File No. S360-48
GC28-2006-1

TSS

IBM Systems Reference Library

Version 8.1

IBM System/360 Time Sharing System

Time Sharing Support System

The Time Sharing Support System is an on-line program error analysis facility that provides the capability of collecting data from the Time Sharing System/360 for analysis, and of altering the TSS/360 storage and machine registers. This system is used only by system programmers with authority code O or P and is not intended to be available to any other TSS/360 users.

The functions of TSSS may be performed on command from a terminal or dynamically during TSS/360 execution. The programs, tables, and control blocks of real, virtual, and secondary storage can all be referred to and modified.

Part I of this publication describes the TSSS system and its capabilities in a general way. Part II describes the TSSS command language, defining the functions of the language elements and the language syntax. Part III presents additional requirements for correct use of TSSS.

Prerequisite Publications

The reader must be familiar with the information contained in:

IBM System/360 Principles of Operation, GA22-6821

IBM System/360 Time Sharing System:
Concepts and Facilities, GC28-2003
System Programmer's Guide, GC28-2008

IBM System/360 Model 67: Functional Characteristics, GA27-2719

PREFACE

This publication is intended primarily for system programmers who are authorized to use the Time Sharing Support System (TSSS). The manual describes the capabilities of TSSS and how the system is invoked and operated by a system programmer. This publication does not describe when to invoke TSSS or how to apply its capabilities.

The publication is organized as follows: Part I is an overview of TSSS and may be of interest to parties other than system programmers; it describes the system and its capabilities in a generalized way. Part II describes the unique TSSS command language, defining the functions of the language elements and the language syntax. Part III presents the additional requirements for correct use of TSSS. Parts II and III constitute, in effect, a handbook for using TSSS.

A Time Sharing System/360 system programmer is assumed to be familiar with IBM System/360 Principles of Operation, GA22-6821, and with most IBM System/360 Time Sharing System publications. Among the latter, the following constitute minimum prerequisite reading for a full understanding of this publication:

Concepts and Facilities, GC28-2003

System Programmer's Guide, GC28-2008

IBM System/360 Model 67: Functional Characteristics, GA27-2719

Information required for use of TSS/360 terminals and the Operator's 1052-7 Printer-Keyboard is not completely duplicated in this publication. The following TSS/360 publications should be referred to:

Terminal User's Guide, GC28-2017

Operator's Guide, GC28-2033

For effective use of TSSS, the system programmer probably will require the following IBM System/360 Time Sharing System Program Logic Manuals (PLMs) for reference purposes:

System Logic Summary PLM, GY28-2009

System Control Blocks PLM, GY28-2011

The Program Logic Manuals for the programs of immediate concern; these are listed, along with other TSS/360 publications of interest to system programmers, in the System Programmer's Guide.

The TSSS user may wish to become familiar with the internal logic of TSSS; the publication that meets this need is IBM System/360 Time Sharing System: Time Sharing Support System PLM, GY28-2022.

Second Edition (September 1971)

This is a revision of, and makes obsolete, GC28-2006-0 and Technical Newsletters GN28-3043, GN28-3062, and GN28-3144. Minor changes have been made to the descriptions of the $PATCH system symbol and the REMOVE command, and cautions have been added concerning the use of the $RM symbol in VSS and implanting an AT in RSS. Changes on the pages are indicated by a vertical bar to the left of the change.

This edition is current with Version 8, Modification 1 of the IBM System/360 Time Sharing System (TSS/360), and remains in effect for all subsequent versions or modifications of TSS/360 unless otherwise noted. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters. Before using this publication, refer to the latest edition of IBM System/360 Time Sharing System: Addendum, GC28-2043, which may contain information pertinent to the topics covered in this edition. The Addendum also lists the editions of all TSS/360 publications that are applicable and current.

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 Printer using a special print train.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Time Sharing System/360 Publications, Department 643, Neighborhood Road, Kingston, New York 12401

ILLUSTRATIONS

INTRODUCTION

The Time Sharing Support System (TSSS) is a subsystem within the Time Sharing System/360 operating environment. The purpose of TSSS is to provide capability for on-line error analysis and modification of system programs with minimal dependence on the system it supports.

Minimal system disturbance when TSSS is called upon is a primary objective, but realization of the objective is necessarily in the hands of the TSSS user. In any case, the presence of TSSS is unnoticed by TSS/360 users when TSSS has not been activated.

Because of the unique function and capability of TSSS, it is described in this publication as if it were totally self-contained and independent of the System/360 Time Sharing System (abbreviated "TSS/360" in this context), which in turn is considered as the system exclusive of TSSS unless otherwise specified. In this way, the functions of TSS/360 and of TSSS may be contrasted with each other and the transfer of control between them may easily be described.

TSSS CONCEPTS: FUNCTIONAL CAPABILITIES

The principal purpose of TSSS is to allow system programmers to selectively gather data for analysis of system program errors, and to dynamically correct those errors while TSS/360 is running. TSSS is essentially a maintenance tool operated from a keyboard terminal; it is not to be confused with programs that automatically attempt to recover from hardware malfunctions.

TSSS should not be confused with the TSS/360 command system subset that appears to be similar to the TSSS command language. That command subset is for problem programmers; it employs the TSS/360 command system and relies on the resident supervisor and privileged virtual programs that TSSS is designed to monitor and analyze.

TSSS provides access to all real, virtual, and secondary storage and to machine registers. TSSS operations may be initiated directly from the terminal or may be initiated at predetermined points during TSS/360 execution. In the latter case, the AT command (defined in detail later) is used to implant "ATs" in executable code and thereby cause the subsequent dynamic initiation of TSSS operations.

The TSSS user has the power to alter as well as to inspect any part of any TSS/360 program, table, or control block, and thus it is incumbent on him to exercise care and restraint. The power to perform dynamic maintenance on a system as complex as TSS/360 is also the power to destroy.

Figure 1 is an illustration of the TSSS relationship to TSS/360, as described in the following paragraphs.

TSSS CONCEPTS: RSS AND VSS

The Time Sharing Support System actually is two systems in one, providing two modes of operation. A Resident Support System (RSS) is the more powerful of the two; when it is invoked, TSS/360 activity is temporarily suspended. RSS is very nearly independent of TSS/360. It has its own input/output capability and may use any device in the current configuration. RSS has access to real storage and to the virtual storage of all current tasks.

The Virtual Support System (VSS) is, from external appearances, very nearly a copy of RSS; it performs the same basic functions, but VSS executes within the time-shared TSS/360 environment.

RSS, then, generally does not rely on the TSS/360 resident supervisor and is not time sliced; the opposite is true of VSS. Further, RSS is not associated with any TSS/360 task, whereas VSS is invoked within a specified, existing task. VSS resides in each task's Initial Virtual Storage and thus may be activated within each task. Nonetheless, VSS has the power to address real storage as well as the virtual storage of the task within which it is active.

Whether RSS or VSS is executing, the use of the TSSS command language is identical and the functions performed are essentially identical. The user who employs this command language is therefore classified according to the mode in which he has invoked TSSS, RSS mode or VSS mode.

TSSS CONCEPTS: TWO USER CLASSES

The TSSS user has been joined with system programmer authority (authority code O or P), which is thereafter recognized by

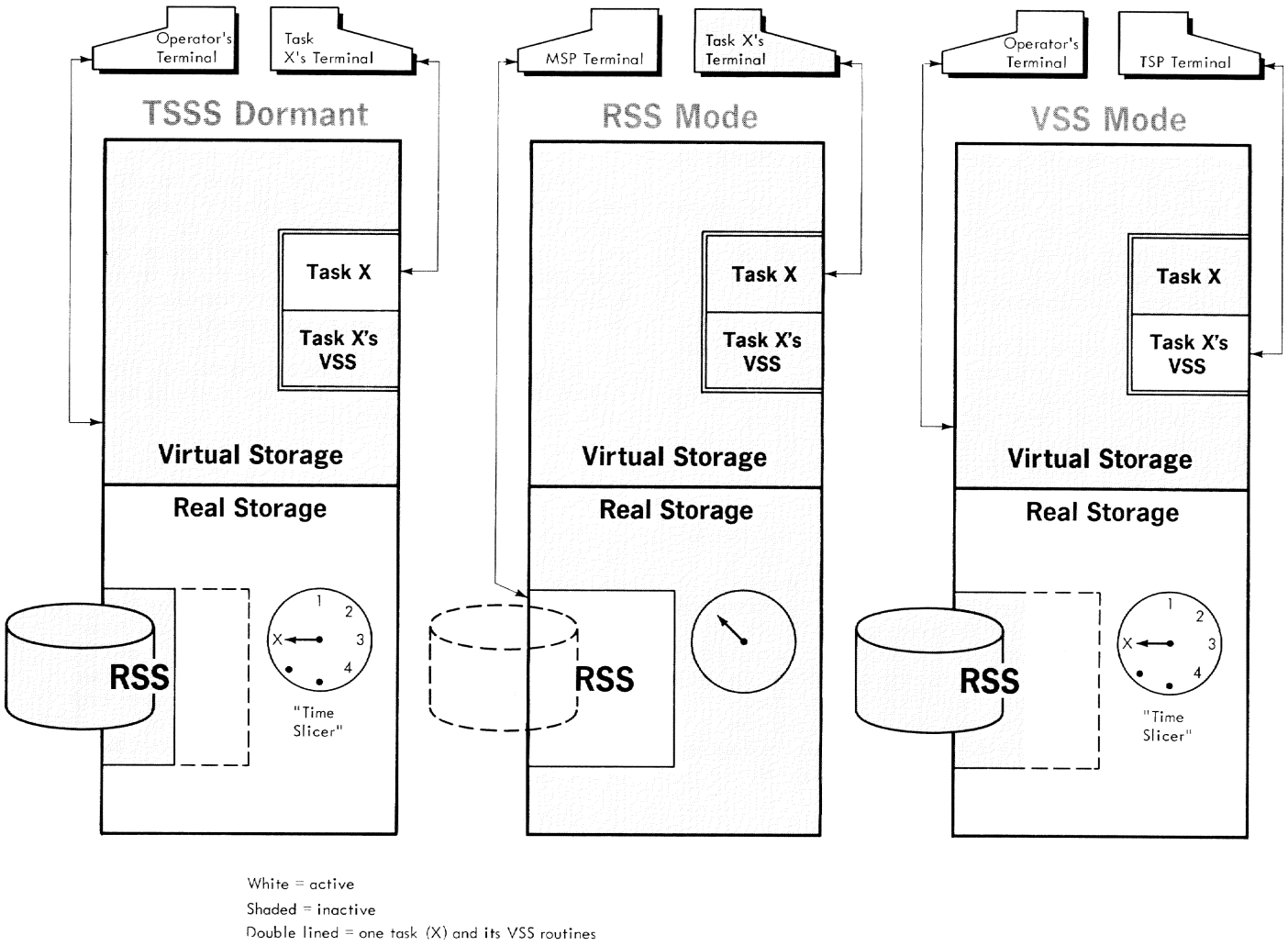


Figure 1. The relationships of RSS, VSS, and TSS/360 in three modes of operation

the TSS/360 command system via his user identification (userid). A system programmer may use TSS/360 without invoking TSSS, however; to make the distinction clear, a TSSS user is referred to in this publication as a "System Programmer" (initial capital letters).

There are two classes of System Programmers: When RSS is connected to a terminal, the user of that terminal is a Master System Programmer (MSP); when VSS is connected to the terminal, the user is a Task System Programmer (TSP). ("Connected" is used here to denote MSP or TSP capability at a given terminal, whether or not TSSS is currently executing on behalf of that System Programmer.)

In describing TSSS, user classification often is irrelevant. The term System Programmer is then used, and in certain instances it is abbreviated "SP."

It is important to note that only one MSP can be connected at a given time (regardless of how many individuals may have the authority to function as an MSP), and that multiple TSPs may be connected at a given time but only one per task.

MACHINE CONFIGURATION

The Time Sharing Support System may be used with any machine configuration of TSS/360; system generation as described in IBM System/360 Time Sharing System: System Generation and Maintenance, GC28-2010, together with the Startup procedures performed by the Operator, will prepare TSS/360 so that TSSS may be invoked. Any terminal supported by TSS/360 may be used as a

TSSS input device. Specifically, the following input/output devices[1] are supported by TSSS:

- IBM 1050 Data Communications System: 1052 Printer-Keyboard, Model 2, and 1056 Card Reader only; attached via a 2702 Transmission Control
- IBM 1052 Printer-Keyboard, Model 7 (the CPU console keyboard device)
- IBM 2741 Communications Terminal, attached via a 2702 Transmission Control
- Teletype Model 33 or 35 KSR teletypewriter (a product of the Teletype Corporation)
- IBM 2540 Card Read Punch, read only
- IBM 2301 Parallel Drum
- IBM 2311 Disk Storage Drive
- IBM 2314 Direct Access Storage Facility
- IBM 2401 Magnetic Tape Unit, Models 1, 2, and 3
- IBM 1403 Printer, Models 2 and N1

Note: The different keyboard input devices are not identical in respect to certain terminology and procedures. The 1052 REQUEST key, the 2741 Attention key, and the teletypewriter equivalent are all referred to as the Attention key in this publication. The signal that the end of an input string has been reached, whether called end-of-block, end-of-message, or end-of-transmission elsewhere, is referred to as end-of-block (EOB) in this publication.

