor.

The cursor marks the position on the screen at which the next character entered from the alphameric keyboard will appear. It also serves as a delimiter that terminates the reading of text from the 2250 buffer into main storage.

A cursor must be displayed on the screen whenever text is to be entered into the 2250 buffer from the alphameric keyboard. Text is read from the buffer by means of the GSPRD subroutine.

The ICURS subroutine inserts a cursor at the beginning of the first text element within a keyed or correlated sequence, at the beginning of a keyed or correlated text element, or at any specified character position within a keyed or correlated text element.

The cursor appears on the screen only when it is inserted into a text element that is in the include status and is being executed. The characters of the text element may be protected or unprotected. However, data can be entered from the alphameric keyboard only when the cursor appears in an unprotected area.

When the ICURS subroutine is called to insert a cursor within an equivalent graphic data set, the cursor will not be inserted until the next call to the EXEC subroutine for that graphic data set.

General Form
CALL ICURS(gdsname[,corrval][,key][,charpos])

gdsname
: is described in "Arguments Used by Many GSP Subroutines."

corrval
is a constant or variable the value of which has been correlated with the sequence or element within the graphic data set into which the cursor is to be inserted. If this argument is not specified, the null variable must be substituted for it and the "key" argument must be specified.

key
is an integer variable the value of which is the key that identifies the sequence or element within the graphic data set into which the cursor is to be inserted. If this argument is not specified, the "corrval" argument must be specified.

charpos
is an integer constant or integer variable the value of which specifies the character position within a text element where a cursor is to be inserted. This value must not exceed the number of alphameric characters in the text element. If it does, no cursor will be inserted and a return code indicating an invalid parameter specification will be made available. The position of the first character in a text element is considered to be one.

PROGRAMMING CONSIDERATIONS: During testing of a GSP program that requests that characters previously created by the PTEXT subroutine be overlaid with characters entered from the alphameric keyboard, additional characters occasionally cannot be entered in the prescribed area without first depressing the JUMP key on the alphameric keyboard. This results from the GDOA becoming full and its contents being transferred to the buffer before all the characters were created. To override this condition, increase the size of the GDOA for that graphic data set by respecifying the "gdoalength" argument in the call to the INGDS subroutine that initialized the graphic data set.

RCURS--Remove Cursor

The RCURS subroutine removes a cursor from a graphic data set.

General Form
CALL RCURS(gdsname)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

EXAMPLE: The statements below cause a cursor to be inserted into a graphic data set, and then later removed from that graphic data set. In the example, NO is used as the null variable.

Statement 50 causes a cursor to be inserted under the third character of a text string in a text element identified by the value assigned by GSP to IKEY. This text element is part of graphic data set IGDS1.

Statement 90 causes the inserted cursor to be removed from the graphic data set.

```
50  CALL  ICURS(IGDS1,NO,IKEY,3)
      .
      .
      .
90  CALL  RCURS(IGDS1)
```

GSPRD--Read Data

The GSPRD subroutine causes graphic orders and data or alphameric characters to be read from the 2250 buffer to a main storage array. The read initiated by this call may be terminated when a cursor is encountered, when the specified amount of data has been read, or when the end of the graphic data set or the element being read is encountered.

Data is placed in the array in A format, four characters left-aligned in a word. The characters are represented in extended binary-coded-decimal interchange code (EBCDIC).

General Form
CALL GSPRD(gdsname,storageloc,count,rdtype[,termcode][,$corrval_1$] [,key_1][,$corrval_2$][,key_2])

gdsname
is described in "Arguments Used by Many GSP Subroutines."

storageloc
is the variable or array name that identifies the beginning of the main storage array where the data being read is to be placed.

count
is an integer constant or integer variable that defines the amount of data to be read. The value assigned to this argument must be given in terms of storage locations (bytes). This value may be specified as a positive value or as a negative value having the following meanings:

Positive-- The read is terminated after whichever of the following conditions occurs first:

- The specified amount of data has been read.
- The end of the graphic data set or the last element being read is encountered.

Negative-- The read is terminated as described for a positive value, unless a cursor is detected first. When the cursor is detected, the read is terminated after the contents of the buffer location immediately preceding the location containing the cursor has been read.

rdtype
is an integer constant or integer variable that identifies what is to be read as a result of this call. This constant or variable must have one of the following values:

1 If text information is to be read. (Note: Both protected and unprotected characters will be read.)

2 If all orders and data are to be read.

termcode
is an integer variable the value of which is assigned by the GSPRD subroutine to indicate what caused the read operation to be terminated. The value and its meaning are as follows:

Value	Meaning
0	The amount of data defined by the "count" argument has been read.
+	The end of the graphic data set or the element being read was encountered. The value is the number of bytes of data read.
-	A cursor was encountered. The value is the number of bytes of data read.

corrval$_1$
is a constant or variable the value of which has been previously correlated with the first or only element to be read. If more than one element has been correlated with the value specified by this argument, the read begins at the first element correlated with that value.

key$_1$
is an integer variable the value of which is the key that identifies the first or only element to be read.

corrval$_2$
is a constant or variable the value of which has been previously correlated with the last element of a series of elements to be read.

key$_2$
is an integer variable the value of which is the key that identifies the last element in a series of elements to be read.

PROGRAMMING CONSIDERATIONS: Since the number of bytes of data actually read may be less than the number of bytes specified by the "count" argument, the "termcode" argument should also be specified and the value assigned to it by the GSPRD subroutine tested before the data that was read is to be used.

GSP provides subroutines that enable two-way communication between the 2250 operator and the program. The program communicates with the operator by placing a message on the 2250 screen. It may also provide the means by which the operator can respond to the message by making the light pen or specific programmed function keys available for use. The operator communicates with the program by entering information from the alphameric keyboard, by depressing a programmed function key, or by pointing the light pen at an image on the screen and activating that light pen. Any such response causes an attention.

An attention is an interruption that causes the program to change its course at an unpredictable point. When an attention occurs, processing of the user's program is interrupted and control is passed to GSP to process the attention. This processing results in information about the attention being made available to the user's program upon request.

How the attention information is to be handled must be specified by the programmer. The information may be ignored, or may be held in a queue (queued) for later processing upon request. The processing is done by routines written by the programmer.

There are thirty-five attention sources available to the GSP user. These are:

- Thirty-two programmed function keys.
- The END key of the alphameric keyboard.
- The light pen.
- An end-order-sequence order.

The alphameric keyboard CANCEL key is only available for system functions. Its use is described in detail in "Using the Alphameric Keyboard CANCEL Key."

An end-order-sequence order is not available to the 2250 operator. It is used by the programmer to control display regeneration (see the description of the STEOS subroutine).

The programmer permits the queueing and processing of attention information from a particular source by enabling that source. He causes the attention information to be ignored by disabling the source. GSP provides subroutines for both enabling and disabling attention sources.

GSP permits one or several enabled attention sources for a 2250 be grouped into sets according to how and when they are used in the program. These sets are called attention levels. Only one level, called the active attention level, is able to accept attention information at any given time. All other levels are called inactive attention levels (see "Using Multiple Attention Levels").

Creating an Attention Level

An attention level is established by a call to the Create an Attention Level (CRATL) subroutine and remains in existence either until its use is terminated by a call to the End an Attention Level (ENATL) subroutine or until the use of its associated 2250 is terminated.

When an attention occurs from an enabled source, attention information is placed in the queue for the active attention level and processed as directed by the programmer. When the use of an attention level is terminated, all attention information in the queue for that level is deleted and the storage occupied by that level is freed.

Generally, the use of one attention level for a 2250 should be terminated before another level is created for that 2250. However, in some cases, it may be desirable to have several attention levels for a 2250 exist simultaneously throughout a period of time. This use of multiple attention levels is described in the section "Using Multiple Attention Levels."

Enabling and Disabling Attention Sources

When an attention level is created, all attention sources for it except the alphameric keyboard CANCEL key are disabled. The CANCEL key always remains enabled. Any attempts by the programmer to disable it are ignored.

Each desired attention source must be enabled by the programmer by a call to the Enable Attention Sources (ENATN) subroutine. Attention information from disabled sources is ignored. Attention sources may be repeatedly enabled or disabled as long as the level associated with them exists.

Except for the light pen attention source, the enabling of an attention source causes all attentions from that source to be accepted by GSP. To process light pen attentions, the programmer must enable the light pen attention source and designate those graphic data sets for which he wishes light pen attentions to be accepted. Light pen attentions occurring on undesignated graphic data sets are ignored by GSP, thus preventing unwanted light pen attentions from being processed. The designation of acceptable light pen attention sources is accomplished by calls to the Set Light Pen Attentions (SLPAT) subroutine.

Placing Attention Information in an Attention Level Queue

As mentioned previously, when an attention occurs from an enabled source for an active attention level, the user's program is interrupted and attention information is queued for that level. For a light pen attention, the attention information is queued only if the graphic data set associated with the image at which the light pen was directed has been designated as an acceptable source by a call to the SLPAT subroutine.

The queued attention information contains the identification of the source that caused the attention. If the source was the light pen, a programmed function key, or an end-order-sequence order, additional information about that attention is also provided. However, for a light pen or end-order-sequence attention, character code and coordinate information is only queued if the Modify Light Pen or End-Order-Sequence Attention Information (MLPEO) subroutine had been previously called for the active attention level.

Once the attention information is queued, control is returned to the user's program at the point where that program was interrupted. The program is not notified that an attention has occurred until attention information is requested by a call to the Request Attention Information (RQATN) subroutine.

Using the Alphameric Keyboard CANCEL Key

The alphameric keyboard CANCEL key is reserved for direct communication between the 2250 operator and the operating system. The key is

automatically enabled as soon as its associated 2250 is initialized. It is to be used when the operator has recognized a condition in his program that warrants interruption.

When the CANCEL key is depressed, an image appears on the screen that gives the 2250 operator the following options:

- TERMINATE -- Terminate the user's program without producing a dump.
- DUMP -- Terminate the user's program and produce an abnormal termination dump.
- RESUME -- Restore the user's program as it was before the CANCEL key attention occurred and cause processing to be resumed.

The 2250 user selects one of these options by using the light pen. Abnormal termination dumps are described in the publication IBM System/360 Operating System: Programmer's Guide to Debugging, Form C28-6670.

Using Multiple Attention Levels

As mentioned previously, the use of a single attention level is normally sufficient. However, in some cases, the use of multiple attention levels may be desirable.

Each level is created by a separate call to the Create an Attention Level (CRATL) subroutine. Attention sources may be enabled and disabled for a particular level at any time as long as the level exists.

As the term "level" implies, there is a strict hierarchy involved among the groups of attention sources. As a result, even though many attention levels may exist simultaneously for a particular 2250 throughout a period of time, only one level will be able to accept attention information at any given time. This is called the active attention level and is the lowest attention level in the hierarchy. All other attention levels for that 2250 are inactive.

When an attention level is created for a 2250, it is made the lowest attention level (the active level) in the hierarchy for that 2250. This newly created level remains active either (1) until its use is terminated by the call to the End an Attention Level (ENATL) subroutine, (2) until a new attention level is created for the same 2250, or (3) until the order of the attention level hierarchy for the 2250 is changed by a call to the Modify Position of an Attention Level (MPATL) subroutine. An inactive level becomes active again whenever it reappears at the bottom of the hierarchy.

When an attention level becomes inactive as a result of the creation of a new level or the reordering of the attention level hierarchy, any attention information associated with it that had not yet been processed remains in the attention queue for that level. No new attention information can be added to that queue until the level is made active again. However, any information currently in the queue may be requested and processed at any time by a call to the Request Attention Information (RQATN) subroutine. When an attention level is ended all attention information in the queue for that level is deleted and the storage occupied by that level is freed.

Attention sources may be enabled and disabled at any time during the existence of the level associated with them, even when the associated level is inactive. When an inactive attention level is reactivated, thecurrent status of its attention sources (enabled or disabled) becomes effective immediately.

An example of the use of multiple level attention handling is included in Appendix G.