SYSTEM SUMMARY

This section discusses in greater detail the relationship of RSS and VSS to each other and to TSS/360, defines the modes of operation for TSSS users and how each is established, and introduces basic concepts of using TSSS through its unique command language.

---

[1]Terminals that are equivalent to those explicitly supported may also function satisfactorily. The customer is responsible for establishing equivalence. IBM assumes no responsibility for the inpact that any changes to the IBM-supplied products or programs may have on such terminals.

CONNECTING THE SYSTEM PROGRAMMER TO RSS OR VSS

A Master System Programmer (MSP) may connect a terminal to RSS in only one way. A Task System Programmer (TSP) may connect a terminal to VSS, or he may have a terminal connected to VSS on his behalf by a connected MSP. In brief, these procedures are:

- Pressing a CPU external interruption key connects the Master System Programmer at the Operator's terminal.
- With a conversational task active at a terminal, the VSS command of the TSS/360 command system connects a TSP at that terminal.
- The MSP may issue the CONNECT command to connect a TSP at the terminal of a logged-on conversational task.

The methods of connecting a System Programmer are defined in detail in Part III and under "CONNECT Command" in Part II.

Note that the term "connected" is used to denote MSP or TSP capability at a terminal. The MSP can be connected only to RSS. A TSP may be said to be connected either to VSS or to a task; the meaning is the same. The period of time during which a System Programmer is connected is called a TSSS terminal session.

MODES OF OPERATION

For convenience in describing TSSS, several modes of operation and states are defined as follows:

- RSS mode means that RSS is executing and TSS/360 operation is suspended. No VSS activity is possible. Usually the MSP is connected, but it is possible to be in RSS mode with no MSP connected; in that case, RSS is performing a service for VSS.
- VSS mode means that VSS is executing within the time slice of a given task or is in control of the task and is awaiting a time slice. VSS mode applies only to the specified task, and usually a TSP is connected to that task. However, VSS mode may occur for any task as a result of global ATs implanted by at least one connected TSP.
- Call mode is concurrent with RSS mode or VSS mode; it means that a card reader or tape drive is currently the input device (see "CALL Command"). The MSP (RSS mode) or a TSP (VSS mode) must be

connected. Call mode is either immediate or dynamic, as described later in this section under "Input Statements."

- Run mode (or TSS/360 mode) means that a connected System Programmer has issued the RUN command and that TSS/360 has resumed execution, in the case of the MSP, or the task has resumed execution, in the case of a TSP.

QUALIFICATION STATES

For TSSS operations, qualification is defined as a state, controlled by the System Programmer, that determines whether storage addresses shall be resolved as real storage addresses, virtual storage addresses, or secondary storage addresses.

The term used to specify real (non-relocatable) storage as the subject state is "real memory" -- real memory is a TSSS qualifier. The term used to specify the virtual storage of a given task as the subject state is "virtual memory."

External qualification (referencing secondary storage) is the implied state for certain operations. As a state, it exists only for the duration of the operation.

"Global" is a unique TSSS qualification state. It is established through the same procedures used to establish real memory or virtual memory qualification. However, once established, the global state pertains to a mode of execution, rather than to specification of unique storage addresses, as described below under the heading "Global Operations."

Either the MSP or a TSP may operate under each of the qualification states, within certain limitations that are defined elsewhere. It should be noted that the use of real memory qualification always implies running in RSS mode, at least for a brief interval. VSS must call upon RSS to perform certain functions it cannot perform for itself.

Global Operations

The preceding list of modes of operation does not include one unique category -- global operations. The reason is that, in addition to special considerations regarding global operations, they always occur while in run mode from the System Programmer's point of view; global operations are initiated during TSS/360 execution by dynamic statements that in turn are triggered by "global ATs."

A global AT is one in shared virtual storage whose execution by any task, whether or not a TSP is connected to that task, causes the associated dynamic statement to be executed. If the global AT was implanted by the MSP, RSS mode is established for the duration of execution of the statement; if implanted by a TSP, VSS mode is established, for each task executing the AT, for the duration of execution of the statement. Global operations for a TSP are the exception to the general rule that a TSP must be connected to a task for VSS to be activated.

All relevant aspects and implications of global operations are described and defined in appropriate sections of this publication; see "AT Command" in Part II and "Global Qualification" in Part III in particular.

INPUT/OUTPUT OVERVIEW

TSSS accepts command statements written in the TSSS command language, interprets those statements, and performs the requested operations. There are very few restrictions on the System Programmer; he bears responsibility for correct use of the TSSS capability.

Input Statements

Figure 2 presents a summary of the TSSS commands and their functions. With the exception of CONNECT, they are used in essentially the same manner for the same purposes by both the MSP and a TSP. A detailed discussion of the commands and their use appears under the heading "Time Sharing Support System Commands." "Language Elements and Notation" defines terminology and describes the ways in which valid command operands are formed.

One or more commands, up to a 256-character maximum, form an input statement. Statements are typed at the System Programmer's terminal or may be issued, after a terminal session has begun, via a predefined statement set on cards or tape. No other external data sets of its own are used by TSSS; the data operated upon consists primarily of TSS/360 programs, tables, control blocks, and status information.

Each TSSS command statement is either "immediate" or "dynamic"; an immediate or a dynamic statement may also be "conditional."

Immediate Statements: A command statement that is processed and executed in its entirety upon receipt of the input is called an immediate statement.

Command	Function
AT	Designates a dynamic statement and the point in TSS/360 execution at which it is to be executed.
CALL	Initiates the execution of a prestored set of command statements.
COLLECT	Moves data from a specified area into a specified collection area.
CONNECT	Causes a TSP to be connected to VSS at the terminal of a specified task. Valid for an MSP only.
DEFINE	Enables the SP to define temporary symbols and allocates storage when necessary.
DISCONNECT	Removes the SP capability from the terminal, restores TSS/360 (except for patches), and permanently transfers control to TSS/360.
DISPLAY	Writes data requested by an SP on his terminal.
DUMP	Writes data requested by an SP on a specified output device.
END	Terminates reading of a device being used for input of prestored statement sets.
IF	Designates a conditional statement; execution of the statement is dependent on the predetermined condition.
PATCH	Alters the contents of a specified data field and keeps a record of the patch.
QUALIFY	Establishes implicit "real memory," "virtual memory," or "global" qualification for subsequent operands.
REMOVE	Deletes ATs and their associated dynamic statements, or deletes patches.
RUN	Causes control to revert to TSS/360; ATs can then be executed.
SET	Alters the contents of a specified data field.
STOP	Causes TSS/360 or a specific task to halt.

Figure 2. TSSS commands and their functions

If call mode is established during execution of an immediate statement, the call mode is considered to be immediate.

Dynamic Statements: All commands following an AT command in a given statement collectively constitute a dynamic statement. A dynamic statement is not executed at the time it is received as input but is saved for subsequent execution at a predetermined point during TSS/360 execution.

If call mode is established during execution of a dynamic statement, the call mode is considered to be dynamic.

If an AT command appears after another AT command in an input statement, a dynamic statement is embedded within another dynamic statement.

Conditional Statements: The presence of an IF command in an immediate or dynamic statement signifies that the remainder of the statement is conditional; it will be executed only if the predetermined condition is satisfied. A conditional statement may be embedded within another conditional statement.

TSSS Output

TSSS is capable of producing hard-copy output in the form of printer dumps and terminal displays, or dumps to tape for subsequent printing. Many operations performed by TSSS are internal, such as defining working storage areas, collecting data into specified areas, or modifying data. No physical output is generated automatically by TSSS except for diagnostic messages written at the System Programmer's terminal. (The character $ is written at

the System Programmer's terminal to invite input, but this output operation has no other significance.)

TSSS diagnostic messages are divided into four classes: Class 0 messages result from permanent I/O errors; Class 1 messages result from System Programmer errors; Class 2 messages result from system errors; and Class 3 messages result from errors in loading and unloading of the transient portion of RSS. (RSS is partially resident and partially transient, whereas VSS resides in each task's Initial Virtual Storage.)

The only documentation of a TSSS terminal session is that which the System Programmer produces, with and without the assistance of TSSS.

The Time Sharing Support System employs a self-contained, interpretive language processor, with separate but basically identical versions in RSS and VSS, that reads the input device and performs the operations requested by a System Programmer. TSSS accepts input written in a unique command language whose syntax is both simple and flexible, enabling the System Programmer to specify a wide variety of operations to be performed by TSSS.

A general discussion of language elements and notation introduces the command language description, followed by detailed definitions of all language elements.

LANGUAGE ELEMENTS AND NOTATION

Conventions and Notational Symbols

A group of notational symbols is used in format representations, in accordance with conventions governing their use, to lend precision and ease of understanding to the syntax of the TSSS command language.

Within each format representation, any item written in capital letters is to be written by the System Programmer exactly as shown (except that they may be typed at a terminal in lowercase letters). Items written within format representations in lowercase letters are terms for which specific values are to be substituted. These terms are either self-defining according to standard usage (e.g., "address," "parameter") or are defined in this publication (e.g., "data field," "system symbol").

TSSS commands have no keyword operands of the type used in the TSS/360 command system (KEYWORD=parameter), nor are there positional operands in the TSS/360 sense. There are positional parameters that are described fully with their occurrence.

The following notational symbols are used in format representations; they are not punctuation marks and must not be written by the System Programmer:

[ ] Brackets. These indicate that the item or items enclosed therein are optional and may be omitted at the user's discretion. If default values are assumed by TSSS, they are described in the accompanying text.

{ } Braces. These indicate that one of the alternative items stacked within the braces must be selected by the System Programmer.

[,...] Ellipsis within brackets. This is the form used to indicate that an operand format may be repeated; that is, it specifies that multiple operands are permitted. An entire operand (simple or compound, as defined in the introductory paragraphs under "Time Sharing Support System Commands"), in any form that is valid according to the format representation, must be supplied, separated from the preceding operand by a comma.

Note: Any operand that references real or virtual storage is implicitly or explicitly qualified. The explicit qualifiers $RM and $VM (see "System Symbols") may prefix any such operand. This option is not shown in the command format representations.

The Character Set

The character set used to construct TSSS command language elements consists of the following:

- the English alphabet, A through Z, upper or lowercase
- the arabic numerals 0 through 9
- the special characters + - * / < = > . , ; : ( ) ' & | ¬ % $ blank

The character $ has special uses in TSSS and is not included in the alphabet that is used to designate symbols other than system symbols.

The character @ is not accepted by TSSS except when within a character literal.

The character # is recognized by TSSS as a backspace character when the input device is a 1052 Model 7 Printer-Keyboard and cannot be used in any other way. For any other input device, the # is accepted by TSSS only when it occurs within a character literal. (The left arrow character on the Model 33 or 35 KSR teletypewriter keyboard is recognized by TSSS as the backspace character for that input device.)

It should be noted that graphic symbols may vary among devices. The correct bit configuration will be transmitted to TSSS

if characters are keyed in at a 1052 or 2741 terminal in accordance with the keyboard position as shown in IBM System/360 Time Sharing System: Terminal User's Guide, GC28-2017.

Language Elements

The building blocks of the TSSS command language are the commands, symbols, literals, and operators. The last three of these elements are combined to form operands, and there are several kinds of each type of element. The sections following describe all of these elements in detail, in the following order: data field defining items (exclusive of system symbols), literals (including constants), operators, system symbols, and commands. The characters used only as delimiters are defined by their use in format representations.

DATA FIELD DEFINING ELEMENTS

For the purpose of describing TSSS and its command language, a data field is defined as any storage area, measured in contiguous bytes, that is addressable in machine language and has a specific length attribute. A data field may thus exist in real storage, in virtual storage, or in secondary storage (on an external but online device); it may have or may be assigned additional attributes.

To define data fields in the TSSS command language, the System Programmer uses symbols, absolute addresses, indirect addressing, subscripting, immediate attribute designation, range, or combinations of these. Of the symbols, SP symbols and TSS/360 external symbols are discussed in this section; the numerous system symbols that define data fields are discussed in the section entitled "System Symbols."

Note: The term "symbol" is used in some of the format representations for data field defining elements to designate storage locations in a generalized way. In that context the term encompasses all types of symbols as well as other means of expressing storage addresses.

SP Symbols

A System Programmer may define data fields and assign symbolic names of his own choosing, as described in detail under "DEFINE Command"; these are called SP symbols. The data field defined by an SP symbol may exist in the System Programmer's working storage or it may be a TSS/360 data field.

An SP symbol carries with it in a symbol table the following data field attributes, shown here with their default values:

- base address - the actual storage address
- pointer - 0
- length - 1 byte
- type - hexadecimal
- size - same as length

base address
: may be a real or virtual storage address.

pointer
: is a value that is always added to the base address when a symbol is resolved, forming the effective address of the data field. Pointer is always zero when an SP symbol is defined and remains unchanged for all TSSS internal operations except execution of the COLLECT command; it should not be confused with "offset" (see "Immediate Attribute Designation" below).

length
: is the number of bytes that participate in an operation when the symbol is used by the System Programmer.

type
: indicates whether the data in the data field is to be treated as hexadecimal, decimal integer, or character (EBCDIC) information.

size
: is the total number of bytes in the data field.

The pointer and size attributes are intended primarily for use by the COLLECT command; nonetheless, all data fields are described internally by TSSS with all of the above attributes. The System Programmer may establish such attributes (except pointer) for a TSS/360 data field by defining that data field with the DEFINE command and thereby creating an SP symbol.

An SP symbol is private for the MSP and for a TSP, with one exception: If a TSP is operating under global qualification when the symbol is defined, the symbol is global. A private SP symbol is recorded in the System Programmer's SP Symbol Table; a global SP symbol is recorded in the Global Symbol Table, which is shared by all TSPs using the system from TSS/360 startup to TSS/360 shutdown.

The attributes of an SP symbol may be changed for a single use of the symbol with the use of immediate attribute designation (see below) or they may be referenced through appropriate use of the $B, $P, $L, $T, and $S system symbols (see "System Symbols").

Immediate Attribute Designation

Every data field is defined in accordance with its default attributes, according to type, or is defined for one-time reference by the System Programmer by using immediate attribute designation. Base address and pointer are not affected by use of this notation, but an offset from the base-plus-pointer address may be specified to modify the effective address of the data field. The format for immediate attribute designation is:

```
┌-------------------------------┐
|       symbol.(o,l,t,s)        |
└-------------------------------┘
```

symbol
refers to any representation of a data field that will resolve to a storage address.

o,l,t,s
represents specification of values for offset, length, type, and size, respectively. Offset, length, and size are normally expressed in decimal digits; type is normally coded I, X, or C to represent decimal integer, hexadecimal, or character, respectively. However, any expressions that can be resolved to appropriate values may be used.

Any of the "o,l,t,s" parameters may be omitted, but all commas to the left of the last one designated must be retained. Default values appropriate for the specified "symbol" are supplied by TSSS for length, type, and size; the default for offset is always zero. In the case of an SP symbol, the SP Symbol Table supplies the default values.

Offset is a displacement from an effective address; it is always specified by the System Programmer. An effective address is calculated by TSSS, prior to application of any specified offset, as the sum of base address and pointer attribute. Manipulating the value of the pointer is normally an internal function.

Note that the value of an offset is not subtracted from length in a given definition of a data field. For example, to specify only the last half of an eight-byte data field, immediate attribute designation should specify an offset of 4 and a length of 4; if length were defaulted, the data field referenced would consist of the four bytes desired plus the following four bytes. Note that symbols given type integer in immediate attribute format can not have a length attribute greater than four. If the length attribute is not given, the default length is one.

External Symbols

TSS/360 external symbols may be used by a System Programmer to specify the real or virtual storage address of a data field. An MSP may, with proper qualification, reference any external symbols in TSS/360. A TSP may reference external symbols of real storage and of the virtual storage addressable by the task to which he is connected.

When an external symbol is referenced by a System Programmer, it is assigned the following default attributes:

- pointer - 0
- length - 1 byte
- type - hexadecimal
- size - same as length

If different attributes are desired, immediate attribute designation must be used. By using an external symbol with an offset, a System Programmer may reference any location within a given control section. Storage maps of external symbols may be obtained with the $MAP system symbol (see "System Symbols").

Note: An MSP or TSP should not reference an external symbol which has a $ as its first character, as the $ as a first character indicates a system symbol in TSSS (e.g., $PSW). If the symbol used has the $ as a first character and is not a valid TSSS system symbol, it is rejected.

Absolute Addresses

Actual hexadecimal storage addresses, real or virtual depending on qualification, are represented with "L-notation." The format for L-notation is:

```
┌-------------------------------┐
|          L'xxxxxxxx'          |
└-------------------------------┘
```

xxxxxxxx
represents one to eight hexadecimal digits; leading zeros are supplied by TSSS as needed.

L-notation designates a data field whose default attributes are:

- pointer - 0
- length - 1 byte
- type - hexadecimal
- size - same as length

Immediate attribute designation can be used with L-notation to specify different length, type, and size attributes.

An example of L-notation is: L'13579B'

Indirect Addressing

The contents of a data field may be the address of another data field. The System Programmer may reference the latter data field via the address of the former (symbolic or otherwise) by using the special character percent sign (%) to designate indirect addressing. The format is:

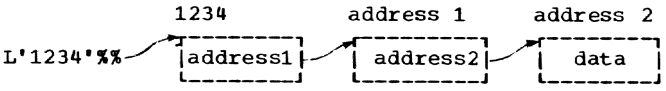
```
┌──────────────────────────────┐
│          symbol%             │
└──────────────────────────────┘
```

symbol
refers to any representation of a data field that will resolve to a storage address.

The effective address of "symbol" (base address plus pointer, plus offset if specified) need not be on a fullword boundary. It designates the starting address of a four-byte field that is assumed to contain an address.

When the indirect addressing operator is used in a TSSS operation, the data field addressed by that use has an implied length of four bytes and a type attribute of hexadecimal. Immediate attribute designation can be used to specify different attributes for the operation.

Multiple levels of indirect addressing can be specified. The following diagram depicts an example:



By including L'1234'%% in the operand of an appropriate command, the System Programmer would actually reference "data." By omitting the second percent sign, he would reference "address2" as the contents of a data field.

Subscripting

A data field may be treated as an array and a subscript used to reference an element in that array. The format for subscripting is:

```
┌──────────────────────────────┐
│          symbol(n)           │
└──────────────────────────────┘
```

symbol
refers to any representation of a data field that will resolve to a storage address.

n
is any expression, usually a decimal integer, that reduces to a value that will designate the desired element of the array. A subscript of 0 specifies the first element, 1 the second element, etc.; the value of "n" may be negative.

The element referenced by subscripting has the length of the length attribute of "symbol." If the System Programmer wishes to create an array in his working storage, he should define an SP symbol with a size attribute that is an appropriate multiple of the length attribute. The size attribute is irrelevant when subscripting TSS/360 data fields. (Note that a data field referenced with an external symbol or with L-notation has a default length attribute of one byte.)

An example of subscripting is: ARRAY(6)

This example references the seventh element of the data field ARRAY, or, put another way, it references a data field whose address is base-plus-pointer of the symbol ARRAY plus six times the length attribute of ARRAY, and whose length is equal to the length attribute of ARRAY.

Range

A data field may be specified as the range from one storage address to a higher storage address. Both addresses must exist in real storage, in the virtual storage of the same task, or on the same I/O device. The format is:

```
┌──────────────────────────────┐
│        symbol:symbol         │
└──────────────────────────────┘
```

symbol
refers to any representation of a data field that will resolve to a storage address. The address represented by "symbol" on the left of the colon must not have a value greater than that of "symbol" on the right.

The length of a range is computed as the second effective address minus the first

effective address plus the length attribute of the second data field representation.

The following are examples of range designation:

```
SYMB1:SYMB3

SYMB2:L'27F6'

SYMB4.(32):SYMB5.(,16)
```

In the first example, the length of the range is from the first byte of the data field named SYMB1 through the last byte of the data field SYMB3 (as specified by the length attribute of SYMB3).

In the second example, the range begins with the first byte of SYMB2 and ends with the byte whose actual hexadecimal address is 27F6.

In the third example, the range begins with the 33rd byte of the data field SYMB4 and extends through a data field named SYMB5 whose length for this use is 16 bytes.

LITERALS AND CONSTANTS

A literal in the TSSS command language is an item of immediate data in the input stream. Internally the literal becomes the contents of a data field, but a literal is not itself to be considered a data field in that it is not defined and thus cannot be addressed. There are three kinds of literals acceptable to TSSS: decimal integer, hexadecimal, and character.

Only one type of data is classified as a constant in the TSSS command language: the address constant. A literal may be called a constant (e.g., "character constant") without changing its definition, however.

Decimal Integer Literals

A decimal integer literal is written by the System Programmer as a string of arabic numerals. Its implied length is always four bytes, and a decimal integer literal's maximum absolute value is $2^{31}-1$. A decimal integer literal is translated internally into its binary equivalent and is treated as a 32-bit signed integer.

An example of a decimal integer literal is: 128

Note that the one-byte bit configuration for the above example is 10000000. If this binary value were contained in a one-byte data field of integer type, and if this data were used in an arithmetic operation, (see "Arithmetic Operators"), it would first be expanded to the four-byte bit configuration:

```
11111111  11111111  11111111  10000000
```

The result is a negative algebraic value (in complement form) instead of the binary representation of 128. Similar results may be obtained when performing arithmetic operations on decimal integer fields two or three bytes long.

Hexadecimal Literals

A hexadecimal literal is written by the System Programmer as a string of hexadecimal digits (0 through F), enclosed in apostrophes and preceded by X. Its maximum value is determined by length; the length is limited by the available space in the 256-byte input buffer.

An example of a hexadecimal literal is: X'13579BDF'

Character Literals

A character literal is written by the System Programmer as a string of characters enclosed in apostrophes and preceded by C. Except for the apostrophe, any character for which there is an EBCDIC graphic, including blank, may be included. If an apostrophe within the literal is desired, two apostrophes must be included in the input (one of which will be edited out internally).

In addition to the above special case, the character # ("pound sign") cannot be included in a character literal when the input device is a 1052-7 Printer-Keyboard. Instead, that character is recognized as a backspace character when keyed in at a 1052-7.

The length of a character literal is as many bytes as there are characters within the apostrophes (including blanks) minus edited apostrophes.

An example of a character literal is: C'2''S COMPLEMENT'

The length of the literal in the example is 14 bytes.

Address Constants

A storage address may be represented as an address constant, written by the System Programmer as a symbol enclosed in apostrophes and preceded by A. The value of the address constant is the actual storage address associated with the symbol (base address plus pointer).

When an address constant is operated upon, the address value itself is used, not the contents of the data field designated by the address.

An example of an address constant is: A'TABLE'

OPERATORS

Arithmetic, relational, and Boolean operators are provided for use with symbols and literals to form expressions. These expressions are used, in turn, in the formation of TSSS command operands.

Expressions are evaluated left to right, subject to the following hierarchy of operator precedence and the conventional rules of parenthesization:

1. unary minus
2. multiplication, division
3. addition, subtraction
4. greater than, less than, equal to
5. logical AND, logical OR
6. logical NOT

Note that if the symbol(s) that are operands designate a data field(s) on an external device(s) the operation will not be performed.

Arithmetic Operators

The conventional arithmetic operations addition, subtraction, multiplication, and division are specified by the characters +, -, *, and /, respectively. The minus sign (-) may also be used as a unary operator to denote a negative value. The arithmetic operators are combined with symbols and literals to form arithmetic expressions.

For example, 3+6*2 is an arithmetic expression; operator precedence makes its value 15. The same expression with parentheses added, so that it becomes (3+6)*2, is evaluated as 18.

The result of an arithmetic operation is a single value in hexadecimal or decimal; it is hexadecimal in all cases where the parameters entering into the operation are not all of decimal integer type. Any remainder from division is lost; only the quotient is saved. A negative result from any arithmetic operation is carried in complement form.

Arithmetic operations are performed with the fixed-point instructions; each parameter is treated as a 32-bit signed integer. If the length of a parameter is not four bytes, the following rules apply:

- If both participating parameters are decimal integer type, and if either is less than four bytes in length, it is expanded to four bytes by propagating the leftmost bit. If either is more than four bytes long, it is truncated on the left.
- In all other cases, any parameter less than four bytes in length is padded on the left with zeros; any parameter more than four bytes long is truncated on the left.

When any representation of a data field is provided as a parameter for an arithmetic operation, the operation is performed on the contents of the data field, not on the address. For example, "A+1" results in one being added to the contents of the data field whose name is A. (This addition takes place in working storage; the original value of the data field A is unchanged.) "A.(1)," on the other hand, correctly specifies a one-byte offset from the address of A.

Relational Operators

The relational operators < > = denote "less than," "greater than," and "equal to," respectively. These operators are used to form logical expressions such as A<B (A less than B), A>B (A greater than B), or A=B (A equal to B). The parameters compared when a relational operator is executed can be the following:

- contents of data fields
- literals
- results of arithmetic operations
- Boolean expressions
- any combination of these

The following are examples of logical expressions containing relational operators:

```
A+B>X'FF'

(A|B)=X'FF'
```

A logical expression is evaluated as either "true" (an indicator byte is set to ones) or "false" (the byte is zeroed); it is meaningless unless used as the operand of the IF command.