THE ATTENTION RELATED SUBROUTINES

This section describes the attention related subroutines available for communication between the program and the 2250 operator. These subroutines are as follows:

- Create an Attention Level (CRATL)
- End Attention Levels (ENATL)
- Enable Attention Sources (ENATN)
- Disable Attention Sources (DSATN)
- Set Light Pen Attentions (SLPAT)
- Request Attention Information (RQATN)
- Modify Light Pen or End-Order-Sequence Attention Information (MLPEO)
- Modify Status of the Programmed Function Indicator Lights (MLITS)
- Modify Position of an Attention Level (MPATL)
- Sound Audible Alarm (SALRM)

CRATL--Create an Attention Level

The CRATL subroutine establishes an active attention level for a particular 2250. When an attention level is created, no attention sources are enabled for it. Therefore, all attentions that occur will be ignored until the ENATN subroutine is called.

General Form
CALL CRATL(devicename,attnlevel[,dequectl])

devicename
is described in "Arguments Used by Many GSP Subroutines."

attnlevel
is an integer variable identifying the attention level created by this subroutine. The CRATL subroutine assigns a value to this variable.

dequectl
is an integer variable that defines when attention information is to be dequeued after it is made available to the user by the RQATN subroutine. This argument need only be specified when multiple attention levels are used. If it is omitted, attention information is dequeued immediately after it is made available to the programmer. If the "dequectl" argument is specified, it must have one of the following values:

1 Dequeue attention information as it is made available.

2 Dequeue attention information as it is made available if the associated level is active when this subroutine is called. Do

not dequeue that attention information if the associated level is inactive when this subroutine is called.

EXAMPLE: Refer to the sample program in Appendix A for an example of the use of the CRATL subroutine.

ENATL--End Attention Levels

The ENATL subroutine is used to perform one of the following:

- Terminate the use of a specified attention level and all levels lower than it in the attention level hierarchy for a 2250. Any level immediately higher than the specified level is now made active.
- Terminate the use of all attention levels lower than a specified attention level thus making the specified level active.
- Terminate the use of the only attention level created for a 2250 or the last attention level in the attention level hierarchy.

General Form
CALL ENATL(attnlevel[,rangecode])

attnlevel
is an integer variable having the same value as was returned as the "attnlevel" argument in the call to the CRATL subroutine that created the attention level to be terminated.

rangecode
is an integer constant or integer variable that designates the operation to be performed by this call. This argument need only be specified when multiple attention levels are used. It may have one of the following values:

1 To terminate the specified attention level and all attention levels lower than it in the attention level hierarchy. Any attention level higher than the specified attention level is made active.

2 To terminate all attention levels lower than the specified attention level and make the specified attention level active.

If this argument is omitted, the specified attention level and all levels lower than it are terminated.

EXAMPLE: Refer to Appendix G for an example of the use of the ENATL subroutine.

ENATN--Enable Attention Sources

The ENATN subroutine designates attention sources to be processed at a particular attention level and removes previous attention information from those sources from the queue for that level. It may be called at any time after the designated attention level has been created, whether or not the attention level is active. Enabled attention sources for an attention level can be disabled at any time by a call to the DSATN subroutine.

Once attention sources are enabled for a particular attention level, as long as that level is active, attention information from the enabled sources is queued for it while attention information from sources not enabled is ignored. No attention information is queued for inactive attention levels. Once attention information has been queued for an attention level, it can be requested at any time (even if the attention level is inactive) by a call to the RQATN subroutine.

The ENATN subroutine may be called as often as desired to enable additional attention sources for an attention level. However, when a previously enabled attention source for an attention level is reenabled for that level, all previously queued attention information from that source is dequeued for the designated attention level.

General Information
CALL ENATN(attnlevel,attnsource[,attnsource...])

attnlevel
: is an integer variable having the same value as was returned as the "attnlevel" argument in the call to the CRATL subroutine that created the attention level for which the designated attention sources are to be enabled.

attnsource
: is an integer constant or integer variable the value of which is the identification code of an attention source (or the first or last sources of a range of attention sources) to be enabled for the specified attention level. This argument may be repeated as many times as necessary to enable desired attention sources. Its value must be positive except when it identifies the highest attention source in a range of attention sources.

  To designate a range of attention sources, a pair of "attnsource" arguments must be specified in the calling sequence as follows (see the example at the end of this description):

  - The argument that identifies the lowest attention source in the range must appear immediately before the argument that identifies the highest attention source in the range.
  - The argument that identifies the highest attention source in the range must be a negative value and must appear in the argument list immediately after the argument that identifies the lowest attention source in the range.

ATTENTION SOURCE IDENTIFICATION CODES: Attention sources and their identification codes are as follows:

0-31	Programmed function keys 0 through 31, respectively.
32	END key of the alphameric keyboard.
33	CANCEL key of the alphameric keyboard.
34	Light pen.
35	End-order-sequence order.

PROGRAMMING CONSIDERATION: The CANCEL key remains enabled as long as its associated 2250 remains initialized. It can only be used for system functions (See "Using the Alphameric Keyboard CANCEL Key"). Reenabling it by a call to this subroutine has no effect upon its previously enabled status or use. However, the ability to specify its identification code in calls to the ENATN subroutine is included to permit specification of an uninterrupted range of attention sources.

EXAMPLE: The example that follows enables attention sources 3, 5 through 10, and 15 for the attention level identified as LEVEL1:

```
CALL  ENATN(LEVEL1,3,5,-10,15)
```

DSATN--Disable Attention Sources

The DSATN subroutine causes future attentions from the designated attention sources to be disregarded for a particular attention level. Disabled attentions are ignored until they are enabled again by a call to the ENATN subroutine. However, any attention information from that source that had been queued prior to the time the source was disabled remains in the queue until it is removed either by reenabling the source or by a call to the RQATN subroutine.

The DSATN subroutine is the inverse of the ENATN subroutine. It can be called even if the designated attention level is inactive.

General Form
CALL DSATN(attnlevel,attnsource[,attnsource...])

attnlevel
: is an integer variable having the same value as was returned as the "attnlevel" argument in the call to the CRATL subroutine that created the attention level for which the designated attention sources are to be disabled.

attnsource
: is an integer constant or integer variable that identifies a single attention source or one source in a range of sources to be disabled for the specified attention level. This argument is to be specified in the same manner as described for the "attnsource" argument in the discussion of the ENATN subroutine.

PROGRAMMING CONSIDERATION: Attention source 33 (the CANCEL key) may be specified in a call to the DSATN subroutine. However, the request to disable this source will be ignored. The CANCEL key remains enabled as long as its associated 2250 is initialized (see "Using the Alphameric Keyboard CANCEL Key").

SLPAT--Set Light Pen Attentions

The SLPAT subroutine informs GSP whether or not to allow light pen attentions on the images associated with a given graphic data set. This subroutine thus allows the programmer to select those graphic data sets for which meaningful light pen attentions could occur, so that information from accidental light pen attentions will not be queued.

If the SLPAT subroutine is not called for a graphic data set, light pen attentions on that graphic data set are not accepted by GSP. Therefore, this subroutine must be called at least once in the program if the programmer wishes to process light pen attention information. The light pen attention source is enabled for processing by the ENATN subroutine.

General Form
CALL SLPAT(gdsname,detect)

gdsname
is described in the section "Arguments Used by Many GSP Subroutines."

detect
is an integer constant or integer variable the value of which describes the function to be performed by this subroutine, as follows:

1 Permit light pen attentions on the designated graphic data set.
2 Do not permit light pen attentions on the graphic data set.

RQATN--Request Attention Information

The RQATN subroutine enables the programmer to obtain attention information from an attention queue at any point in his program. By calling this subroutine, the programmer can determine if an attention has occurred and can identify its source, or can designate that execution of his program be suspended until an attention from a specific source occurs.

General Form
CALL RQATN(attnlevel,codeloc,wait[,arrayname],attnsource [,attnsource...])

attnlevel
is an integer variable having the same value as was returned as the "attnlevel" argument in the call to the CRATL subroutine that created the attention level for which attention information is requested.

codeloc
is an integer variable to which this subroutine will assign a value that identifies the source of the attention information being returned. The possible values that can be assigned are as follows:

-1 Programmed function key 0 caused the attention.
0 No attention has occurred from the designated sources.
1-31 A programmed function key caused the attention. The value assigned to this variable is the number of that key.
32 The END key of the alphameric keyboard caused the attention.
34 The light pen caused the attention.
35 An end-order-sequence order was encountered.

wait
is an integer constant or integer variable that designates whether or not execution of the user's program is to be suspended until an attention occurs from any of the attention sources designated in this CALL statement. This argument may be assigned one of the following values:

1 Return the first attention information queued for any attention source specified in the "attnsource" argument(s). If no information has been queued for any of these sources, return immediately to the user's program.

2 Return the first attention information queued for any attention sources specified in the "attnsource" argument(s). If no information has been queued for any of these sources and the attention level specified in this call is active, wait until

an attention occurs from one of the designated sources. This value must not be specified if attention information is requested from an inactive level. If it is, an argument error return code is made available, this request for attention information is ignored, and the next statement in the user's program is processed.

arrayname
is an array name identifying a ten-word array into which this subroutine is to place additional information about light pen or end-order-sequence attentions. This argument may also be used to have an overlay code returned for a programmed function keyboard attention. In this case, the overlay code is placed in the first word of the array and all other words in the array are unused. Therefore, the array need only be one word long. Table 5 lists the light pen information placed in the array. Except for word 10, this information also applies for end-order-sequence information.

attnsource
is an integer constant or integer variable that identifies one or the first or last source of a range of attention sources for which information is to be returned. This argument may be repeated as many times as necessary. The manner in which attention sources are specified by this argument is described in the discussion of the ENATN subroutine.

CAUTION: To have character code and coordinate information queued for attentions related to a specific attention level, the MLPEO subroutine must have been previously called for that attention level.

PROGRAMMING CONSIDERATION: If attention information is not to be dequeued when it is made available to the user, only information from the first attention that occurred from a particular source can be accessed by calls to the RQATN subroutine that refer to that source (see the "dequectl" argument in the CRATL subroutine).

Table 5. Contents of Array That Provides Light Pen Attention Information (Part 1 of 3)

Word	Contents
1	• The integer value that identifies the graphic data set associated with the image detected by the light pen. Note: For a programmed function keyboard attention, an integer value that identifies the overlay being used will be placed (right-aligned) in this word and none of the other words in the array will be used.
2	• The key assigned to the element associated with the image detected by the light pen. • A value of zero if the element associated with the image detected by the light pen has not been keyed or correlated.

Table 5. Contents of Array That Provides Light Pen Attention Information (Part 2 of 3)

3	• If the light pen detect was on a character and the element that generated the character was keyed, a number that designates the location of the character within the character string produced by the keyed element. For example: if ABC was produced by a keyed element and the light pen was pointed at C, this word would contain a value of three. • If the light pen detect was not on a character and the element that generated the detected image was keyed, the number of storage locations that the first graphic order for producing the detected image is from the beginning of the element. • If the element that produced the image detected by the light pen is not keyed, the number of storage locations that the first graphic order for producing the detected image is from the beginning of the graphic data set.
4	• The correlation value of the element associated with the image detected by the light pen. • A value of zero if the element associated with the image detected by the light pen has not been correlated.
5	• The key of the sequence associated with the image detected by the light pen. • A value of zero if the image detected by the light pen is not associated with a sequence.
6	• The number of storage locations that the graphic order for producing the detected image is from the beginning of the sequence. • A value of zero if word 5 is zero.
7	• The correlation value of the sequence associated with the image detected by the light pen. • A value of zero if the sequence had not been correlated, or if the image detected by the light pen is not associated with a sequence.
8 and 9	• The absolute coordinate values (x-coordinate in word 8 and y-coordinate in word 9) of the end point of a line detected by the light pen, or the absolute coordinate of the point or character detected by the light pen. • A value of zero if it had been specified that no coordinates are to be returned for the type of image that was detected by the light pen. Note: The coordinates placed in words 8 and 9 will be in the programmer coordinate system for the graphic data set identified in word 1 of this array. These coordinates will be real or integer as specified by the most recent call to the SDATM subroutine for that graphic data set, and will be scaled as designated by the most recent calls to the SDATL and SGDSL subroutines.