A comparison caused by execution of a relational operator is left-to-right for as

many bytes as are specified by the length attribute of the shorter parameter. The comparison is logical, not algebraic, and begins with the leftmost byte of both fields unless both parameters are of decimal integer type. If both have a type attribute of decimal integer, the longer parameter is truncated on the left before the comparison is performed.

For example, given the expression A=64 where A is a one-byte data field, if A is decimal integer type its contents are compared with the binary representation of 64 (64 is a four-byte decimal integer literal). If A has any other type attribute, it is compared with the high-order (leftmost) byte of the literal 64, which contains zeros.

Note that the equal sign (=) is also used in the format of the COLLECT, DEFINE, PATCH, and SET commands, in which instances it is not a relational operator.

Boolean Operators

The Boolean operators &, |, and ¬ represent the logical functions AND, OR, and NOT, respectively. Each of them causes a bit-by-bit operation to be performed. The AND and OR operations are exactly as indicated for the comparable machine instructions in IBM System/360 Principles of Operation, GA22-6821. The NOT operation inverts the bits of the specified data.

AND and OR may be used in the sense of "both" and "either" in the operand of an IF command by using the Boolean operator between two logical expressions containing relational operators. When that is done, each expression is evaluated separately, the result of which is a true or false indication in the form of a byte of all ones or all zeros, respectively. The AND or the OR operation is then performed on the two indicator bytes to produce a single true or false indication as the evaluated operand of the IF command.

Bit manipulation by a Boolean operator is not performed directly in a specified data field but rather in working storage. Thus, only with an appropriate command (such as SET) will data actually be changed with a Boolean operator.

The parameters that enter into a Boolean operation can be the following:

- contents of data fields
- literals
- results of arithmetic operations
- evaluated logical expressions
- any combination of these

The following are examples of Boolean expressions:

```
A&B

MASK|X'02'

¬(A<B)
```

Note that the third example is the equivalent of specifying "A equal to or greater than B."

The following are examples of Boolean expressions comprised of logical expressions:

```
A=B|A=C

(A=B|A=C)&D=X'FF'
```

In the first example, the entire expression would be evaluated as true if either A equals B or A equals C. In the second example, the entire expression would be evaluated as true only if the above is true and data field D contains X'FF'.

Note that a parenthesized Boolean expression within a logical expression (as in the second example under "Relational Operators") causes the bit manipulation to be performed prior to the relational comparison.

SYSTEM SYMBOLS

TSSS defines a group of "system symbols" for the System Programmer. These symbols, each of which has $ as its first character, define certain data fields or perform certain functions when used as command operands or within operands. They are grouped according to function and are discussed in the following order:

- $RM, $VM -- used to specify qualification.
- $B, $P, $L, $T, $S -- used to refer to SP symbol attributes.
- $R, $C, $E -- used to refer to machine registers.
- $PSW, $PPSW, $PPSW1, $PPSW2, $SPSW, $XPSW, $APSW, $TPSW, $IPSW, $MPSW, $VPSW -- used to refer to real or virtual program status words.
- $CAW, $CSW -- used to refer to the channel address word and channel status word.

- $TSKID -- used to refer to the identification number of the "current" task.
- $ID, $MAP -- used to retrieve external symbols.
- $IO, $VAM, $DOUT -- used to specify I/O devices.
- $AT, $PATCH -- used to refer to ATs and patches.
- $DHDR -- used to label output of the DUMP command.
- $STATUS, $TASK -- used to retrieve status indicators.

In addition to a definition of the function and a description of the attributes of each system symbol (with examples), a representation of the format is shown in each instance where a variable parameter is included. If no format representation is shown, the system symbol or symbols under discussion appear in the same format as symbols in general -- simply as a character string.

Those system symbols that represent data fields occur in RSS for the MSP domain and in VSS for a Task System Programmer. They are resolved with the System Programmer's default qualification, and any other implicit or explicit qualification is ignored.

The system symbols that represent data fields may be used in accordance with the rules for all data fields (e.g., with immediate attribute designation). It is the System Programmer's responsibility to determine what usage is appropriate in a given instance.

System Symbols Used for Qualification

Operands containing storage addresses are qualified either implicitly or explicitly as real memory, a virtual memory, or external (a data field in secondary storage). Real memory is the term applied to real storage qualification; virtual memory is the term used to specify as qualification the virtual storage of a task. The system symbols $RM and $VM are used to establish real memory and virtual memory qualification, respectively.

$RM or $VM may be used as the operand of the QUALIFY command to establish implicit qualification, or it may be used with an individual operand to establish explicit qualification for that operand only, overriding the existing state of implicit qualification. The format is:

```
+-------------------------------+
|       $RM[(n)][.]             |
+-------------------------------+
|       $VM[(n)][.]             |
+-------------------------------+
```

n

with $RM represents a CPU number in a duplex Model 67 configuration; its value may be 1 or 2. When it is specified, any operand that references a Prefixed Storage Area (PSA) will reference the PSA of the designated CPU. If "n" is omitted, PSA dedication is considered irrelevant. The optional period (.) is used when $RM precedes a command operand; it is omitted when $RM is the operand of the QUALIFY command.

n

with $VM specifies the virtual memory of a specific task in literal notation. Its value is the internal task identification number of the task. A TSP can specify only the task to which he is connected. When "n" is omitted, global qualification is specified for a TSP or the current task is specified by the MSP. The optional period is used when $VM precedes a command operand; it is omitted when $VM is the operand of a QUALIFY command.

Each System Programmer begins a terminal session in his default implicit qualification state: real memory (no CPU specified) for a Master System Programmer, or the virtual memory of the task to which the Task System Programmer is connected. Each use of the $RM or $VM system symbol will respecify the existing qualification, either for resolution of one or more storage addresses in a single use of a single operand or, if used with the QUALIFY command, until specified again.

For further details, see "QUALIFY Command" in the following section and "Modes of Operation" in Part I.

Caution: When using the $RM symbol in VSS to SET or PATCH, a page of real storage containing the target location to be changed is obtained, without TSS/360 activity being suspended. The entire page is replaced after the change has been made. Caution should be used, therefore, since results to the system are unpredictable.

The following examples depict a variety of formats in which the $RM and $VM system symbols would appear:

```
QUALIFY $RM(2)

 QUALIFY $VM
```

command $VM(13).operand

In the first example, implicit qualification of real memory is established for a System Programmer when the QUALIFY command is executed. If an operand refers to a PSA in a subsequent statement, the PSA of CPU 2 will be used.

In the second example, global qualification is established when the QUALIFY command is executed on behalf of a TSP, or the virtual memory of the current task is established as implicit qualification if the TSSS user is the MSP.

In the third example, the qualification of "operand" becomes the virtual memory of task 13 (decimal), overriding the implicit qualification.

All three examples are valid for an MSP. The first two are valid for a TSP; the third for a TSP only if he is connected to task 13.

Note that $VM(13) and $VM(X'0D') have the same value.

$B, $P, $L, $T, and $S System Symbols

As was pointed out in the discussion of SP symbols, the SP symbol attributes are carried with the symbol in the SP Symbol Table. The data fields containing this information may be referenced by the System Programmer with the $B, $P, $L, $T, and $S system symbols, referring to the base address, pointer, length, type, and size, respectively.

The format for this group of system symbols is:

$B(sp symbol)
$P(sp symbol)
$L(sp symbol)
$T(sp symbol)
$S(sp symbol)

sp symbol
: represents an SP symbol that was defined by the System Programmer during the current terminal session, or, under global qualification for a TSP, a global SP symbol that was defined by any TSP since the last TSS/360 startup.

The base address, pointer, length, and size attributes of an SP symbol are recorded in four-byte fields whose addresses are symbolically expressed with use of the $B, $P, $L, and $S system symbols. The type attribute of each of these fields is hexadecimal. The data field that designates the type of an SP symbol is one byte long and contains a hexadecimal code as follows:

- type is hexadecimal -- 01
- type is character -- 02
- type is decimal integer -- 03

The valid use of the $B, $P, $L, $T, or $S system symbol (i.e., when it is accompanied by an SP symbol that can be resolved under existing qualification) makes it a designation of a data field, subject to all rules applicable to data field designation.

The following are examples of this type of system symbol:

$P(COLLAREA)

$T(TABLE)

Assuming COLLAREA is an SP symbol that will be resolved with existing qualification, the first example above represents the four-byte data field containing the pointer attribute of the SP symbol COLLAREA. The second example designates a data field one byte long whose contents is a code that defines the type attribute of TABLE.

$R, $C, and $E System Symbols

General purpose, extended control, and floating point machine registers may be indirectly referenced with the $R, $C, and $E system symbols, respectively. The format is:

$R(n)
$C(n)
$E(n)

n
: represents the register number; its value may be 0 through 15 when used with $R or $C, and it may be 0, 2, 4, or 6 when used with $E.

The $R, $C, and $E system symbols actually represent data fields where the values contained in the registers when TSSS receives control are stored. Since the registers are loaded from those data fields when TSS/360 regains control, the System Programmer may alter the contents of TSS/360 registers as well as inspect the stored

values. If a range of registers is specified, the second must not have a lower value than the first; wraparound will not be performed.

The data fields referenced with this group of system symbols are four bytes long for $R and $C, eight bytes long for $E. In each case the default type is hexadecimal.

The following is a typical example of use of this type of system symbol, wherein all 16 general purpose registers are specified:

```
$R(0):$R(15)
```

The PSW System Symbols

A group of old program status words (TSS/360 extended mode) and virtual old program status words constitute data fields that may be referenced with appropriate system symbols. These are listed below as they exist in RSS and in VSS.

In RSS

System Symbol	Type of Old PSW
$PSW	"Current" PSW
$PPSW	Program Interrupt Old PSW
$SPSW	Supervisor Call Interrupt Old PSW
$XPSW	External Interrupt Old PSW
$IPSW	I/O Interrupt Old PSW
$MPSW	Machine Check Interrupt Old PSW

In VSS

System Symbol	Type of Old VPSW
$PSW	"Current" VPSW
$PPSW1	Recoverable Data Set Paging Error VPSW
$PPSW2	Program Interrupt Old VPSW
$SPSW	Supervisor Call Interrupt Old VPSW
$XPSW	External Interrupt Old VPSW
$APSW	Asynchronous I/O Interrupt Old VPSW
$IPSW	I/O Interrupt Old VPSW
$TPSW	Timer Interrupt Old VPSW
$VPSW	VSS Activation Old VPSW

The "current" PSW referenced with the $PSW system symbol by the MSP is actually the old PSW that was stored when RSS received control. The current PSW referenced with the $PSW system symbol by a TSP is the same as the one referenced by the system symbol $VPSW. This is the virtual PSW that was current for the task when VSS received control.

$PPSW1 references the old VPSW portion of the Recoverable Data Set Paging Error VPSW. The other PSW system symbols reference self-defining TSS/360 real and virtual PSWs. They represent the real and virtual old PSWs as they were at the time TSS/360 or the task was interrupted by activation of RSS or VSS.

Real PSWs as referenced by system symbols have a length of ten bytes -- eight bytes for the stored old PSW and two contiguous bytes for the corresponding interruption code. Virtual PSWs as referenced by system symbols have a length of eight bytes (the virtual PSW format includes the interruption code). Both have a type attribute of hexadecimal. The PSW system symbols represent data fields and are subject to all rules for data field designation.

$CAW and $CSW System Symbols

The $CAW and $CSW system symbols exist for the MSP only (i.e., in the RSS system symbol table). The $CAW system symbol is used to reference the channel address word. It is four bytes long and has a type attribute of hexadecimal.

The $CSW system symbol is used by the MSP to reference the channel status word. This data field is eight bytes long and has a type attribute of hexadecimal.

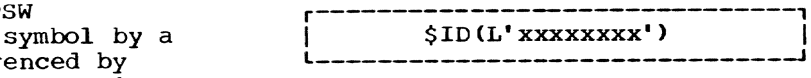
The channel address word and the channel status word are those that were current when RSS received control.

$TSKID System Symbol

The $TSKID system symbol represents a two-byte data field containing the identification number (hexadecimal) of the task that was current when RSS received control, for the MSP, or for a TSP of the task to which he is connected.

$ID System Symbol

The $ID system symbol makes it possible for a System Programmer to specify an actual address and have returned to him the CSECT, PSECT or entry point name whose address is nearest to but not greater than the address specified in RM. (In VM the symbols provided are only the nearest CSECTs or PSECTs.) The format is:



xxxxxxxx
: represents a hexadecimal number that is assumed to be a real or virtual storage address. The value specified may be from one to eight digits long; leading zeros will be supplied by TSSS as needed.

$MAP System Symbol

A storage map of the TSS/360 supervisor or of a specific task may be obtained with the $MAP system symbol. When $MAP is used as the operand of an appropriate command (usually DUMP) under real memory qualification, the external symbols and their hexadecimal addresses of the TSS/360 supervisor constitute the resulting storage map. In RSS the elements of the storage map are formatted for output in ascending order of the addresses. As an RSS option, the MSP may specify $MAP with type character and produce a map that is ordered alphabetically. In VSS the elements of the real storage map are always formatted in alphabetic ascending order.

When the $MAP system symbol is used as the operand of an appropriate command under a virtual memory qualification, the external symbols of the task's virtual storage and their virtual addresses (in hexadecimal) constitute the resulting storage map. The external symbols provided in a virtual memory storage map are:

- CSECT and PSECT names, in ascending order of their addresses.
- Entry point names within a given control section, in the order in which they occur in the Control Section Dictionary. These follow each of the associated control section names and are identified with asterisks after their addresses.

Qualification determines which storage map is produced through use of the $MAP system symbol. The default value when $VM is specified as qualification by the MSP is the current task, and when specified by a TSP it is the task to which he is connected.

The $MAP system symbol is valid as the operand of either the DUMP or DISPLAY command. In either case, the output format is the same for each element of the map, as described in Appendix C.

$IO System Symbol

The $IO system symbol may be used to specify a data field in secondary storage or to otherwise reference an I/O device. In Figure 3:

xxxx
: when prefixed by C represents a symbolic device address; when prefixed by X represents an actual device address (physical path).

decimal integer
: a device address which, when converted into hexadecimal, represents the physical path.

sp symbol
: represents a data field defined by the System Programmer that contains the actual or symbolic address of an I/O device, depending on the symbol type. (For example, a symbol with type character should be used if the symbolic address is required; a symbol with type not character, e.g., hexadecimal, should denote the actual address.)

mode set
: represents a literal or data field. It is used only when specifying a seven-track magnetic tape unit. The literal or the contents of the data field provide a byte of information that is used in construction of a mode set CCW. The extra commas shown in the format illustration with mode set do not represent optional parameters, but they may not be omitted.

cylinder, track, record
: specify the desired cylinder, track, and record on a direct access device. If any of these is defaulted, every item to the right is ignored. If "cylinder" is defaulted, the entire device is assumed.

o,l,t,s
: represents the data field attributes "offset," "length," "type," and "size," respectively. These may be defaulted according to the rules for immediate attribute designation. The default values are offset, 0; length, 1 byte; type, hexadecimal; size, same as length.

The following examples depict typical usage of the $IO system symbol:

$IO(X'0181',,,X'AB')

```
      ( C'xxxx'         ) [ (                                   ) ]
      ( X'xxxx'         ) [ ( ,,,[number of records],mode set   ) ]
 $IO( ( decimal integer ) [ (                                   ) ] )[.(o,l,t,s)]
      ( sp symbol       ) [ ( ,cylinder[,track[,record,]]       ) ]
```

Figure 3. Format Illustration of the $IO System Symbol

$IO(X'0190',4,4,15).(32,8)

In the first example a tape drive whose actual address is 181 is specified. The literal X'AB' specifies the bit configuration that is to be used for a mode set CCW command.

In the second example, a data field on a direct access device is specified. The actual device address is 190, and the data field address is cylinder 4, track 4, record 15 plus 32 bytes (offset). The data field length is eight bytes. The type and size attributes are defaulted as hexadecimal and eight, respectively.

Note: While an MSP may address any device in the system, a TSP may address and use only those devices allocated to his task.

$VAM System Symbol

The $VAM system symbol is similar to the $IO system symbol, except that it is used only to specify a direct access device that is formatted for a virtual access method (VAM). The format is:

```
        ┌C'xxxx'      ┐
$VAM(   │X'xxxx'      │ [,y])[.(o,l,t,s)]
        │decimal int. │
        └sp symbol    ┘
```

xxxx
: when prefixed by C represents the symbolic device address of a VAM-formatted direct access device; when prefixed by X it represents the actual device address (physical path) of a VAM-formatted direct access device.

decimal int.
: a device address which, when converted into hexadecimal, represents the physical path.

sp symbol
: represents a data field defined by the System Programmer that contains the actual or symbolic address of a VAM-formatted direct access device, depending on the symbol type.

y
: represents a relative page number on the specified device, if that device is a disk; a head and slot number of the device if it is a drum. If "y" is omitted, the entire device is assumed.

o,l,t,s
: represents the data field attributes "offset," "length," "type," and "size," respectively. These attributes may be defaulted according to the rules for immediate attribute designation. The default values are offset, 0; length, 1 page (4096 bytes); type, hexadecimal; size, same as length. Offset, length, and size are specified in bytes.

The following is an example of the $VAM system symbol specifying the direct access device (assumed to be VAM formatted) whose address is 190 and further specifying the data field comprising the first 256 bytes on relative page 4:

$VAM(X'0190',4).(,256)

$DOUT System Symbol

The $DOUT system symbol represents a two-byte data field whose contents are assumed to be the symbolic device address of the output device for each execution of the DUMP command. The value contained in $DOUT must be established by the System Programmer, normally using the SET command and an operand consisting of the $DOUT and $IO system symbols. An example of this appears as follows:

SET $DOUT=$IO(C'2A')

As the $IO system symbol is resolved internally, its value becomes the symbolic device address in hexadecimal format.

The MSP may specify any tape drive or printer as the output device for a dump. A TSP may use only those devices allocated to the task to which he is connected. When a valid device designation has been made with the $DOUT system symbol, each subsequent execution of a DUMP command issued by that System Programmer will automatically use that device for output.

Note: A TSP should obtain the value for $DOUT (the address of his dump output device) from the main operator.

$AT System Symbol

The $AT system symbol is used to reference ATs that have been implanted by the System Programmer; it may be used as the operand of a DISPLAY, DUMP, or REMOVE command. The format is:

```
$AT[.location]
```

location
: represents a symbol or a hexadecimal address (in L-notation) that corresponds to an address where an AT was implanted by the System Programmer. If "parameter" is omitted by the MSP, all of his current ATs are implied.

If "parameter" is omitted by a TSP, all of his private ATs in virtual storage and all of the global ATs he implanted are implied.

$AT represents a data field of variable length and type as described under "AT Command."

When the $AT system symbol is used with the REMOVE command, each specified AT and its associated dynamic statement are removed from the appropriate AT Table. The space in the AT Table may then be reused by the System Programmer.

When the $AT system symbol is used with the DUMP or DISPLAY command, the output is formatted as described in Appendix C.

$PATCH System Symbol

The $PATCH system symbol is used to reference patches that have been inserted by the System Programmer with the PATCH command. It may be used as the operand of a DISPLAY, DUMP, or REMOVE command. The format is:

```
┌──────────────────────────────┐
│      $PATCH[.location]       │
└──────────────────────────────┘
```

location
: represents a symbol or a hexadecimal address (in L-notation) that corresponds to an address where a patch was inserted by the System Programmer. If "parameter" is omitted by the MSP, all of his current patches are implied. If "parameter" is omitted by a TSP, all of his current patches are implied except for any implanted dynamically via ATs in real storage (see "REMOVE Command").

$PATCH represents a data field of variable length and type as described under "PATCH Command."

When the $PATCH system symbol is used with the REMOVE command, the original data is restored and the record of the patch (in the System Programmer's Patch Table) is deleted. The space in the Patch Table may be reused.

When the $PATCH system symbol is used with the DISPLAY or DUMP command, the output is formatted as described in Appendix C.

In RSS only, if the task associated with a particular patch is no longer available to the system (due perhaps to abnormal termination or LOGOFF), the format of a DUMP or DISPLAY of a patch is as described in Appendix C, except that the "patch data" field contains the character string "Task Unavailable".

$DHDR System Symbol

The $DHDR system symbol is used to label output of the DUMP command. The SET command is used to set the symbol to a character string which is used as a subheading on the next dump taken. The maximum permissible length of the header is 80 bytes of character information. For example:

```
SET $DHDR=C'SAMPLE OF TSSS DUMP'
```

$STATUS System Symbol

The $STATUS symbol may be used only as the operand of a DUMP command, and produces a formatted dump of the primary system status indicators, including all machine registers, PSWs, the TSI, and the XTSI header. The output data is formatted as indicated in Appendix C.

$STATUS is not implemented for the VSS subsystem.

$TASK System Symbol

The $TASK symbol may be used only as the operand of a DUMP command, and produces a dump of the primary task status indicators. This symbol is the virtual storage counterpart of $STATUS. The output data is formatted as shown in Appendix C.

In the VSS subsystem, $TASK produces a dump which is formatted the same as is that for RSS, except for the exclusion of the XTSI header. Task data is available only for the task to which VSS is connected. Accordingly, the $TASK operand is used alone with no subscript.

For use in the RSS subsystem, the symbol may be subscripted by a task ID, and if so subscripted, the data produced is for that task. For example:

```
DUMP $TASK(X'13')
```

If no subscript is used, the data produced is for the current task.

TIME SHARING SUPPORT SYSTEM COMMANDS

The TSSS commands, in the order in which they are described, are: QUALIFY, DEFINE, AT, DISPLAY, DUMP, COLLECT, SET, PATCH, REMOVE, IF, RUN, STOP, CONNECT, DISCONNECT, CALL, and END.

Each command description includes a definition of the command function, the command format, a detailed functional description, cautions (where applicable), programming notes, and examples of usage.

The generalized format for TSSS commands is:

Operation	Operand
command	[qualification.]operand[.(o,l,t,s)]

command
: one of the 16 TSSS operations.

qualification
: either $RM or $VM, which qualifies a storage address designated by "operand" as real or virtual storage (see "System Symbols Used for Qualification"). This optional parameter is not shown in the individual command format illustrations; its applicability can be determined in each case from the command's functional description.

operand
: the definition of each command operand and the acceptable means for specifying it are included with each command description.

o,l,t,s
: specification of offset, length, type, and size, respectively, which may be included with each designation of a data field in a command operand except for one instance with the DEFINE command. See "Immediate Attribute Designation" for use of these parameters in all operands except those occurring with the DEFINE command. Except for the DEFINE command, this option is not shown in the command format illustrations.

The command format includes one or more blanks between operation and operand. The COLLECT, SET, and PATCH commands, and one format of the DEFINE command, require two operands separated by an equal sign. For purposes of this discussion, however, the syntactical unit comprising these operands and the equal sign are considered a single compound operand. Each operand, simple or compound, usually may be followed by additional operands to save retyping of the operation itself. These multiple operands are separated by commas. Note that there are as many executions of the command as there are operands.

Some of the examples depict more than one command. Except for AT and IF, each command operand is separated from a succeeding command in the same statement by a semicolon.

TSSS will attempt to execute each command whenever possible. Incorrect specification of parameters is no guarantee that a command will be rejected. It is the System Programmer's responsibility to observe his progress closely to be certain he is obtaining the results he intends.

No prompting or confirmation messages are issued by TSSS. Error handling procedures are discussed in Part III, and the diagnostic messages written to a System Programmer's terminal are listed and explained in Appendix B.

QUALIFY Command

The QUALIFY command is used to establish uniqueness for subsequent command operands that reference real or virtual storage and to establish global qualification for a TSP. Every symbolic or actual storage address is qualified either implicitly or explicitly; the QUALIFY command is used to change the state of implicit qualification.

The format is:

Operation	Operand
QUALIFY	system symbol

system symbol
: the real memory, virtual memory, or global qualifier.

  Specified as: the $RM or $VM system symbol, together with an appropriate parameter if desired or required (see "System Symbols Used for Qualification"). $VM with no parameter specifies global qualification for a TSP or the virtual memory of the current task for the MSP.

Functional Description: A TSSS terminal session begins in a default qualification state -- real memory for the MSP, the virtual memory of the associated task for a TSP. Each time the QUALIFY command is executed, the qualification state (implicit qualification) becomes that which is specified in the command operand. There is no limit to the number of QUALIFY commands that may be issued during a terminal session.

When an AT is implanted with the AT command, the state of implicit qualification is assigned to the associated dynamic statement. Therefore, when the AT is executed the qualification is that which was current when the AT was implanted, regardless of changes made with the QUALIFY command after the AT was implanted. The state of qualification that was current just prior to execution of the AT is restored upon execution of an implied or actual RUN.

Cautions: When a TSP establishes global qualification to implant a global AT and create a global dynamic statement, any SP symbols in that statement must be globally defined (see "DEFINE Command" and "AT Command").

If an AT is implanted in real storage by a TSP, execution of the associated dynamic statement is performed by RSS, and it is treated as if the AT had been implanted by the MSP. Qualification must be handled accordingly.