Table 5. Contents of Array That Provides Light Pen Attention Information (Part 3 of 3)

Word	Contents
	Caution: The values of the coordinates in words 8 and 9 may differ slightly from the values originally supplied by the user for the specific end point or character detected by the light pen. This results from any rounding of coordinate values that may be necessary to scale these values to raster units, and then reconverting the scaled raster units back to user coordinates when a light pen attention occurs. (Raster units are the standard measurement units used by the 2250 to position the 2250 beam.)
10	• The EBCDIC code of the character at which the light pen was pointed. This code is left aligned with the remainder of the word filled with zeros. • A value of zero if it had been specified that no character code is to be returned, or if the light pen was not directed at a character. Note: Word 10 refers only to characters produced by the character generator feature of the 2250. It is not used for end-order-sequence attentions.

Notes:

1. To have character code and coordinate information queued for attentions related to a specific attention level, the MLPEO subroutine must have been previously called for that attention level.

2. Integer values will be returned in words 1, 2, 3, 5, and 6. All other words will contain either real or integer values as previously set by the programmer. To access both real and integer information, two array names -- one integer and one real -- should be defined in EQUIVALENCE with each other.

3. If a value of two was specified as the "gdslevel" argument in the call to the INGDS subroutine for the graphic data set identified in word 1, only words 1, 3, 8, and 9 are applicable. Word 3 will always contain an integer value representing the logical buffer address of the image detected by the light pen or of the end-order-sequence order that was encountered. For additional information, see Appendix F.

MLPEO--Modify Light Pen or End-Order-Sequence Attention Information

The MLPEO subroutine designates whether character code or coordinate information, or both, is to be queued for light pen or end-order-sequence attentions for an attention level; and whether display regeneration is to be restarted after a light pen or end-order-sequence attention has occurred for the specified attention level. The subroutine is used to designate this type of information for attentions related to an attention level in the same manner as the SLPAT subroutine is used to designate that light pen attentions are to be accepted for a graphic data set.

If the MLPEO subroutine is not called for an attention level, the display is restarted after a light pen or an end-order-sequence attention occurs, and no character code or x- and y-coordinate information is queued in the user's attention information array. Table 5 shows

the normal contents of the return array for light pen and end-order-sequence attention information.

General Form
CALL MLPEO(attnlevel,attntype[,info][,restart])

attnlevel
is an integer varible having the same value as was returned as the "attnlevel" argument in the call to the CRATL subroutine for the attention level associated with this call.

attntype
is an integer constant or integer variable that designates whether this call pertains to light pen or end-order-sequence order attentions. It may have one of the following values:

1 for end-order-sequence attentions.
2 for light pen attentions.

If a value of one is specified, only values one and two can be specified as the "info" argument.

info
is an integer constant or integer variable that designates the type of information to be returned for either light pen or end-order-sequence attentions as designated by the "attntype" argument. It may have one of the following values:

1 Do not return coordinate or character code information.

2 Return the coordinates of the end point of the element detected by the pen, or where the 2250 beam was located when the end-order-sequence order was encountered.

3 Return the character code information for a light pen attention occurrence ("attntype" should be two).

4 Return both character code information (if the light pen was pointed at character) and the coordinates of the character detected by the light pen ("attntype" should be two).

If the "info" argument is not specified, it is assumed that neither coordinate nor character code information is to be returned.

restart
is an integer constant or integer variable that designates whether or not the display is to be started after the return of light pen or end-order-sequence attention information in response to a call to the RQATN subroutine. This argument may have one of the following values:

1 Designates that the display is to be restarted at the point of interruption.

2 Designates that the display is not to be restarted at the point of interruption. If this value is selected, the next call to the EXEC subroutine causes the display to be restarted.

If the "restart" argument is not specified in this call, the display is restarted at the point of interruption.

MLITS--Modify Status of the Programmed Function Indicator Lights

The MLITS subroutine defines which programmed function indicator lights are to be on and which are to be off at a given time. From the time a 2250 is initialized until specified otherwise by this subroutine, all indicator lights are off. This is the standard GSP default status.

The programmer can alter the GSP default status for a 2250 at any time by including the following in a call to the MLITS subroutine:

- The "devicename" argument.
- The "status" argument with a value other than one.

The new status selected as a result of this call now becomes the currently applicable default status for the specified 2250.

The programmer can override the currently applicable default status for a particular attention level by including the following in a call to the MLITS subroutine:

- The "attnlevel" argument.
- The "status" argument with a value other than one.

The keys designated by this call are lighted when the specified attention level becomes active. If another call to the MLITS subroutine is issued for this same attention level with a value of one as the "status" argument, the currently applicable GSP default status for the 2250 associated with this level is invoked.

General Form
CALL MLITS({devicename|attnlevel},status[,lights...])

devicename
: is an integer variable that identifies the 2250 for which a new default status is to be set. The value of this variable must be the same as was returned as the "devicename" argument in the call to the INDEV subroutine that initialized the 2250. If this argument is specified, the status indicated by this call applies to all attention levels for the specified 2250 except for those levels for which the MLITS subroutine is specifically called. Neither this argument, nor the null variable in lieu of it, is to be specified if the "attnlevel" argument is specified.

attnlevel
: is an integer variable that identifies the attention level associated with this call. The value of this variable must be the same as was returned as the "attnlevel" argument in the call to the CRATL subroutine for this attention level. Neither this argument, nor the null variable in lieu of it, is to be specified if the "devicename" argument is specified.

status
: is an integer constant or integer variable that specifies what is to be done by this call. This argument may have one of the following values:

  1 Use the currently applicable default status for the 2250 associated with the specified attention level. When this value is specified, the "attnlevel" argument must also be specified.

2 Turn off all lights.

3 Turn on those lights associated with enabled keys.

4 Turn on all lights specified by the "lights" argument.

lights
is an integer constant or integer variable the value of which is a number from 0 through 31 that identifies one of the programmed function indicator lights to be turned on by this call. This value must be positive except when it identifies the highest numbered indicator light in a range of indicator lights to be turned on. The "lights" argument may be repeated as many times as necessary to turn on all desired indicator lights. However, it can only be specified if a value of four is assigned as the "status" argument.

To designate a range of indicator lights to be turned on, a pair of "lights" arguments must be specified in the calling sequence as follows: (see the example at the end of this description):

- The argument that identifies the lowest numbered indicator light in the range must appear immediately before the argument that identifies the highest numbered indicator light in the range.

- The argument that identifies the highest numbered indicator light in the range must be a negative value and must appear in the argument list immediately after the argument that identifies the lowest numbered indicator light in the range.

EXAMPLE: The example that follows turns on indicator lights 1, 7 through 21, and 30 associated with attention level IATL1.

```
CALL  MLITS(IATL1,4,1,7,-21,30)
```

MPATL--Modify Position of an Attention Level

The MPATL subroutine is to be used by programs having more than one attention level existing at a given time (see "Using Multiple Attention Levels"). It changes the position of an attention level with respect to all other attention levels associated with the same 2250.

General Form
CALL MPATL(attnlevel,direction[,relattnlevel])

attnlevel
is an integer variable that identifies the attention level that is to have its position modified by this call. Its value must be the same as was returned as the "attnlevel" argument in the call to the CRATL subroutine for this attention level.

direction
is an integer constant or integer variable the value of which defines the direction and number of levels that the attention level specified by the "attnlevel" argument is to be moved. A value of zero causes this call to be ignored. One of the following values may be specified:

a positive value — Defines the number of levels that the specified attention level is to be moved up the attention level hierarchy relative to the attention level designated by the "relattnlevel" argument.

a negative value Defines the number of levels that the specified attention level is to be moved down the attention level hierarchy relative to the attention level designated by the "relattnlevel" argument.

relattnlevel
is an integer variable that identifies an attention level above or below which the attention level identified by the "attnlevel" argument is to be moved. The number of levels the attention level identified by the "attnlevel" argument is to be placed above or below this relative level is specified by the "direction" argument.

If the "relattnlevel" argument is omitted from this call, the attention level specified by the "attnlevel" argument is moved up or down the attention level hierarchy from its present position.

SALRM--Sound Audible Alarm

The SALRM subroutine causes the single-stroke audible alarm on the designated 2250 to be sounded. Sounding the audible alarm has no effect on 2250 performance.

General Form
CALL SALRM(devicename)

devicename
is described in "Arguments Used by Many GSP Subroutines."

LIGHT PEN SUBROUTINES

This section describes those subroutines that facilitate two-way communication between the GSP program and the 2250 operator through use of the light pen. The subroutines involve locating a position on the screen at which the light pen is pointed, and using the light pen to cause the program to move a tracking symbol from one screen location to another.

LOCATING THE LIGHT PEN

Normally, the light pen is only detectable when it is directed at an image on the screen. However, through the Locate the Light Pen (LOCPN) subroutine, GSP provides a means of locating any position on the 2250 screen at which the light pen is directed even if the light pen is not directed at an image.

LOCPN--Locate the Position of the Light Pen

The LOCPN subroutine performs a scan of the 2250 screen to locate the position of the light pen. Its use allows the program to identify the position of the light pen when the light pen is directed at a blank area of the screen. The x- and y-coordinates of the located position are returned to the calling program in terms of the programmer's coordinate system. These values are always absolute.

The scan is accomplished by displaying characters on the screen beginning at the location of the 2250 beam when the LOCPN subroutine is called, proceeding left to right across each line of the screen until the bottom right corner of the screen is reached. Once the scan has reached the lower right corner of the screen, the characters are displayed at the upper left corner of the screen and the scan continues. The scan stops when a character is detected by the light pen.

General Form
CALL LOCPN(gdsname,xpos,ypos)

gdsname
is described in "Arguments Used By Many GSP Subroutines."

xpos,ypos
are integer or real variables to which the LOCPN subroutine assigns absolute values in the programmer's coordinate system that represent the location on the screen at which the light pen was detected. These values will be of the type (integer or real) applicable for the specified graphic data set at the time the LOCPN subroutine is executed. This type was previously set by the SDATM subroutine.

PROGRAMMING CONSIDERATIONS: It may be desirable to request that the 2250 operator position the light pen before the scan begins. However, for the light pen to be located, the pen must be activated after the scan begins and must be left activated until the light pen attention occurs.

EXAMPLE: The example that follows shows the use of the LOCPN subroutine in conjunction with the Plot Points subroutine to display a point on the screen at the position at which the light pen was pointed. The example assumes that the 2250 operator has notified the program that the scan for the light pen is to be begun.

Statement 60 calls the LOCPN subroutine to scan the screen for a light pen detect and to place the x- and y-coordinates of the position where the light pen is detected into locations LX and LY respectively. Statement 70 creates an element for displaying a point on the screen at the position indicated by the coordinates returned by the call to the LOCPN subroutine in statement 60. Statement 80 executes the element generated by statement 70, thus displaying the point on the screen.

```
60  CALL  LOCPN(IGDS1,LX,LY)
70  CALL  PPNT(IGDS1,LX,LY)
80  CALL  EXEC(IGDS1)
```

TRACKING THE MOVEMENT OF THE LIGHT PEN (2250 MODEL 3 ONLY)

Light pen tracking consists of displaying a tracking symbol that can be moved by the 2250 operator by means of the light pen and of following the path that the symbol is moved. It can only be used with the 2250 Model 3.

The tracking symbol can be moved by the 2250 operator by placing the light pen on the symbol and moving the pen to a new screen position. GSP causes the tracking symbol to move in the same path as the pen.

To accomplish light pen tracking, GSP provides the following subroutines:

- Begin Light Pen Tracking (BGTRK).
- End Light Pen Tracking (ENTRK).
- Read the Current Location of the Tracking Symbol (RDTRK).

BGTRK--Begin Light Pen Tracking

The BGTRK subroutine places the tracking symbol on the 2250 screen. It is removed from the screen by a call to the ENTRK subroutine.

Only one tracking symbol can be placed on the screen at a given time. If the BGTRK subroutine is called when the tracking symbol is already being displayed, the displayed tracking symbol is repositioned at the location specified by that new call.

General Form
CALL BGTRK(gdsname,xcoor,ycoor)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

xcoor,ycoor
are constants or variables the values of which are the x- and y-coordinates of the location where the center of the tracking symbol is to be placed on the screen. These values must be of absolute form and must be given in terms of the programmer's coordinate system. They must also be of real or integer type as defined by the most recent call to the SDATM subroutine for the specified graphic data set. If these values identify a position that is outside the screen boundaries, the tracking symbol is positioned at the center of the screen and a scissoring return code is made available.

PROGRAMMING CONSIDERATIONS: The "gdsname" argument is used to identify the 2250 on which the tracking symbol is to be displayed and the type of data (real or integer) to be specified as the "xcoor" and "ycoor" arguments. The tracking symbol is not affected by any other operation performed on the graphic data set identified by the "gdsname" argument; e.g., a call to the TMGDS subroutine for the specified graphic data set will not remove the tracking symbol from the screen.

ENTRK--End Light Pen Tracking

The ENTRK subroutine removes the tracking symbol from the 2250 screen.

General Form
CALL ENTRK(gdsname)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

RDTRK--Read the Current Location of the Tracking Symbol

The RDTRK subroutine provides the x- and y-coordinates of the location of the tracking symbol at the time this subroutine is called. It may be called any time the tracking symbol is being displayed.

General Form
CALL RDTRK(gdsname,xpos,ypos)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

xpos,ypos
are variables the values of which are assigned by the RDTRK subroutine to identify the screen location of the tracking symbol at the time this call was issued. The values are returned in terms of the programmer's coordinate system and are always in absolute form. They are also of real or integer type as defined by the most recent call to the SDATM subroutine for the specified graphic data set.

EXAMPLE: Figure 12 shows the use of the RDTRK subroutine in a program that causes a line to be drawn along the path a tracking symbol is moved. The 2250 operator must depress programmed function key 6 to stop the tracking operation. Note that a combination of the STEOS, RQATN, and RESET subroutines is used to prevent interruption of the display regeneration cycle before a desired display completely appears on the screen.

```
100       CALL BGTRK (IGDS1,200.0,200.0)
           .
           .
           .
          (calls to other image generation subroutines)
           .
           .
           .
170       CALL STEOS(IGDS1,NULL,EOSKEY)
180       CALL EXEC(IGDS1)
190       CALL RQATN(IATL1,ICODE,1,NULL,6)
200       IF(ICODE) 260,210,260
210       CALL RQATN(IATL1,ICODE,2,NULL,35)
220       CALL RESET(IGDS1,NULL,EOSKEY)
230       CALL RDTRK(IGDS1,X,Y)
240       CALL PLINE(IGDS1,X,Y)
250       GO TO 170
260       CALL RESET(IGDS1,NULL,FOSKEY)
270       CALL ENTRK(IGDS1)
           .
           .
           .
```

100 Places a tracking symbol on the screen at location 200.0, 200.0.

170 Places an end-order-sequence order in IGDS1 and keys the order.

180 Causes the end-order-sequence order and all other elements in IGDS1 that were created since the last call to EXEC to be transferred to the 2250 buffer.

190 Checks if programmed function key 6 has been depressed. If it has been depressed, ICODE is set equal to six. If it has not been depressed, ICODE is set equal to zero.

200 If ICODE has a value of six (i.e., programmed function key 6 had been depressed), control is passed to statement 260 to conclude the tracking operation. Otherwise, the tracking operation continues.

Figure 12. Example of Use of Light Pen Tracking Subroutines (Part 1 of 2)

210	Stops program execution and waits for an end-order-sequence attention. This provides time for all elements in the buffer to be regenerated at least once, thus enabling the entire display to be produced before display regeneration is stopped.
220	Removes the end-order-sequence order identified by EOSKEY from the buffer.
230	Stops display regeneration, identifies the current location of the tracking symbol, places the coordinates of that location in X and Y, and restarts display regeneration.
240	Creates the element for drawing a line from the position of the 2250 beam after the last call to an image generation subroutine to the coordinates supplied by the call to the RDTRK subroutine at statement 230.
250	Branches back to statement 170 to place the STEOS order in IGDS1 immediately after the element created by the call to PLINE in statement 240 and to transfer to the buffer all elements in IGDS1 that were created since the last call to the EXEC subroutine.
260	Removes the end-order-sequence order from graphic data set IGDS1.
270	Removes the tracking symbol from the 2250 screen.

• Figure 12. Example of Use of Light Pen Tracking Subroutines (Part 2 of 2)

CHECKING PROGRAM STATUS AFTER A CALL TO A GSP SUBROUTINE

The programmer can check the status of his program after a call to a particular subroutine by requesting that certain information be made available about the execution of that subroutine. This status information may be the return codes produced by the subroutine (see "The Return Codes" below), the location of the 2250 beam, or the option definition conditions in effect (see "Option Definition Subroutines").

Four functions are available for requesting this status information. They are the Test Return Code (ITRC), the Test Integer Beam Position (ITBP), the Test Real Beam Position (RTBP), and the Test Status (ITST) functions. These functions are described in detail in the section "The Status Information Functions".

To obtain the desired information about a subroutine, one or more of these functions must be included immediately after the call to that subroutine. Each function may be used in conjunction with an IF statement or an assignment statement.

THE RETURN CODES

Return codes are integer values set during the execution of each subroutine that tell whether or not an error has been encountered during that execution, and that define the type of error (if any) that occurred. There are six possible return codes. One or several (but not necessarily all) of them may be made available by each subroutine except the INGSP and TMGSP subroutines. Except for return codes 0 and 5, additional information providing the reason for the code is also made available. All return codes and additional information about them may be obtained by means of the ITRC function. All return codes are listed in Table 6. The return codes produced by each subroutine are listed in Table 7.

THE STATUS INFORMATION FUNCTIONS

The paragraphs that follow describe the functions available for requesting status information about a subroutine.

ITRC--Test Return Code

The ITRC function assumes the value of a return code produced by the subroutine immediately preceding it. Since more than one return code can be associated with a subroutine, the programmer specifies the return code he is testing for. The ITRC function assumes either the value of the highest code produced that is greater than or equal to the code being tested, or a value of zero if no code greater than or equal to the code being tested exists.

If a value other than zero or five is assumed, additional information concerning the reason for the return code is made available. This additional information can be obtained by again executing the ITRC function, this time specifying a value of nine as the "code" argument. In this case, the ITRC function will assume the value of the additional information as listed in Table 6.

Table 6. Description of All GSP Return Codes

Return Code	Meaning	Additional Information
0	The subroutine completed its execution successfully.	None
1	A scissoring option has been invoked.	• The value assigned as the "count" argument in the call to the image generation subroutine that created the data that was last scissored. When the scissored data is associated with a call to the STPOS or MVPOS subroutine (the calling sequences of which do not contain the "count" argument), a value of of one is returned. • Zero, if no data was produced by the call; i.e., the call was ignored.
2	An error occurred during the scaling of input data.	Same as for return code 1, except that it applies to scaled data rather than to scissored data.
3	An attempt has been made to exceed a program-defined or system-defined storage area during the subroutine execution. The call essentially has been ignored.	1 The update function cannot be performed because the update element is larger than the originally created element it is replacing (see "The Update Facility"). 2 A 128-byte graphic data set has been exceeded. 3 Insufficient buffer storage is available for the data. 4 Insufficient main storage is available for the data.
4	The arguments in the call to this subroutine have been specified incorrectly, or there is a logical error in subroutine usage (i.e., an attempt was made to end a sequence not yet begun). The call essentially has been ignored.	0 The error cannot be attributed to a single argument. 1-n The integer number of the first argument in the calling sequence that caused the error (from left to right beginning at one).
5	An input/output error was encountered during execution of the subroutine. The call essentially has been ignored.	None

Table 7. Return Codes for Each GSP Subroutine

Subroutine Group	Subroutine	Return Codes			
		1	2	3	5
Initiation	INGSP				
	INDEV			X	X
	INGDS			X	
	SPEC				
Termination	TMGSP				
	TMDEV				X
	TMGDS				X
Option Definition	SDATM				
	SGRAM				
	SCHAM				
	SGDSL				
	SDATL				
	SSCIS				
Image Generation	MVPOS	X	X	X	X
	STPOS	X	X	X	X
	PLINE	X	X	X	X
	PPNT	X	X	X	X
	PSGMT	X	X	X	X
	PTEXT	X	X	X	X
	STEOS			X	X
Identification	BGSEQ			X	X
	ENSEQ			X	X
	BGSUB			X	X
	ENSUB			X	X
	LKSUB			X	X
Image Control	EXEC				X
	INCL				X
	OMIT				X
	ORGDS				X
	RESET				X
	IDPOS		X		
Keyboard Input & Buffer Data Analysis	ICURS				X
	RCURS				X
	GSPRD				X
Attention Related	CRATL			X	X
	ENATL				X
	ENATN				X
	DSATN				X
	SLPAT				X
	RQATN				
	MLITS				X
	MLPEO				
	MPATL				X
	SALRM				X
Light Pen	LOCPN		X		X
	BGTRK	X	X		X
	ENTRK				X
	RDTRK		X		X
Stroke Generator	DFSTR				X
	PLSTR	X	X	X	X
Miscellaneous	CNVRT		X		
	ORGEN			X	X
	FSMOD				

NOTE: A code of zero or four may be returned by any of the subroutines except INGSP and TMGSP, which do not make return codes available.

General Form
ITRC(gspname,code)

gspname
is an integer variable the value of which must be the same as was returned as the "gspname" argument in the call to the INGSP subroutine.

code
is an integer constant or integer variable the value of which either designates the return code being tested, or denotes a request for additional information about the last return code assumed by the ITRC function. This argument may have one of the following values:

1 For return code 1
2 For return code 2
3 For return code 3
4 For return code 4
5 For return code 5
9 For additional information, if a code other than zero or five was last returned.

If a value other than 1, 2, 3, 4, 5, or 9 is specified as the "code" argument, the ITRC function assumes a value of -1.

CAUTION: Once a return code is tested for, the record of its existence is destroyed. That is, the ITRC function will not return a specific return code value about a particular call more than once.

ITBP--Test Integer Beam Position

The ITBP function returns coordinate information about the location of the 2250 beam if integer output is produced by the image generation subroutine being tested. The information returned may be either (1) the x- or y-coordinate where the 2250 beam was positioned after execution of the most recent image generation subroutine associated with the specified graphic data set, or (2) the x- or y-coordinate where the programmer intended the beam to be at the time this function is executed. The actual beam position and the intended beam position are the same except when scissoring has occurred. The coordinates returned are in terms of the programmer's coordinate system.

If the coordinate to be returned is of real type, the maximum negative value is returned. In this case, the correct coordinate can be obtained by using the RTBP function.

General Form
ITBP(gdsname,info)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

info
is an integer constant the value of which defines the coordinate to be returned as follows:

1 The x-coordinate where the beam is supposed to be at this time.

2 The x-coordinate where the beam actually is at this time.
3 The y-coordinate where the beam is supposed to be at this time.
4 The y-coordinate where the beam actually is at this time.

RTBP--Test Real Beam Position

The RTBP function is the same as the ITBP function except that it only returns coordinates if real output data is produced by the image generation subroutine being tested. If the coordinate to be returned is of integer type, the maximum negative value is returned. In this case, the correct coordinate can be obtained by using the ITBP function.

General Form
RTBP(gdsname,info)

gdsname,info
are described in the detailed description of the ITBP function.

ITST--Test Status

The ITST function assumes a value corresponding to the type of data currently in effect for a particular option definition subroutine for the associated graphic data set. The value assumed depends on the type of data defined by the most recent execution of the specified option definition subroutine, as follows:

Subroutine	Assumed Value	Meaning
SDATM	1	Input data is to be real, absolute.
	2	Input data is to be real, incremental.
	3	Input data is to be integer, absolute.
	4	Input data is to be integer, incremental.
SGRAM	1	Output data is in optimized form.
	2	Output data is in absolute form.
	3	Output data is in incremental form.
SCHAM	1	Basic-size, protected characters were requested.
	2	Large-size, protected characters were requested.
	3	Basic-size, unprotected characters were requested.
	4	Large-size, unprotected characters were requested.

General Form
ITST(gdsname,optionsub)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

optionsub
is an integer constant that designates the option definition subroutine being tested, as follows:

1 SDATM subroutine (x-coordinate)
2 SDATM subroutine (y-coordinate)
3 SGRAM subroutine
4 SCHAM subroutine

This appendix describes sample coding techniques using GSP subroutines and functions in conjunction with the FORTRAN IV language. The program illustrates coding to generate displays on the IBM 2250 Display Unit, Models 1 and 3.