When using the $RM symbol in VSS to SET or PATCH, a page of real storage containing the target location to be changed is obtained, without TSS/360 activity being suspended. The entire page is replaced after the change has been made. Caution should be used, therefore, since results to the system are unpredictable.

Programming Notes: The $RM and $VM system symbols may be used within the operands of commands other than QUALIFY to specify explicit qualification, overriding but not changing the state of implicit qualification.

The QUALIFY command may appear in a dynamic statement to specify implicit qualification for subsequent commands in that statement.

If a STOP or CALL command occurs in a dynamic statement, implicit qualification remains the same as that which was in effect for the STOP or CALL command. Subsequently, upon execution of an implied or actual RUN, qualification reverts to that which was current prior to execution of the AT which initiated execution of the dynamic statement.

Because execution of ATs appears asynchronous to the System Programmer, and because qualification can undergo one or more changes due to AT execution, the current state of qualification at a given time may be in doubt. Liberal use of the QUALIFY command can overcome this potential difficulty.

Example 1:

Assume that a TSP has begun a terminal session and that at some point he wishes to reference the TSS/360 resident supervisor. He issues:

```
QUALIFY $RM
```

Execution of this command establishes the implicit state of qualification as real memory; subsequent command operands that represent storage addresses are assumed to reference real storage unless they are SP symbols that represent data fields with virtual memory qualification. Any reference to data in a Prefixed Storage Area (PSA) is directed to the current PSA (primary or alternate) of the CPU executing on behalf of this TSP, since the $RM operand does not specify a CPU number.

Example 2:

```
QUALIFY $VM;DEFINE A;QUALIFY $VM(X'0D')
```

Execution of this statement, on behalf of the TSP connected to task 13 (decimal), establishes global qualification for execution of the DEFINE command, then establishes virtual memory 13 as the implicit qualification state.

DEFINE Command

The DEFINE command enables a System Programmer to create temporary symbols, either private or global, and to specify their attributes. These symbols, called SP symbols, may be the names of data fields in the System Programmer's working storage or they may be associated with TSS/360 data fields (real, virtual, or secondary storage). The format differs accordingly and is designated "format 1" or "format 2."

The distinction between private and global SP symbols is relevant only for a TSP; it specifies whether a private or global (shared) SP symbol table records the symbol definition. All of the MSP's SP symbols are recorded in the same SP symbol table.

Format 1: When an SP symbol is to be defined as the name of a data field in working storage (storage space is to be allocated by TSSS when the command is executed), the format of the DEFINE command is:

Operation	Operand
DEFINE	symbol[.(o,l,t,s)][,...]

symbol
the data field name.

Specified as: a character string of one to eight alphameric characters, the first of which must be alphabetic.

o,l,t,s
offset (always zero) plus length, type, and size attributes, respectively.

Specified as: decimal integers, normally, for length and size, although any expression that reduces to an appropriate value may be used. Offset is ignored by TSSS and normally is defaulted by the System Programmer. Type is coded I, X, or C to indicate decimal integer, hexadecimal, or character, respectively. Any of these attributes may be defaulted, but all commas to the left of each attribute specified (including the comma following the "o") must be retained to identify the attributes by position.

TSSS default: offset, 0; length, 1 byte; type, hexadecimal; size, same as length.

Format 1 Functional Description: Except for a TSP's global qualification, the state of qualification when the format 1 DEFINE command is executed is irrelevant, because both the symbol definition and allocated storage exist in the System Programmer's working storage and will be correctly referenced when the symbol is used in subsequent command operands.

SP symbols defined by a TSP under global qualification are global symbols. The symbol definition and, if format 1, allocated storage exist in shared storage, and the symbol will be resolved only when used in a statement under global qualification. Any TSP with global qualification can reference any other TSP's global symbols.

Format 2: When the SP symbol being defined is to be a pseudonym (or "alias") for an existing symbol or the temporary name of a TSS/360 data field or I/O device (no storage is to be allocated), the format of the DEFINE command is:

Operation	Operand
DEFINE	symbol= { external symbol / sp symbol / system symbol / address } [.(o,l,t,s)][,...]

symbol
the data field or device name. If attributes and/or explicit qualification are designated they are ignored.

Specified as: a character string of one to eight alphameric characters, the first of which must be alphabetic.

external symbol
a TSS/360 external symbol, which in turn represents an address in real or virtual storage, depending on qualification.

Specified as: the character string as assembled for the TSS/360 that is currently running.

sp symbol
a previously defined SP symbol, private or global depending on the current qualification state.

Specified as: a character string identical to the previously defined SP symbol.

system symbol
any TSSS system symbol that names a data field or designates an I/O device. The excluded system symbols are $RM, $VM, $ID, $MAP, $AT, and $PATCH. Permissible qualification is defined for each system symbol with its description (see "System Symbols").

Specified as: a character string.

address
an actual storage address, real or virtual.

Specified as: a hexadecimal value in L-notation (e.g., L'012345').

o,l,t,s
an offset specification plus length, type, and size attributes, respectively.

Specified as: decimal digits, normally, for offset, length, and size, although any expression that reduces to an appropriate value may be used; type is coded I, X, or C to indicate decimal integer, hexadecimal, or character, respectively. Rules for defaulting are described for format 1.

TSS default: offset, 0; length, 1 byte; type, hexadecimal; size, same as length.

Format 2 Functional Description: When the DEFINE command is used with format 2, the base address of the data field named "symbol" becomes the effective address of the parameter on the right of the equal sign plus the specified offset.

If format 2 of the DEFINE command is used under global qualification, the parameter on the right of the equal sign must be "logically global"; i.e., it must be an external symbol that is shared by all tasks, a global SP symbol, or a system symbol as defined above but with $IO, $VAM, and $DOUT excluded as well.

When format 2 of the DEFINE command is used other than with global qualification, the state of qualification when the command is executed is irrelevant if the parameter on the right of the equal sign is an SP symbol or a system symbol that designates an I/O device.

General Functional Description: Each SP symbol created with the DEFINE command is represented by a 48-byte entry in an SP symbol table. This table is built dynamically with successive executions of the DEFINE command, using a block of storage that is also used to allocate storage when format 1 is used, as shown in Figure 4. When execution of a DEFINE command would overflow the appropriate storage block, the command is rejected.

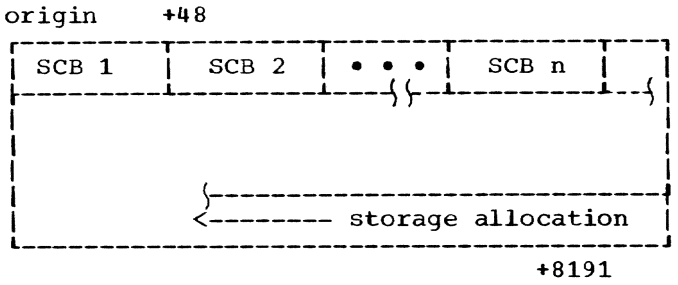


Figure 4. SP Symbol Table and Working Storage Block

Figure 4 depicts the format of a storage block used to build an SP symbol table and to allocate working storage when the format 1 DEFINE command is executed. "SCB" stands for symbol control block, each of which constitutes an entry in the symbol table. Note that the first SCB built via execution of a format 1 DEFINE command points to the last (high-order) data field in the storage block.

The total space reserved for an SP symbol table and for format 1 storage allocation is two pages (8192 bytes). There is one such SP symbol table and storage block for an MSP, in which all his SP symbols reside, and one for each connected TSP. There is one Global Symbol Table and storage block that is shared by all connected TSPs when defining global SP symbols and is shared by all tasks for resolution of global SP symbols used in global dynamic statements.

Cautions: The DEFINE command should not be used in global dynamic statements. If it is, the repeated attempts to redefine the symbol result eventually in an error condition.

If the SP symbol being defined by the MSP ("symbol" in both the format 1 and format 2 illustrations) already exists in the RSS SP Symbol Table, the symbol is redefined, the original attributes are lost, and any storage allocated for the previous symbol definition cannot be referenced with the symbol.

If the SP symbol being defined by a TSP ("symbol" in both the format 1 and format 2 illustrations) already exists in his private SP Symbol Table and qualification is non-global, the symbol is redefined, the original attributes are lost, and any storage allocated for the previous symbol definition cannot be referenced with the symbol. The symbol may exist independently in the Global Symbol Table, however.

A symbol may exist independently in a TSP's private SP Symbol Table, but if it already exists in the Global Symbol Table the symbol is redefined, the original attributes are lost, and any storage allocated for the previous global symbol definition cannot be referenced with the symbol.

Programming Notes: The attributes assigned to an SP symbol through use of the DEFINE command may be overridden for a single use of the symbol with immediate attribute designation, as described under "Data Field Defining Elements."

The attributes of SP symbols may be referenced with the $B, $P, $L, $T, and $S system symbols, as described in the "System Symbols" section.

The pointer attribute of an SP symbol is always zero at the time the symbol is defined with the DEFINE command.

Example 1:

```
DEFINE A
```

Execution of this format 1 command causes one byte of storage (the default length and size attributes) to be allocated and assigned the name A. The contents of the data field A are unknown, and the type attribute has been defaulted as hexadecimal.

Example 2:

```
DEFINE MYSYMB=EXTSYMB.(16,4)
```

Execution of this format 2 command assigns the temporary name, or pseudonym, MYSYMB to the TSS/360 data field beginning 16 bytes beyond the address represented by the external symbol EXTSYMB. The resulting data field has a length attribute of four bytes and the default type and size attributes of hexadecimal and four, respectively. EXTSYMB is assumed to exist in real storage, shared virtual storage, or a task's private virtual storage, depending on the implicit qualification when this DEFINE command is executed. Note that with format 2 there is no allocation of storage.

Example 3:

```
DEFINE HEADER.(,8,C),AREA.(,8,,512),
MYSYMB=EXTSYMB.(16,4)
```

Multiple operands, as shown here, may include both format 1 and format 2 in the same statement. Execution of this command allocates eight bytes of storage as the data field named HEADER and assigns it a type attribute of character; it then allocates 512 bytes (size) of storage as the data field named AREA and assigns it a length attribute of eight bytes and a type attribute of hexadecimal. AREA thus constitutes a 512-byte array comprising 64 eight-byte elements and can meaningfully be used with subscripts. The contents of HEADER and AREA are unknown. Execution of the command finally would process the third operand as described under Example 2.

AT Command

The AT command is one of the most basic tools in the System Programmer's kit. It provides the means for specifying an instruction location in running code where, when that location is reached during TSS/360 execution, the execution of a TSSS command statement is triggered. That statement, called a dynamic statement, follows the AT command in the TSSS input stream and itself is followed by end-of-block (EOB); it is then stored to await execution when the specified instruction location is reached.

The format of the AT command is:

Operation	Operand
AT	address[,...]

address
a real storage or virtual storage location, assumed to contain an executable instruction.

Specified as: a symbol, or a hexadecimal address in L-notation.

Functional Description: The AT command must be followed by at least one other TSSS command; any TSSS statement is accepted without being checked for validity. The operand of an AT command must be followed by one or more blanks; unlike other TSSS commands except IF, the AT command is not separated from the following command by a semicolon.

If more than one operand is included with an AT command, the associated dynamic statement will be executed when each of the specified instruction locations is reached. Conversely, if the operands of two or more AT commands issued by this System Programmer specify the same instruction location, the ATs are "chained," which results in execution of the dynamic statements in the order in which they were received as input.

Execution of the AT command results in an SVC instruction being implanted at the specified instruction location. (For convenience, this SVC, its associated control block, and the processing it initiates are collectively referred to as an AT.) The overlaid instruction is saved and the remainder of the input statement is stored; the input device is then read again.

When an implanted AT is executed, the stored dynamic statement is treated as fresh input and is executed, after which control reverts to TSS/360 with execution of the original instruction that was saved when the AT was implanted. In other words, a dynamic statement ends with an implied (or explicit) RUN. The exceptions are the occurrence of a STOP, CALL, or DISCONNECT command in the dynamic statement.

A TSP's AT in shared virtual storage that is globally qualified or any MSP-implanted AT in shared virtual storage is called a global AT. Other ATs are called private ATs; if they reside in virtual storage, they are executed only by the designated task.

The number of ATs that may be specified by the MSP during a given terminal session is governed by his AT Table, which is one page (4096 bytes) long. Each AT requires 28 bytes for control purposes plus as many bytes as there are characters in the dynamic statement being stored. When overflow of the AT Table occurs with execution of an AT command, that command is rejected.

The above is also true for a TSP, except that only his private ATs are recorded in his AT Table.

The number of global ATs that can exist for all TSPs at a given time is governed in the same manner by the Global AT Table, a shared table in virtual storage. This

table is also one page long, but it is used by all connected TSPs who wish to use it and is referenced by each task that executes a global AT in shared virtual storage.