The coding is contained in Figure 13. Displays produced by it are shown in Figure 14. Comments in the coding explain the functions of various portions of the program.

The program is intended for use with a 2250 Model 1 equipped with the buffer, light pen, programmed function keyboard, and character generator features; or with a 2250 Model 3 equipped with the programmed function keyboard feature.

The GSPSAMP card deck can be punched from SYS1.SAMPLIB. It consists of:

1. The following job control language statements that will call a cataloged procedure to compile, linkage edit, and execute the program when the FORTRAN (G) compiler is used in the operating system:

```
//GSPSAMY       JOB     1234567,GSPSAMP,MSGLEVEL=1
//SAMPL         EXEC    FORTGCLG,PARM.FORT=(NODECK,MAP,LOAD)
//FORT.SYSIN    DD      *
```

   If the FORTRAN (E) or (H) compiler is to be used, the EXEC statement listed above must be changed accordingly.

2. Sample program input symbolic deck.

3. The following cards after the FORTRAN Language END card:

```
/*
//LKED.SYSIN    DD
  INCLUDE       SYSLIB(IHCGSP03)
/*
//GO.SYSABEND   DD      SYSOUT=A
//GO.FT10F001   DD      UNIT=(2250-1)
//GO.SYSIN      DD      *
X
NUM1NUM2NUM3NUM4NUM5NUM6NUM7NUM8
/*
```

   If the FORTRAN (E) compiler is to be used, the following card must be added after the GO.SYSABEND DD statement listed above:

```
//GO.FT05F001   DD      UNIT=AFF=FT01F001
```

Once the GSPSAMP card deck has been punched from SYS1.SAMPLIB, the following steps are necessary to process the program:

1. Insure that the graphic device is on.

2. If the FORTRAN (E) or (H) compiler is being used, modify the card deck accordingly as described in the preceding paragraph.

3. Place the sample program deck in the card reader.

4. Mount the operating system.

5. Set the load address switches and press the Initial Program Load key to load the system.

6. Perform compile-link edit-go procedure.

7. Follow instructions which appear with sample program in Figure 13.

IBM FORTRAN Coding Form X28-7327-5 Printed in U.S.A.

PROGRAM FORTRAN GSP SAMPLE PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 1 OF 7

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER*

STATEMENT NUMBER	CONT.	FORTRAN STATEMENT	IDENTIFICATION SEQUENCE
C		THIS SAMPLE PROGRAM DRAWS A CIRCLE OF 8 BASIC SIZE X'S WITH A	
C		LARGE SIZE X AS ITS CENTER. DEPENDING ON THE ATTENTION CREATED,	
C		THE PROGRAM WILL DO THE FOLLOWING:	
C			
C			
C		1. LIGHT PEN ATTENTION ON CENTER X LABELS OUTER X'S FROM	
C		NUM1 THROUGH NUM8. (STATEMENT 221).	
C			
C		2. LIGHT PEN ATTENTION ON OUTER X GENERATES A CIRCLE WHOSE	
C		CENTER IS THE DETECTED X. (STATEMENT 222).	
C			
C		3. LIGHT PEN ATTENTION ON CIRCLE AROUND OUTER X MAKES THAT	
C		CIRCLE INVISIBLE. (STATEMENT 230).	
C			
C		4. LIGHT PEN ATTENTION ON ANY LABEL MAKES ALL LABELS	
C		INVISIBLE. (STATEMENT 225).	
C			
C		5. PROGRAMMED FUNCTION KEY 1 ENDS RUN. (STATEMENT 240).	
C			
C		6. PROGRAMMED FUNCTION KEY 2 MAKES ALL CIRCLES INVISIBLE.	
C		(STATEMENT 250).	
C			
C		7. PROGRAMMED FUNCTION KEY 3 MAKES ALL CIRCLES VISIBLE.	
C		(STATEMENT 260).	

*A standard card form, IBM electro 888157, is available for punching statements from this form

Figure 13. GSP Sample Program (Sheet 1 of 7)

IBM

FORTRAN Coding Form

X28-7327-5
Printed in U.S.A.

PROGRAM FORTRAN GSP SAMPLE PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 2 OF 7

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER*

STATEMENT NUMBER	CONT.	FORTRAN STATEMENT	IDENTIFICATION SEQUENCE
		C BEGIN PROGRAM	
		C SET UP DIMENSION STATEMENT FOR VARIOUS ARRAYS USED.	
		DIMENSION INTARY(10),CIRX(8),CIRY(8),SMALX(8,24),SMALY(8,24),	
		X LABEL(32),NULL(1)	
		C INITIALIZE THE GRAPHIC SUBROUTINE PACKAGE. DEFINE	
		C THE NULL VARIABLE.	
		NULL(1)=-5	
		CALL INGSP(IGSP,NULL)	
		C INITIALIZE THE 2250. ASSIGN IT UNIT NUMBER 10.	
		CALL INDEV(IGSP,10,I2250)	
		C INITIALIZE A GRAPHIC DATA SET THAT WILL BE USED TO	
		C DISPLAY THE CIRCLE OF X'S AND ITS CENTER X.	
		CALL INGDS(I2250,IGDS1)	
		C INITIALIZE A GRAPHIC DATA SET THAT WILL BE USED TO	
		C LABEL THE OUTER X'S OF THE CIRCLE.	
		CALL INGDS(I2250,IGDS2)	
		C INITIALIZE A GRAPHIC DATA SET THAT WILL BE USED TO	
		C DISPLAY A CIRCLE AROUND ANY OF THE OUTER X'S.	
		CALL INGDS(I2250,IGDS3)	
		C CREATE AN ATTENTION LEVEL FOR THE 2250.	
		CALL CRATL(I2250,IATL)	
		C ENABLE PROGRAMMED FUNCTION KEYS 1,2, + 3 + LIGHT PEN.	
		CALL ENATN(IATL,1,-3,34)	
		CALL MLITS(I2250,3)	

*A standard card form, IBM electro 888157, is available for punching statements from this form

• Figure 13. GSP Sample Program (Sheet 2 of 7)

IBM FORTRAN Coding Form X28-7327-5 Printed in U.S.A.

PROGRAM FORTRAN GSP SAMPLE PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 3 OF 7

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER*

STATEMENT NUMBER	CONT.	FORTRAN STATEMENT	IDENTIFICATION SEQUENCE
C		ALLOW LIGHT PEN ATTENTIONS FOR EACH GRAPHIC DATA SET.	
		CALL SLPAT(IGDS1,1)	
		CALL SLPAT(IGDS2,1)	
		CALL SLPAT(IGDS3,1)	
C		THE FOLLOWING INFORMATION IS ON CARD RECORDS. DEVICE	
C		NUMBER IS 5. CHARACTER X IS LEFT ALIGNED AND APPEARS	
C		ON THE CARD RECORD AS "Xbbb". EIGHT 4-DIGIT LABELS	
C		(NUM1,NUM2,ETC.) FOLLOW THE X. STORAGE WAS ALLOCATED	
C		FOR THE ARRAY "LABEL" BY THE PREVIOUS DIMENSION STATE	
C		MENT.	
C			
C		PLACE CHARACTER X IN MAIN STORAGE.	
20		FORMAT(A4)	
		READ(5,20)X	
C		PLACE LABELS IN MAIN STORAGE.	
30		FORMAT(8A4)	
		READ(5,30)(LABEL(INCR),INCR=1,8)	
C		SET BEAM POSITION TO CENTER OF CIRCLE.	
		CX=2047.0	
		CY=2047.0	
C		SET CHARACTER MODE FOR LARGE SIZE, PROTECTED.	
		CALL SCHAM(IGDS1,2)	
C		DISPLAY CENTER X.	
		NUM=-1	

*A standard card form, IBM electro 888157, is available for punching statements from this form

Figure 13. GSP Sample Program (Sheet 3 of 7)

IBM

FORTRAN Coding Form

X28-7327-5
Printed in U.S.A.

PROGRAM FORTRAN GSP SAMPLE PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 4 OF 7

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER*

STATEMENT NUMBER	CONT.	FORTRAN STATEMENT	IDENTIFICATION SEQUENCE
		CALL PTEXT(IGDS1,X,1,NUM,NULL,1,CX,CY)	
C		COMPUTE X AND Y COORDINATES FOR X'S OF OUTER CIRCLE.	
C		PLACE THEM IN THE CIRX AND CIRY ARRAYS FOR WHICH MAIN	
C		STORAGE HAS BEEN ALLOCATED BY DIMENSION STATEMENT.	
		R=1200.0	
		C=3.141593/180	
		THETA=45.0	
		DO 10 I=1,8	
		RADIAN=THETA*FLOAT(I)*C	
		CIRX(I)=CX+R*COS(RADIAN)	
10		CIRY(I)=CY+R*SIN(RADIAN)	
C		SET CHARACTER MODE FOR BASIC SIZE, PROTECTED X.	
		CALL SCHAM(IGDS1,1)	
C		GENERATE AN X IN INCLUDE STATUS. CALCULATE	
C		X-COORDINATE OF LABEL ASSOCIATED WITH THAT X.	
C		GENERATE LABEL IN INCLUDE STATUS. SET INCR TO NEXT	
C		LABEL IN ARRAY. DO THIS 8 TIMES TO COMPLETE CIRCLE.	
		DO 50 NUM=1,8	
		CALL PTEXT(IGDS1,X,1,NUM,NULL,1,CIRX(NUM),CIRY(NUM))	
		CXNUM=CIRX(NUM)+300.0	
50		CALL PTEXT(IGDS2,LABEL(NUM),4,NUM,NULL,1,CXNUM,CIRY(NUM))	
C		COMPUTE COORDINATES FOR CIRCLES AROUND OUTER X'S.	
C		PLACE THEM IN SMALX AND SMALY ARRAYS FOR WHICH MAIN	
C		STORAGE HAS BEEN ALLOCATED BY DIMENSION STATEMENT.	

*A standard card form, IBM electro 888157, is available for punching statements from this form

• Figure 13. GSP Sample Program (Sheet 4 of 7)

IBM

FORTRAN Coding Form

X28-7327-5
Printed in U.S.A.

PROGRAM FORTRAN GSP SAMPLE PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 5 OF 7

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER*

STATEMENT NUMBER	CONT.	FORTRAN STATEMENT	IDENTIFICATION SEQUENCE
		R=200.0	
		THETA=360.0/24.0	
		DO 105 N=1,24	
		RADIAN=THETA*FLOAT(N)*C	
		DO 105 I=1,8	
		SMALX(I,N)=CIRX(I)+R*COS(RADIAN)	
105		SMALY(I,N)=CIRY(I)+R*SIN(RADIAN)	
C		GENERATE CIRCLES AROUND OUTER X'S IN OMIT STATUS.	
		DO 110 NUM=1,8	
		CALL STPOS(IGDS3,CIRX(NUM)+200.0,CIRY(NUM))	
110		CALL PLINE(IGDS3,SMALX(NUM,1),SMALY(NUM,1),NUM,NULL,2,24,8,8)	
C		OMIT LABELS FROM THE DISPLAY.	
		CALL OMIT(IGDS2)	
C		EXECUTE ALL GRAPHIC DATA PRODUCED.	
		CALL EXEC(IGDS1)	
		CALL EXEC(IGDS2)	
		CALL EXEC(IGDS3)	
C		REQUEST ATTENTION INFORMATION TO DETERMINE WHETHER	
C		2250 OPERATOR WANTS TO CHANGE OR END THE DISPLAY.	
C		WAIT FOR AN ATTENTION TO OCCUR.	
200		CALL RQATN(IATL,INTCD,2,INTARY,1,2,3,34)	
C		IF LIGHT PEN ATTENTION, GO TO 210.	
C		IF PROGRAMMED FUNCTION KEY ATTENTION, GO TO 235.	
		IF (INTCD-34) 235,210,210	

*A standard card form, IBM electro 888157, is available for punching statements from this form

Figure 13. GSP Sample Program (Sheet 5 of 7)

IBM

FORTRAN Coding Form

X28-7327-5
Printed in U.S.A.