It should be noted that a TSP issuing an AT command with real memory qualification invokes RSS for implanting the AT and again each time the AT is executed; the MSP's AT Table is used to record the AT, whether or not the MSP is connected. The TSP is said to be operating in the MSP domain. Terminal output is to the MSP terminal, which is always the Operator's terminal. Commands within the dynamic statement must meet the requirements for any commands issued by the MSP (e.g., SP symbols must be defined in the RSS SP Symbol Table).

When a TSP implants an AT in real storage, the maximum length of the input statement is 238 characters.

Cautions: TSSS will not accept any AT for a destination instruction location containing any of the following:

- An Execute or Diagnose instruction.
- An SVC instruction, unless the SVC is an AT for which chaining is permitted.
- A privileged instruction in virtual storage.

An AT should not be implanted where the subject instruction of an Execute instruction is located; execution of the AT SVC would appear to be spurious. A TSSS error condition would be recognized, followed by resumption of TSS/360 processing without execution of the overlaid instruction.

In RSS only, care should be taken when an AT is implanted in a private IVM page that has not been paged into main storage and has never been changed by the task. The AT SVC is placed in the permanent copy of the page and is picked up by tasks that refer to the page and have no record of the AT. This results in a system logic error.

An AT cannot be implanted in any location where a private AT already exists for another System Programmer. Note, however, that an AT specified by a TSP for a real storage location is treated as if it had been specified by the MSP. Consequently, the presence of an MSP AT at the specified location would result in chaining in this case.

An SP should employ caution when implanting an AT in:

1. a module which executes with interrupts disabled, or
2. a TSS/360 module which is called by RSS or VSS or upon which RSS or VSS is dependent.
3. (RSS only) a program that was loaded from a JOBLIB.

The TSS/360 modules referred to in 2 include:

RSS	VSS
CEAAF	Task Monitor
CEAL1	Dynamic Loader and its subroutines
CEAIC	
CIP	
SERR-Bootstrap	
Recovery Nucleus	
Pathfinding Routines	

When a TSP implants an AT in a module that has been loaded by his task, it is his responsibility to remove the AT (with a REMOVE command) prior to unloading the module; failure to do so prevents later removal of the AT and results in a virtual storage system error when the task terminates as a result of LOGOFF or an abnormal termination (ABEND).

Programming Notes: For a TSP, an AT in shared virtual storage can be either global or private. It is his responsibility to ensure that the operand of an AT command issued with global qualification in fact specifies a shared code location, and to specify a dynamic statement that can be executed successfully by all tasks using the shared code.

If execution of the dynamic statement associated with a TSP's global AT results in an error condition that normally produces a diagnostic message, the attempt to write the message is suppressed if the current task has no TSP connected. In most cases, but not always, the error-producing operation is not performed (see Appendix B, "Messages"). In the event an I/O operation was requested (e.g., a DUMP or DISPLAY command), only when an allocated and specified device is available to the executing task will the I/O be performed.

If an AT is implanted in real storage by a TSP, the AT may be executed without prior issuance of a RUN command, since the TSS/360 resident supervisor continues to execute on behalf of all tasks. Note, however, that if such an AT causes a STOP command to be executed, it occurs in the MSP domain. A $ is written at the MSP terminal and RSS is in control until a RUN or DISCONNECT command is issued by the MSP.

An AT can be referenced through use of the $AT system symbol and an appropriate command, such as the REMOVE command.

Except for a TSP's ATs in real storage, all of a System Programmer's ATs are removed automatically when he issues a DISCONNECT command. A TSP should remove any ATs he has implanted in real storage prior to issuing a DISCONNECT (see "REMOVE Command").

Example 1:

```
AT INTPROC COLLECT COLLAREA=$IPSW
```

When this AT command is processed the remainder of the statement is saved for subsequent execution. The character $ is then written at the terminal to invite more input (or the next statement is read if in call mode). When the symbolic instruction location INTPROC is reached (which is after a RUN command has been issued if INTPROC is in virtual storage), the COLLECT command is executed. Upon completion of that operation, the instruction originally located at INTPROC is executed and normal processing resumes. If INTPROC is in shared virtual code and either the user is the MSP or the statement was issued by a TSP under global qualification, the COLLECT command is executed each time a CPU fetches the instruction at INTPROC, until the AT is removed. The same is true if INTPROC is in real storage. However, if INTPROC is in shared virtual storage but the TSP has specified non-global qualification, only the TSP's task executes the COLLECT command.

Example 2:

```
AT INTPROC COLLECT COLLAREA=$IPSW;STOP
```

This statement is processed as is that in Example 1 until the COLLECT command execution is completed. Then the STOP command is executed, which results in a $ being written at the terminal to invite input; the original instruction at INTPROC would not be executed at this time.

DISPLAY Command

The DISPLAY command is used to write the contents of data fields on the System Programmer's terminal. The format is:

Operation	Operand
DISPLAY	{data field | literal}[,...]

data field
the location of data residing within the system; this data can be the result of an arithmetic or Boolean operation.

Specified as: any representation of a data field, or an arithmetic or Boolean expression.

literal
immediate data in the input stream.

Specified as: a character, hexadecimal, or decimal integer literal.

Functional Description: The number of bytes to be displayed is determined by the length attribute (not size) of the data field in storage or of the length of the data protion of the literal. Immediate attribute designation can be used to assign a temporary length attribute to a data field in real or virtual storage.

If the data field is in secondary storage, the type is assumed to be hexadecimal. If an offset is specified in this case, the block is truncated on the left by the amount of the offset. An entire track, cylinder, or device can be specified, as defined under "$IO System Symbol." As defined under $VAM System Symbol, an entire page or device can be specified.

The format of the terminal printout is described in Appendix C. If multiple operands are used with the DISPLAY command, the data specified by each is written beginning on a new line.

Note: When a system symbol representing a data field is used with the DISPLAY command, the system programmer's default qualification is used, overriding any further qualification.

Programming Notes: The DISPLAY command normally is used for short data fields, especially when executed by RSS. If extensive data fields are to be written, the DUMP command normally is used.

If the System Programmer presses the Attention key while the DISPLAY command is being executed, the display is canceled, and a message is written to the terminal. The terminal is then ready to receive input; the remainder of the current statement, if any, is ignored. If the DISPLAY command was executed from a predefined statement set (call mode), reading of the card reader or tape drive ceases and the terminal becomes the input device again.

Example 1:

```
DISPLAY C'IDENTIFIER',A:B
```

Execution of this command results in the printing of the character string IDENTIFIER on the System Programmer's terminal, followed by, on a new line, the contents of the data field beginning at the base-plus-

pointer address of A and ending with the last byte specified by the length attribute of B.

Example 2:

```
DISPLAY L'1234'.(,4),$PSW
```

Execution of this command causes the contents of a four-byte data field whose storage address is 1234 (hexadecimal) to be printed on the System Programmer's terminal, followed by the PSW that was current when TSSS received control (on a new line).

DUMP Command

The DUMP command is used to write the contents of specified data fields on a previously specified output device. The format is:

Operation	Operand
DUMP	{data field / literal} [,...]

data field
the location of data residing within the system; this data can be the result of an arithmetic or Boolean operation.

Specified as: any representation of a data field, or an arithmetic or Boolean expression.

literal
immediate data in the input stream.

Specified as: a character, hexadecimal, or decimal integer literal.

Functional Description: The number of bytes to be written on the output device is determined by the length attribute (not size) of the data field in storage or of the length of the data portion of the literal. Immediate attribute designation can be used to assign a temporary length attribute to a data field in real or virtual storage.

If the data field is in secondary storage, the type is assumed to be hexadecimal. If an offset is specified in this case, the block is truncated on the left by the amount of the offset. An entire track, cylinder, or device can be specified, as defined under "$IO System Symbol" and "$VAM System Symbol."

When the DUMP command is executed, the address of the output device must be in the data field $DOUT, and the device must be a printer or a tape drive. (For a TSP, the device must be currently allocated to the task to which he is connected.)

If a dump is to tape, the data is written exactly the same as it would be to the printer -- i.e., in EBCDIC characters, 132 characters to a print line including a forms-motion byte -- followed by a tape mark. No labels are written automatically. The SET command can be used to print the tape.

The output format of a TSSS dump, and the procedure for printing a TSSS dump tape, are described in Appendix C. If multiple operands are used with the DUMP command, the data specified by each is written beginning on a new page.

Note: When a system symbol representing a data field is used with the DUMP command, the system programmer's default qualification is used, overriding any further qualification.

Programming Note: Pressing the Attention key (causing an asynchronous interruption) while the DUMP command is being executed has the same result as is described under "DISPLAY Command."

Examples: The following examples would be equally valid if the DISPLAY command appeared in place of DUMP. In the same fashion, the examples under "DISPLAY Command" would be valid here by substituting DUMP for DISPLAY.

Example 1:

```
DUMP COLLAREA
```

Assuming COLLAREA is an SP symbol representing a data field (as defined for the examples under "COLLECT Command"), execution of this command causes 512 bytes (the length attribute) of data beginning at the symbolic address COLLAREA incremented by the value of the data field's pointer to be written on the output device specified in the $DOUT data field. Note, however, that if COLLAREA is being used as a collection area as described under "COLLECT Command," the value of the pointer may be unknown. The $P system symbol can be used to display and to manipulate the value of pointer.

Example 2:

```
DUMP $B(COLLAREA)%.(,512)
```

When this DUMP command is executed the indirect addressing operator (%) causes the data field represented by $B(COLLAREA) to be used as the starting address of a 512-byte dump. As the indirect addressing operator is used, the defined length of COLLAREA is lost; the 512 must be speci-

fied. As $B(COLLAREA) resolves to the base address alone of COLLAREA, the problem presented in Example 1 (unknown pointer value) is overcome.

Example 3:

```
DUMP $IO(X'0191',2,11,38).(8)
```

Execution of this command results in record 38 on track 11 of cylinder 2, direct access device whose actual address is X'191', minus the first 8 bytes, being written in hexadecimal format on the output device specified in $DOUT.

COLLECT Command

The COLLECT command is used to gather data from one data field into another data field. When more than one exectuion of a COLLECT command causes data to be moved into the same destination data field, the command will cause the data to be placed in successive elements of the field. The format is:

Operation	Operand
COLLECT	sp symbol={data field / literal}[,...]

sp symbol
: a data field defined to serve as a collection area.

  Specified as: a character string.

data field
: the location of data to be collected.

  Specified as: any representation of a data field.

literal
: immediate data in the input stream.

  Specified as: a character, hexadecimal, or decimal integer literal.

Functional Description: The first parameter (SP symbol) designates the destination data field, and the second parameter specifies the data to be moved into the destination data field. When the COLLECT command is executed, the data is moved in accordance with the length attribute of the second parameter, and then the pointer attribute of the first parameter is incremented by the value of the length attribute of the second parameter.

Note that the destination data field address is always base plus pointer, thereby causing successive elements of the field to be filled with each execution of every COLLECT command that specifies the same SP symbol as the first parameter. The size attribute defines the extent of the data field.

When the value of the pointer attribute of the first parameter is equal to the size attribute, or is large enough that the length of the second parameter is greater than the remaining space in the data field (size minus pointer), the pointer is reset to zero before the movement of data takes place. The data field is then used as before each time it is specified as the first parameter of any succeeding COLLECT command operands.

A single COLLECT command can appear in a dynamic statement that will be executed repetitively (depending on the location of the AT), in which case each execution will move the same amount of data into the collection area. On the other hand, more than one COLLECT command can move data into a common collection area. In the latter case, the fields of the filled-in collection area may be of variable length.

A data field being collected with a single execution of the COLLECT command cannot be more than 4096 bytes long.

Caution: Truncation occurs if the length attribute of the second parameter in a COLLECT operand exceeds the size attribute of the first parameter; type is set to character and the data is truncated on the right.

Programming Notes: A data field (such as COLLAREA in the examples below) cannot be referenced symbolically in the usual way, after it has been used as a collection area, with any assurance that the value of pointer is zero. The SET command and the $P system symbol can be used to reset the value of pointer.

To determine which byte of a collection area was the last one used by an execution of the COLLECT command, the System Programmer can display the pointer value with the DISPLAY command and the $P system symbol. Whether the pointer has cycled through the collection area one or more times will not be known unless a conditional statement is used to predefine that aspect of the operation, or unless inspection of the collection area reveals that information.

The COLLECT command is useful in a global dynamic statement. A TSP must globally define the SP symbol that represents the collection area if the COLLECT command is to be used globally.

Examples: It is the System Programmer's responsibility to define a collection area properly and to manage the space with the COLLECT command so that the collected data will be meaningful. Figure 5 depicts an example of a collection area named COLLAREA.