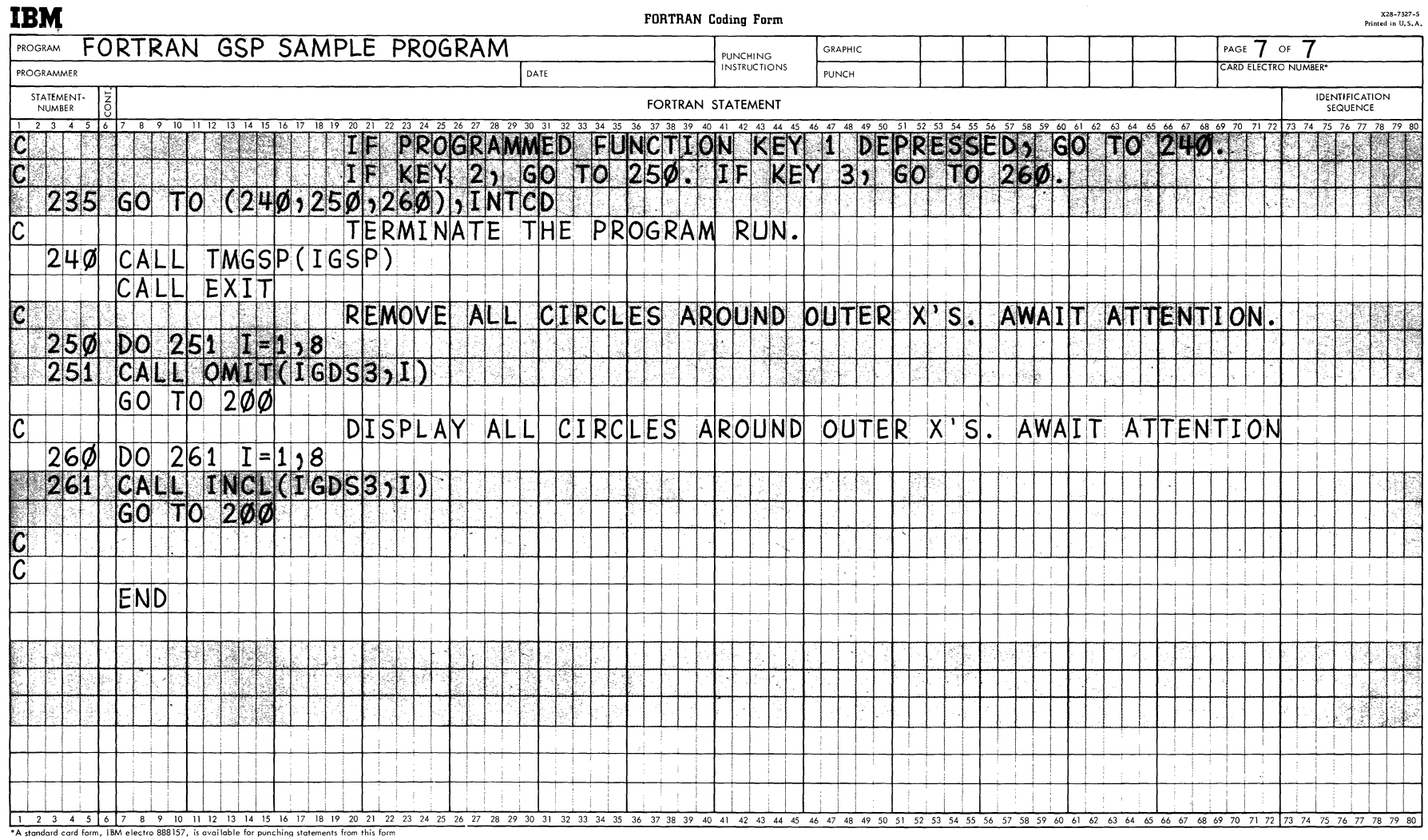
PROGRAM FORTRAN GSP SAMPLE PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 6 OF 7

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER*

STATEMENT NUMBER	CONT.	FORTRAN STATEMENT	IDENTIFICATION SEQUENCE
C		IF LIGHT PEN ATTENTION WAS ASSOCIATED WITH IGDS1, GO	
C		TO 220. IF IGDS2, GO TO 225. IF IGDS3, GO TO 230.	
210		IF(INTARY(1)-IGDS1) 201,220,201	
201		IF(INTARY(1)-IGDS2) 202,225,201	
202		IF(INTARY(1)-IGDS3) 200,230,200	
C		IF LIGHT PEN ATTENTION WAS ON CENTER OF LARGE CIRCLE,	
C		GO TO 221. IF ON OUTER CIRCLE, GO TO 222.	
220		IF(INTARY(4)) 221,222,222	
C		DISPLAY LABELS AND WAIT FOR ANOTHER ATTENTION.	
221		CALL INCL(IGDS2)	
		GO TO 200	
C		DISPLAY A CIRCLE AROUND THE OUTER X DETECTED AND WAIT	
C		FOR ANOTHER ATTENTION.	
222		NUM=INTARY(4)	
		CALL INCL(IGDS3,NUM)	
		GO TO 200	
C		REMOVE LABELS FROM DISPLAY. AWAIT NEW ATTENTION.	
225		CALL OMIT(IGDS2)	
		GO TO 200	
C		REMOVE DETECTED CIRCLE FROM DISPLAY. AWAIT ATTENTION.	
230		NUM=INTARY(4)	
		CALL OMIT(IGDS3,NUM)	
		GO TO 200	
C			

*A standard card form, IBM electro 888157, is available for punching statements from this form

Figure 13. GSP Sample Program (Sheet 6 of 7)



IBM

FORTRAN Coding Form

X28-7327-5
Printed in U.S.A.

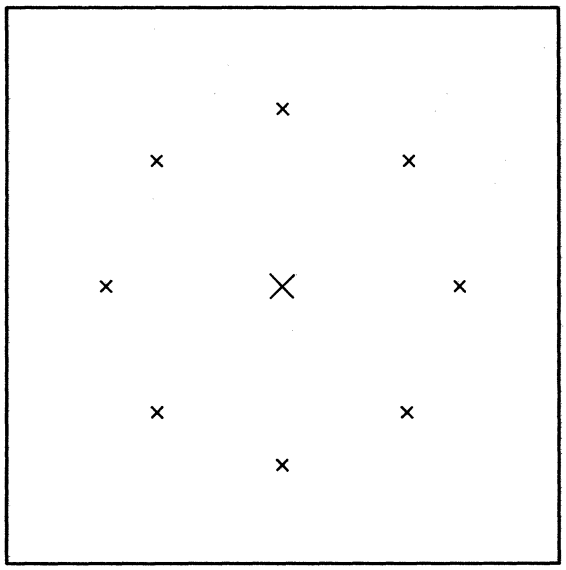
PROGRAM FORTRAN GSP SAMPLE PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE 7 OF 7

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER*

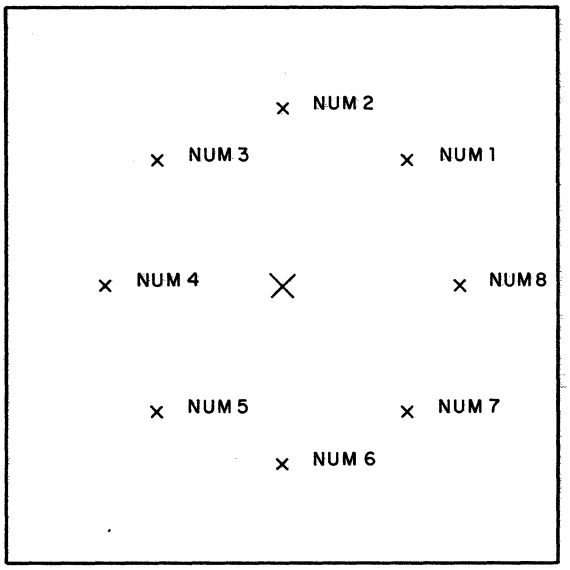
STATEMENT NUMBER	CONT	FORTRAN STATEMENT	IDENTIFICATION SEQUENCE
C		IF PROGRAMMED FUNCTION KEY 1 DEPRESSED, GO TO 240.	
C		IF KEY 2, GO TO 250. IF KEY 3, GO TO 260.	
235		GO TO (240,250,260),INTCD	
C		TERMINATE THE PROGRAM RUN.	
240		CALL TMGSP(IGSP)	
		CALL EXIT	
C		REMOVE ALL CIRCLES AROUND OUTER X'S. AWAIT ATTENTION.	
250		DO 251 I=1,8	
251		CALL OMIT(IGDS3,I)	
		GO TO 200	
C		DISPLAY ALL CIRCLES AROUND OUTER X'S. AWAIT ATTENTION	
260		DO 261 I=1,8	
261		CALL INCL(IGDS3,I)	
		GO TO 200	
C			
C			
		END	

*A standard card form, IBM electro 888157, is available for punching statements from this form

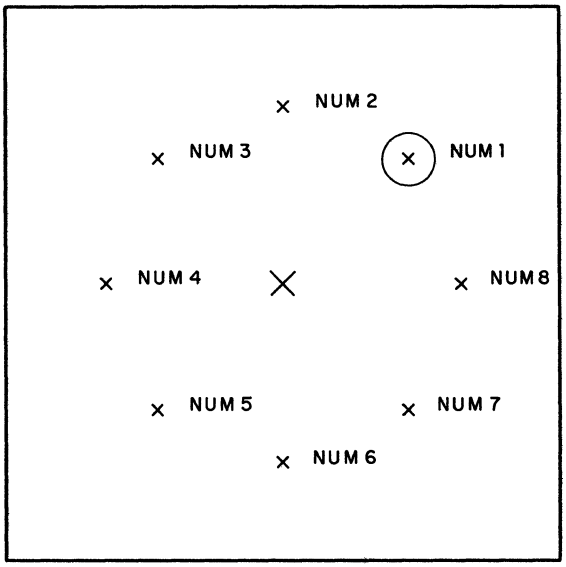
Figure 13. GSP Sample Program (Sheet 7 of 7)



A. Display Produced After Call to EXEC for IGDS1.



B. Display Produced After Light Pen Attention on Center X.



C. Display Produced After Light Pen Attention on X Labeled NUM1.

Figure 14. Displays Produced by GSP Sample Program

All GSP capabilities available to the FORTRAN programmer are also available to the Assembler Language programmer.

GSP does not set or alter the interruption mask established by the Specify Program Interruption Exit (SPIE) macro instruction. Setting this mask is the responsibility of the Assembler Language programmer.

LINKAGES

All linkages in GSP are as described for a simple program structure in the publication IBM System/360 Operating System: Supervisor and Data Management Services, Form C28-6646.

CALLING A GSP SUBROUTINE

A GSP subroutine is called from an Assembler Language program by means of the CALL macro instruction formatted as described in the publication IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, Form C28-6647. The parameter list for each CALL macro instruction must contain the addresses of the constants or variables being passed to the called subroutine; if the values of constants or variables are placed in a parameter list, those values are considered to be addresses.

FUNCTIONS

The value returned by the functions ITRC, ITBP, RTBP, and ITST will be in general register 0 for fixed-point values, or in floating-point register 0 for floating-point values, rather than in a parameter list. For a complete discussion of calling FORTRAN functions, see the Programmer's Guide for the FORTRAN compiler being used.

LINKAGE EDITOR REQUIREMENTS

For calls to GSP subroutines to be resolved, the following must be specified as input to the linkage editor:

- Automatic library call mechanism must be used and must refer (via the SYSLIB DD statement) to the FORTRAN library (SYS1.FORTLIB).
- INCLUDE SYSLIB(IHCGSP03) statement must be specified.

Additional job control requirements for processing an Assembler Language program are contained in the programmer's guide for the assembler being used.

APPENDIX C: PRODUCING CHARACTERS WITHOUT A CHARACTER GENERATOR

The character generator of the 2250 produces a standard set of characters of two sizes (basic and large) and of one orientation (vertical). If the programmer wishes to display special characters, or characters of a different size and orientation, he must plot these characters by means of connecting lines, or strokes. GSP provides for defining as a special character any design that can be constructed as strokes joining points on an 128-by-128 matrix.

The strokes that define characters (hereafter referred to as graphic symbols) in this manner are placed in a stroke table. Once placed in a stroke table, a graphic symbol may be plotted in any position, size, and orientation by calling the Plot Strokes (PLSTR) subroutine. Once created, stroke tables can be modified by the programmer by means of the Define Strokes (DFSTR) subroutine.

The System Stroke Table

GSP provides a stroke table, called the system stroke table, which may be used by the programmer. The system stroke table (named GSP01) contains the strokes which produce the standard set of extended binary-coded-decimal interchange code (EBCDIC) graphic symbols and the address of each set of strokes in the table.

The location of the address within the table of any specific set of strokes is determined by the addition of the decimal integer value of the EBCDIC representation of the graphic symbol to the address of the table. This function is performed by GSP.

The programmer must use the integer value of the EBCDIC representation of the graphic symbol in calls to the PLSTR and DFSTR subroutines to identify the location of the symbols to be affected by the call. For example, the integer value of A is 193; therefore, to locate the address of the strokes to generate A, 193 would be used in the call.

The section of the system stroke table containing the addresses of the strokes consists of 256 locations, one for each possible EBCDIC representation. A value of zero is assigned to those locations that do not correspond to the integer value of a valid EBCDIC graphic symbol. Therefore, space for strokes is only allocated for those patterns which represent graphic symbols. Subsequently, if special graphic symbols are desired, the programmer should either (1) modify the system stroke table by associating the special symbol with EBCDIC representations that would normally produce another type of EBCDIC graphic symbol, or (2) create a new stroke table.

The system stroke table allocates space for sixteen strokes (including the positioning stroke) for each graphic symbol. EBCDIC representations and their associated graphic symbols are described in the publication IBM System/360 Principles of Operation, Form A22-6821.

Creating a stroke table, and the DFSTR and PLSTR subroutines are described in detail in the paragraphs that follow.

Creating a Stroke Table

Whenever the system stroke table does not contain symbols necessary for a particular application, the GSP programmer can create his own stroke

table. This stroke table must be placed in the link library or the job library. It must not be reusable or reenterable.

The stroke table may contain addresses and associated constants for defining up to 256 symbols. Each address is the address of the constants that define the strokes for a particular symbol. The location of an address in the table of addresses must be used in calls to the PLSTR and DFSTR subroutines to identify the location of the symbols to be affected by the call.

The stroke table must be named GSPnn, where "nn" is a number from 02 to 99. Words 1 and 2 of the stroke table must be reserved; the table of addresses and constants must begin in word 3 (see Figure 15).

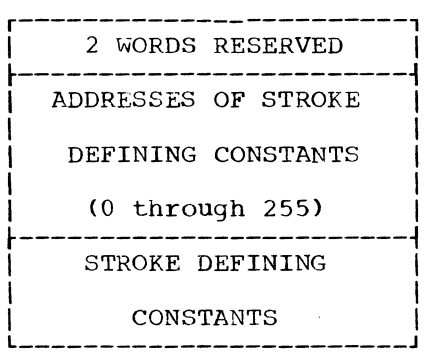


Figure 15. Composition of Stroke Table

The starting position for a symbol and the strokes necessary to produce that symbol are defined by a series of halfwords formed by Define Constant (DC) instructions. The first halfword defined for each symbol must position the 2250 beam at the starting location within the 128-by-128 matrix of the first stroke for the symbol. Each subsequent halfword defines one of the strokes that make up the symbol. Each halfword must contain the following (also see Figure 16):

Bits	Contents
0	A zero to designate that the 2250 beam is to be in the unblanked mode. A one to designate that the 2250 beam is to be in the blanked mode. A one must be specified in the halfword that defines the starting position of the strokes.
1-7	An integer value from 0 through 127 that indicates the x-coordinate in an 128-by-128 matrix to which the beam is to be moved.
8	A zero to indicate that additional strokes are necessary to complete this symbol. A one to indicate that this is the last stroke necessary to complete this symbol.
9-15	An integer value from 0 through 127 that indicates the y-coordinate in a 128-by-128 matrix to which the beam is to be moved.

CAUTION: When creating a stroke table, the programmer should allocate adequate space for future modification of the table. Also, any increment in the matrix for defining the graphic symbol that is less than one raster unit (0.0117 inch) from the preceding increment is ignored.

EXAMPLE: A series of halfwords in a stroke table for producing the letter "V" may be specified as follows:

```
CODEV  DC  X'9060300050E0'
```

Where:

9060 positions the 2250 beam (in blanked mode) to the starting location within the matrix of the first stroke making up the symbol.

3000 and 50E0 define the two strokes making up the V.

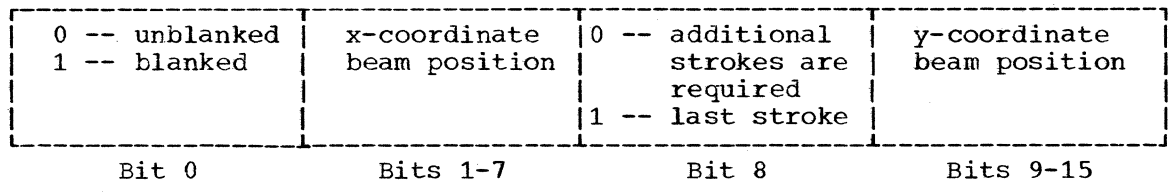


Figure 16. Contents of Stroke Defining Halfwords

DFSTR--Define Strokes

The DFSTR subroutine allows the programmer to modify the symbols previously defined by a stroke table. The stroke table modified by this subroutine may be the system stroke table (identified as number 01) or any other stroke table. The control information about the strokes making up the new symbols to be added by this subroutine must have been previously placed in a main storage array.

When this subroutine is called, a copy of the stroke table previously placed in a system library is brought into main storage. All modifications and additions are made to that main storage copy. The copy in the system library remains unchanged.

General Form
CALL DFSTR(gdsname,table,strokes,symbol,strokecount)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

table
is an integer constant or integer variable that defines the stroke table to be modified. A value of one designates the system stroke table.

strokes
is a variable or array name that identifies the entry in a main storage array that defines the strokes necessary to generate the desired symbol. A pair of words must be included in the array for each stroke. The contents of each word is as follows:

Word 1

Is a positive or negative integer value from 1 through 128 that designates the x-coordinate in a 128-by-128 matrix to which the beam is to be moved. If this value is positive, the beam is moved in the unblanked mode. If this value is negative, the beam is moved in the blanked mode.

Word 2

Is a positive integer value from 1 through 128 that designates the y-coordinate in a 128-by-128 matrix to which the beam is to be moved. The beam is moved in the blanked or unblanked mode as designated by the sign (positive or negative) of the value in word 1.

symbol
is an integer constant or integer variable from 0 through 255 that defines the EBCDIC representation of the graphic symbol to be modified in the stroke table. Integer equivalents of EBCDIC

graphic symbols are described in the publication IBM System/360 Principles of Operation, Form A22-6821.

strokecount
is an integer constant or integer variable the value of which specifies the number of strokes that are used to make up the symbol added to the stroke table by this call.

CAUTION: GSP assumes that there is adequate space in the stroke table to make the requested modifications. If there is not enough space in the stroke table for a specific modification, the symbol immediately following the modified symbol may be altered or overlaid. The system stroke table allocates space for sixteen strokes (including the positioning stroke) for each EBCDIC graphic symbol.

Any increment in the matrix for defining the graphic symbol that is less than one raster unit (0.0117 inch) from the preceding increment is ignored.

PROGRAMMING CONSIDERATIONS: If the configuration of any symbols in a stroke table is modified by this subroutine, the next call to the PLSTR subroutine for that EBCDIC representation produces the element necessary for displaying the new symbol. The previous symbol is lost until it is redefined by another call to the DFSTR subroutine or until a new copy of the stroke table is brought into main storage. Once brought into main storage, a copy of a stroke table remains there until the TMGSP subroutine is called.

PLSTR--Plot Strokes

The PLSTR subroutine creates the element necessary for displaying graphic symbols in any position, size, and orientation desired by the programmer. The strokes necessary to produce these symbols must have been previously placed in a stroke table.

The PLSTR subroutine may also be used to update an element produced by a previous call to the PLSTR subroutine. This facility is described in detail in a previous section entitled, "The Update Facility."

General Form
CALL PLSTR(gdsname,table,text,count,height[,width][,spacing] [,orientation][,corrval][,key][,gencode] [,xcoor,ycoor])

gdsname,corrval,key,gencode
are described in "Arguments Used by Many GSP Subroutines."

table
is an integer constant or integer variable that defines the stroke table containing the symbols to be displayed. A value of one designates the system stroke table.

text
is either (1) an integer constant or integer variable with a value from 0 through 255 that identifies the location of an address of a word containing the symbol(s) to be plotted by this call or (2) an array name that designates an array containing integer values from 0 through 255 that identify the location of the addresses of several words containing symbols to be plotted by this call. In either case, there may be one symbol (right justified) or four symbols in each word.

count
is an integer constant or integer variable that specifies the number of symbols in the text string to be plotted. A positive sign indicates there are four EBCDIC symbols per word; a negative sign indicates there is one EBCDIC symbol (right justified) per word.

height
is a real constant or real variable that specifies the height (in inches) of the matrix in which the symbol is to be plotted. The minimum value that can be specified is 0.125 inches. No more than three digits may appear to the right of the decimal.

width
is a real constant or real variable that specifies the width (in inches) of the matrix in which the symbol is to be plotted. The minimum value that can be specified is 0.125 inches. No more than three digits may appear to the right of the decimal. If this argument is not specified, the width is assumed to be two-thirds of the value specified in the "height" argument.

spacing
is a real constant or real variable that specifies the center-to-center spacing between the symbols plotted. No more than three digits may appear to the right of the decimal. Spacing may be greater or less than the value specified as the "width" argument. If this argument is not specified, spacing is assumed to be 1 1/3 times the value specified in the "height" argument.

orientation
is the name of a real, single-precision array consisting of four elements that specify the orientation of the symbols to be displayed. Elements one and two contain the values of the sine and cosine respectively of the angle of rotation of the symbol from the positive x-axis. Elements three and four contain the values of the sine and cosine respectively of the angle of rotation of the entire text string from the positive x-axis. If this argument is not specified, the symbol and text string will be plotted with the angle of rotation of both of them at zero degrees.

xcoor,ycoor
are constants or variables representing the programmer's x- and y-coordinates on the screen where the center of the first symbol plotted by this call is to be displayed when the element produced by this call is executed. Both arguments must be specified as a pair. If they are not specified, the first symbol is displayed wherever the 2250 beam is positioned at the time this subroutine is called. If these arguments are specified, the coordinates must be of the type and form defined by the most recent call to the SDATM subroutine for the specified graphic data set, or by default. The coordinates are appropriately scaled as defined by the most recent call to the SDATL subroutine, or by default.

CAUTION: Unlike text elements, the 2250 beam is moved when a PLSTR element in omit status is executed. As usual, the image associated with the element is not displayed as long as the element is in omit status.

PROGRAMMING CONSIDERATIONS: Scissoring options set by the SSCIS subroutine apply to strokes in the same manner as they apply to images resulting from the execution of the image generation subroutines.

Coordinates may be converted from the values specified by the programmer to their corresponding values used by the 2250 to position the 2250 beam, and vice versa. This conversion may be accomplished by means of the Convert Coordinates (CNVRT) subroutine described in the paragraphs that follow.

CNVRT--Convert Coordinates

The CNVRT subroutine converts a specified coordinate value in the programmer's coordinate system to a coordinate value used by the 2250, and vice versa. The device coordinates recognized by the 2250 are called raster units.

General Form
CALL CNVRT(gdsname,convert[,xinput][,yinput][,xoutput][,youtput])

gdsname
is described in "Arguments Used by Many GSP Subroutines."

convert
is an integer constant or integer variable the value of which defines the type of conversion desired, as follows:

1 Convert integer coordinates supplied in raster units to their equivalent scaled programmer coordinates of the type (integer or real) currently applicable for the specified graphic data set as defined by the SDATM subroutine.

2 Convert coordinates specified in terms of the programmer's coordinate system into their equivalent integer raster unit values.

xinput,yinput
are constants or variables the values of which are the coordinates to be converted by this call.

xoutput,youtput
are variables to which the CNVRT subroutine will return the converted values.

APPENDIX E: DIRECT ORDER GENERATION

By using the Generate Graphic Orders (ORGEN) subroutine, graphic orders valid for the IBM 2250 and their related data can be moved into a graphic data set from a main storage array without calling the image generation subroutines. The logic of the moved orders is not checked.

The graphic orders, their functions, and their hexadecimal equivalents are described in the publication IBM System/360 Operating System: Graphic Programming Services for IBM 2250 Display Unit, Form C27-6909.

ORGEN--Generate Graphic Orders

The ORGEN subroutine moves graphic orders and related data into a graphic data set from a main storage array. The rules that apply to the use of image generation subroutines also apply to the use of this subroutine.

General Form
CALL ORGEN(gdsname,arrayname,count[,keylist][,corrval][,key] [,gencode])

gdsname,corrval,key,gencode
: are described in "Arguments Used by Many GSP Subroutines."

arrayname
: is a variable or an array name that identifies an array containing the graphic orders and data to be added to the designated graphic data set. These orders must be in the format defined for the 2250, and must be as if they were written at buffer address 0. (These orders will be relocated as necessary by the ORGEN subroutine.) Any addresses which refer to an element outside of the orders contained in this array must be designated as X'FFFF'.

count
: is an integer constant or integer variable that defines the number of bytes of graphic orders that are in the array identified by the "arrayname" argument.

keylist
: is a variable or an array name that identifies a list of full-word entries, each containing a key. As the value X'FFFF' is found in address references in the graphic orders in the array identified by the "arrayname" argument, the key identifying the element being referred to is taken from this list. If the value X'FFFF' appears four times in the array identified by the "arrayname" argument, this key list must be a four-word array of keys.

The programmer can use the standard keying and correlating features provided in GSP to identify elements or can use his own correlation scheme to identify the elements. This choice is made on a graphic data set basis and is indicated by the value assigned to the "gdslevel" argument in the call to the INGDS subroutine. When the programmer uses his own correlation scheme, the GSP key and correlation value table for the specified graphic data set is not created or maintained.

Except for this appendix, all sections of this publication describe the use of the standard GSP keying and correlating features. These sections also apply to the use of the programmer's own correlation scheme, except as follows:

- The "corrval" argment, or a null variable in its position in the calling sequence, must be specified. However, the correlation value assigned as that argument is ignored.

- The value assigned by GSP as the "key" argument is not retained by GSP. It is computed as follows:

  Key Value = length x 2^{16} + logical buffer address

  Where:

  length is the number of bytes of the element generated.

  logical buffer address is the number of bytes of orders from the beginning of the graphic data set to the first byte of data for the keyed element, sequence, or buffer subroutine. The actual buffer address of the keyed item does not affect the value of the logical buffer address.

  Notes:

  1. Identical input data will not always occupy the same amount of space in the 2250 buffer. Therefore, the length of the element may differ from its original size. This must be considered in any programmer correlation scheme.

  2. Transfer orders resulting from the dynamic allocation of a new buffer segment when the existing buffer segment is filled are not included in the length or logical buffer address factors used to compute the value of the key.

- A value of two cannot be assigned as the "gencode" argument. That is, elements or buffer subroutines cannot be placed in the omit status when they are created.

- The INCL and OMIT subroutines can only refer to graphic data sets. Keyed elements or buffer subroutines cannot be placed in the include or omit status by means of these two subroutines. The key must be omitted, or the null variable substituted for it, in a call to either of these subroutines.

- Elements cannot be grouped into sequences. Calls to the BGSEQ and ENSEQ subroutines are ignored.

- Buffer subroutines must be keyed. That is, the "key" argument must be specified in the call to the ENSUB subroutine.

- The Force a Set Mode Order (FSMOD) subroutine, described below, may be used to facilitate the updating of elements.

FSMOD--Force a Set Mode Order

The FSMOD subroutine causes an appropriate set mode order to be generated immediately preceding the data that follows an element to be updated. If the FSMOD subroutine is not called, image generation subroutines will not generate set mode orders unless necessary. For example, if the last element in a graphic data set is a text element and the PLINE subroutine is then called repeatedly to produce a single unkeyed line for each call, only the first call to the PLINE subroutine generates a set mode order for producing lines.

General Form
CALL FSMOD(gdsname)

gdsname
is described in "Arguments Used by Many GSP Subroutines."

The example that follows depicts the use of multiple attention levels in a GSP program. In the example, an attention level identified by the value of the variable LEVEL is established for the 2250 identified by the value of IGRAFD (statement 100) and a call is issued to a graphic subprogram identified as FCTAB (statement 200). The call to FCTAB passes the value of the "devicename" argument (hence, of IGRAFD) to that subprogram. FCTAB identifies that value as INAME (statement 300) and establishes its own attention level identified as ATL (statement 310). The creation of the new attention level (ATL) makes the previously established attention level (LEVEL) inactive, thus preventing subprogram interference with the calling program's attention processing routines. Before subprogram FCTAB returns control to the calling program, it ends its attention level (ATL), thereby reactivating the calling program's attention level (LEVEL).

```
      .
      .
      .
100   CALL   CRATL(IDEV,LEVEL)
      .
      .
      .
200   CALL   FCTAB(IDEV,X,Y)
      .
      .
      .
300   SUBROUTINE   FCTAB(INAME,X,Y)
310   CALL   CRATL(INAME,ATL)
      .
      .
      .
400   CALL   ENATL(ATL)
      RETURN
      END
```

APPENDIX H: DIMENSIONS OF STANDARD 2250 CHARACTERS

Following is a table listing the dimensions and spacing of the characters produced by the 2250 character generator:

Character	Character Size	
	Basic	Large
Characters per line (maximum)	74	49
Lines per display (maximum)	52	35
Number of characters on display (maximum)	3,848	1,715
Spacing between characters (raster units -- center to center)	14	21
Spacing between characters (inches -- center to center)	0.164	0.246
Spacing between lines of characters (raster units -- center to center)	20	30
Spacing between lines of characters (inches -- center to center)	0.234	0.351

This appendix provides a quick reference to statements for invoking GSP subroutines and functions. The subroutines are listed first, followed by the functions. These subroutines are listed as they appear in the preceding sections of this publication.

GSP SUBROUTINES

Initiation

```
CALL INGSP(gspname,null)
CALL INDEV(gspname,unit,devicename[,gdoalength])
CALL INGDS(devicename,gdsname[,gdoalength][,gdslevel]
        [,gdsname1...,gdsname49]
CALL SPEC(gspname,code,rtnumber[,rtnumber...])
```

Termination

```
CALL TMGDS(gdsname)
CALL TMDEV(devicename)
CALL TMGSP(gspname)
```

Option Definition

```
CALL SDATM(gdsname,xmode[,ymode])
CALL SGRAM(gdsname,gmode)
CALL SCHAM(gdsname,mode)
CALL SGDSL(gdsname,gllx,glly,gurx,gury[,sllx,slly,surx,sury])
CALL SDATL(gdsname,xlim1,ylim1,xlim2,ylim2)
CALL SSCIS(gdsname,scissoring)
```

Image Generation

```
CALL MVPOS(gdsname,xcoor,ycoor[,corrval][,key][,gencode])
CALL STPOS(gdsname,xcoor,ycoor[,corrval][,key][,gencode])
CALL PLINE(gdsname,xcoor,ycoor[,corrval][,key][,gencode][,count]
        [,xindex][,yindex][,xincr][,yincr])
CALL PPNT(gdsname,xcoor,ycoor[,corrval][,key][,gencode][,count]
        [,xindex][,yindex][,xincr][,yincr])
CALL PSGMT(gdsname,xstart,ystart,xend,yend[,corrval][,key][,gencode]
        [,count][,xstartindex][,ystartindex][,xendindex][,yendindex]
        [,xstartincr][,ystartincr][,xendincr][,yendincr])
CALL PTEXT(gdsname,text,count[,corrval][,key][,gencode][,xcoor,ycoor])
CALL STEOS(gdsname[,corrval][,key][,gencode])
```

Identification

```
CALL BGSEQ(gdsname[,corrval][,key][,gencode])
CALL ENSEQ(gdsname[,key])
CALL BGSUB(gdsname[,corrval][,key][,gencode])
CALL ENSUB(gdsname[,key])
CALL LKSUB(gdsname[,bufcorrval][,bufkey][,linkcorrval][,linkkey]
        [,gencode])
```

Image Control

```
CALL RESET(gdsname[,corrval][,key])
CALL IDPOS(gdsname,xlast,ylast[,xcurr,ycurr])
CALL EXEC(gdsname)
CALL INCL(gdsname[,corrval][,key])
```

```
CALL OMIT(gdsname[,corrval][,key])
CALL ORGDS(gdsname1[,gdsname2...,gdsnamen])
```

Keyboard Input & Buffer Data Analysis

```
CALL ICURS(gdsname[,corrval][,key][,charpos])
CALL RCURS(gdsname)
CALL GSPRD(gdsname,storageloc,count,rdtype[,termcode][,corrval1][,key1]
          [,corrval2][,key2])
```

Attention Related

```
CALL CRATL(devicename,attnlevel[,dequectl])
CALL ENATL(attnlevel[,rangecode])
CALL ENATN(attnlevel,attnsource[,attnsource...])
CALL DSATN(attnlevel,attnsource[,attnsource...])
CALL SLPAT(gdsname,detect)
CALL RQATN(attnlevel,codeloc,wait[,arrayname],attnsource
          [,attnsource...])
CALL MLITS({devicename|attnlevel},status[,lights...])
CALL MLPEO(attnlevel,attntype[,info][,restart])
CALL MPATL(attnlevel,direction[,relattnlevel])
CALL SALRM(devicename)
```

Light Pen

```
CALL LOCPN(gdsname,xpos,ypos)
CALL BGTRK(gdsname,xcoor,ycoor)
CALL ENTRK(gdsname)
CALL RDTRK(gdsname,xpos,ypos)
```

Stroke Generator

```
CALL DFSTR(gdsname,table,strokes,symbol,strokecount)
CALL PLSTR(gdsname,table,text,count,height[,width][,spacing]
          [,orientation][,corrval][,key][,gencode]
          [,xcoor,ycoor])
```

Miscellaneous

```
CALL CNVRT(gdsname,convert[,xinput][,yinput][,xoutput][,youtput])
CALL ORGEN(gdsname,arrayname,count[,keylist][,corrval][,key][,gencode])
CALL FSMOD(gdsname)
```

GSP FUNCTIONS

```
ITRC(gspname,code)
ITBP(gdsname,info)
RTBP(gdsname,info)
ITST(gdsname,optionsub)
```

Where more than one reference is given, the first page number indicates the major reference.

READER'S COMMENT FORM

IBM System/360 Operating System: Graphic Programming Services for FORTRAN IV; Program Number 360S-LM-537

C27-6932-2

Please check or fill in the items below, adding explanations and other comments in the space provided.

Which of the following terms best describes your job?

¤ Programmer	¤ Systems Analyst	¤ Customer Engineer
¤ Manager	¤ Engineer	¤ Systems Engineer
¤ Operator	¤ Mathematician	¤ Sales Representative
¤ Instructor	¤ Student/Trainee	¤ Other (explain) _____

Does your installation subscribe to the SRL Revision Service? ¤ Yes ¤ No

How did you use this publication?

¤ As an introduction
¤ As a reference manual
¤ As a text (student)
¤ As a text (instructor)
¤ For another purpose (explain) _____

Did you find the material easy to read and understand? ¤ Yes ¤ No (explain below)

Did you find the material organized for convenient use? ¤ Yes ¤ No (explain below)

Specific criticisms (explain below)

Clarifications on pages _____

Additions on pages _____

Deletions on pages _____

Errors on pages _____

Explanations and other comments:

Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

FOLD FOLD

FIRST CLASS
PERMIT NO. 116
KINGSTON, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
NEIGHBORHOOD ROAD
KINGSTON, N. Y. 12401

ATTN: PROGRAMMING PUBLICATIONS
DEPARTMENT 637

FOLD FOLD

Printed in U.S.A. C27-6932-2

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Printed in U.S.A. C27-6932-2

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]