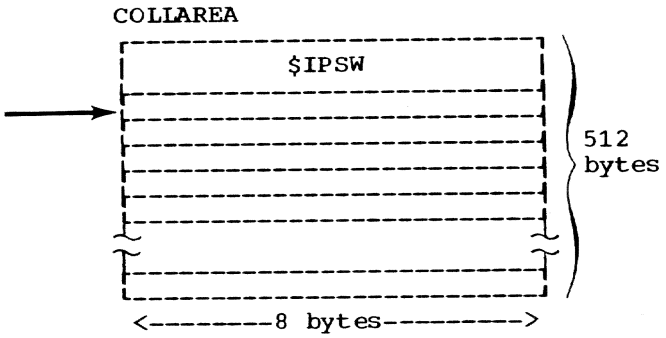


Figure 5. Collection Area

COLLAREA as shown in Figure 5 is an SP symbol that was defined as having a length attribute of eight bytes and a size attribute of 512 bytes. If COLLAREA was used to collect IPSWs, as shown below in Example 1, eight-byte elements will be filled successively with each execution of the COLLECT command. Note that Example 1 is the same as Example 1 under "AT Command."

Example 1:

```
AT INTPROC COLLECT COLLAREA=$IPSW
```

Each time the instruction location INTPROC is reached, this COLLECT command is executed and results in the I/O old VPSW being stored successively in eight-byte fields of the data field COLLAREA (assuming virtual memory qualification). The arrow in Figure 5 points to the effective address (base plus pointer) of COLLAREA after one execution of the COLLECT command.

Example 2:

```
AT NEWADDR COLLECT COLLAREA=C'ASYNC',
COLLAREA=$APSW
```

With this example, each time the instruction location NEWADDR is reached this COLLECT is executed. In this case, the first operand specifies that a literal is to be placed in COLLAREA at the current base-plus-pointer position. The pointer is then incremented by five (ASYNC is five bytes long) and the asynchronous I/O old VPSW is moved to COLLAREA at the new base-plus-pointer position. Note that a total of 13 bytes is moved to the data field COLLAREA with this execution of a COLLECT command.

SET Command

The SET command is used to insert data into a data field. The format is:

Operation	Operand
SET	data field$_1$ = {data field$_2$ | literal} [,...]

data field$_1$
the location where data is to be inserted; it can specify an I/O device.

Specified as: any representation of a data field, including the $IO or $VAM system symbol.

data field$_2$
the location from which data is to be moved; it can specify an I/O device. The data can be the result of an arithmetic or Boolean operation.

Specified as: any representation of a data field, including the $IO or $VAM system symbol, or an arithmetic or Boolean expression.

literal
immediate data in the input stream to be placed in data field$_1$.

Specified as: a character, hexadecimal, or decimal integer literal.

Functional Description: The SET command can be used to alter any data field, up to 4096 bytes long, in real, virtual, or secondary storage. No record of the operation is kept by TSSS.

Note: When a system symbol representing a data field is used with the SET command, the system programmer's default qualification is used, overriding any further qualification.

Cautions: The lengths of both parameters are used for the SET operation. If they are not the same, the data specified by the second parameter is padded or truncated, according to data type, as shown in Figure 6.

When using the $RM symbol in VSS to SET or PATCH, a page of real storage containing the target location to be changed is obtained, without TSS/360 activity being suspended. The entire page is replaced after the change has been made. Caution should be used, therefore, since results to the system are unpredictable.

Data Type			
Parameter 1	Parameter 2	Parameter 2 Greater	Parameter 2 Less (SET Command Only)
Character	Any	Parameter 2 truncated on right	Parameter 2 padded on right with blanks
Integer	Integer	Parameter 2 truncated on left	Parameter 2 padded on left by propagating leftmost bit
Integer	Noninteger	Parameter 2 truncated on left	Parameter 2 padded on left with zeros
Hexadecimal	Any	Parameter 2 truncated on left	Parameter 2 padded on left with zeros

Figure 6. Truncation and padding by SET command and truncation by COLLECT command

Programming Notes: When both parameters in a SET command operand specify I/O devices with the $IO system symbol, and if the second parameter designates a 2540 Card Reader or a tape drive, that device is read repeatedly until end-of-file is reached. A corresponding write operation is performed for each read (with no blocking and no data formatting). This makes possible a card-to-tape operation for creating a tape file comprising a TSSS predefined statement set (see "CALL Command") or a tape-to-printer operation for printing a TSSS dump that was written on tape. When the SET command is used to write to a printer, the record length is 101 bytes, including the forms-motion byte.

When the SET command is used to write to a tape drive, no tape mark is written. If the operation's purpose is to create a TSSS predefined statement set, the last command included in that statement set must be END, RUN, or DISCONNECT.

When the first parameter in the SET command operand specifies a direct access device (using either $IO or $VAM), alteration will not occur past the end of the specified record. If the length specified for the first parameter would cause alteration past the end of the record, the statement will be rejected.

Example 1:

```
DEFINE CTR;SET CTR=X'00';AT LOC SET
CTR=CTR+X'01'
```

In this statement, the System Programmer defines a one-byte field named CTR (an SP symbol), sets the value of CTR to zero, and then creates a dynamic statement that will increment CTR by one each time the instruction location LOC is reached.

Example 2:

```
SET FLAG=FLAG|X'40'
```

With this command, the System Programmer sets a bit in a replica of the data field FLAG with the OR operator, then overlays FLAG with the replica. (Bit manipulation with Boolean operators is performed in working storage.)

PATCH Command

The PATCH command is used to insert data into a data field and concurrently to save the original data and build a block of control information. The format is:

Operation	Operand
PATCH	data field$_1$={data field$_2$ / literal} [,...]

data field$_1$
the location where data is to be inserted; it can specify an I/O device.

Specified as: any representation of a data field, including the $IO or $VAM system symbol.

data field$_2$
the location from which data is to be moved; it can specify an I/O device. The data can be the result of an arithmetic or Boolean operation.

Specified as: any representation of a data field, including the $IO or $VAM system symbol, or an arithmetic or Boolean expression.

literal
immediate data in the input stream to be placed in data field$_1$.

Specified as: a character, hexadecimal, or decimal integer literal.

Functional Description: When a patch is placed in a real or virtual storage location, no other copy of the affected module is automatically changed. The System Programmer may reference the appropriate I/O device with another PATCH (or a SET) command, however, if he wishes to make such a change.

Each System Programmer has in his working storage a two-page Patch Table (8192 bytes) that is the limiting factor in determining how many patches may be inserted during a given terminal session. Each execution of the PATCH command produces a 24-byte Patch Control Block in the Patch Table and stores the original data, from the location that was patched, in the Patch Table. When a PATCH command is issued that overflows the Patch Table, the command is rejected with a message to the System Programmer's terminal.

The length of a patch (and thus of the original data being stored in the Patch Table) is determined by the length attribute of the parameter on the right of the equal sign; it cannot be greater than 4096 bytes.

Note: When a system symbol representing a data field is used with the PATCH command, the system programmer's default qualification is used, overriding any further qualification.

Cautions: A patch cannot be inserted where another patch has been inserted with the same starting address by the same System Programmer. If partial overlapping of patches is performed (starting addresses at least one byte apart), the patches can be removed without loss of any data only by removing them in the reverse order of their insertion.

The PATCH command is not appropriate in a global dynamic statement.

Programming Note: A patch can be referenced through use of the $PATCH system symbol with an appropriate command. If the patch is deleted with the REMOVE command, the space used by that patch in the Patch Table is freed for reuse.

Example 1:

```
PATCH MASK=X'F0'
```

When this command is executed, one byte of data at the data field named MASK is overlaid with the value X'F0', and the value previously contained in that byte is saved in the Patch Table with 24 bytes of control information.

Caution: When using the $RM symbol in VSS to SET or PATCH, a page of real storage containing the target location to be changed is obtained, without TSS/360 activity being suspended. The entire page is replaced after the change has been made. Caution should be used, therefore, since results to the system are unpredictable.

Example 2:

```
PATCH $VAM(X'0190',14).(32)=X'F0'
```

When this command is executed, byte 32 of relative page 14 on the VAM-formatted device, whose actual address is X'190', is overlaid with the value X'F0' (in main storage). The original data is saved in the Patch Table with 24 bytes of control information, and the altered page is written onto its original external location.

REMOVE Command

The REMOVE command is used to delete one or more ATs and the associated information in the AT Table, or to restore data that was altered with the PATCH command and delete the associated Patch Table entry. The format is:

Operation	Operand
REMOVE	{$AT / $PATCH}[.location][,...]

location
the symbolic or actual address of a real, virtual, or secondary storage location.

Specified as: a symbol or L-notation.

RSS default: all ATs or patches implanted by the MSP, plus (1) if $AT, all real-storage ATs implanted by all connected TSPs; (2) if $PATCH, all patches implanted by TSPs via dynamic statements associated with real-storage ATs.

VSS default: (1) if $AT, all virtual-storage ATs, private and global, implanted by this TSP; (2) if $PATCH, all patches implanted by this TSP except any implanted via dynamic statements associated with real-storage ATs.

Functional Description: When the REMOVE command is executed, the original data at the specified location is restored (an instruction in the case of $AT, or the overlaid data in the case of $PATCH). The space in the AT Table or Patch Table that

corresponds to each operand is freed and may be reused.

Cautions: Each operand must correspond to an AT or a patch that has not been removed, although it need not be specified in the same way as when the AT or patch was implanted.

If a TSP implants ATs in real storage (the MSP domain), he should use the REMOVE command with a separate operand ($AT with a location specified) for each such AT prior to issuing a DISCONNECT command.

An AT implanted in real storage by a TSP can be referenced with the REMOVE command of VSS only if the $AT system symbol is used to specify each AT. If the AT location is defaulted, an error occurs.

Programming Note: The REMOVE command is implied for outstanding ATs implanted by this System Programmer when a DISCONNECT command is issued. Patches are not automatically removed when a DISCONNECT is issued, but the internal record of original data and control information is lost.

Example 1:

```
REMOVE $AT.INTPROC
```

When this command is executed, the instruction at location INTPROC is restored to its original state, and the associated dynamic statement and control information are deleted from the appropriate AT Table.

Example 2:

```
REMOVE $PATCH.$VAM(X'0190',14).(32)
```

When this command is executed, the specified patch (from Example 2 under "PATCH Command") is overlaid by the original data from byte 32 of relative page 14 on the VAM-formatted device, whose address is X'190'. Corresponding entries in the Patch Table are removed.

When the REMOVE command is used with $PATCH in RSS and the task associated with a particular patch is not available to the system (due, for example, to abnormal termination or LOGOFF), the REMOVE command removes the PCB with no replacement of the original data.

Note: When removing an AT or PATCH, qualification must be the same as at the time of implantation. If the qualification of any AT or PATCH is unknown, it can be determined by displaying the $AT or $PATCH system symbols.

IF Command

The IF command is used to designate that the remainder of the input statement is conditional; that is, the operand of the IF command must be evaluated as "true" if the remainder of the statement is to be executed. An IF command may appear within an input statement as well as at the beginning. The format is:

Operation	Operand
IF	expression

expression
: a logical expression that reduces to a single "true" or "false" value; it may contain arithmetic, relational, or Boolean operators or a combination of these.

  Specified as: a character string containing data field defining elements and/or literals, plus, normally, one or more operators.

Functional Description: When more than one IF command appears in a statement, performance of the evaluation of each is conditional on the result from the preceding one.

The IF command, like the AT command, is not separated from the following command by a semicolon. If the last character of the operand is alphameric, at least one blank must separate the operand from the following command.

Programming Notes: When an expression is evaluated as true, a field in working storage is set to X'FF'; when false, to X'00'. If there is no relational operator (< or = or >) in the operand of an IF command, the first byte of the specified field is inspected for the value "true." For example, "IF A DISPLAY B" would require the first byte of A to be all ones for the DISPLAY to be executed. Likewise, "IF A&B DISPLAY C" would require that the AND operation performed on the contents of A and B produces a field whose first byte is all ones for the DISPLAY to be executed.

When the operand of an IF command in an unchained dynamic statement is evaluated as "false," the IF command is followed by an implied RUN. If the dynamic statement is chained, however, the "false" value causes the next dynamic statement in the chain to be executed. When the operand of an IF command in an immediate statement proves to be "false," the input device is then read again (a $ is first written if not in call mode).

Example 1:

```
IF LOCKBYTE=X'FF' DUMP TABLE
```

Execution of this statement causes the contents of the data field named LOCKBYTE to be compared to the literal X'FF' and, if equal, the DUMP command to be executed. If the comparison results in an inequality, the DUMP command is ignored.

Example 2:

```
AT LOC IF CTR=X'40' DUMP COLLAREA;IF
¬(PTR=A'TABLE') STOP
```

In this example, the first IF command begins a dynamic statement that is stored until TSS/360 execution reaches the instruction at LOC. Then the contents of the data field CTR are compared to the literal X'40', and if not equal the remainder of the statement is ignored and control is returned to TSS/360. An equality results in execution of the DUMP command followed by a comparison of the contents of the data field PTR with the address of TABLE. In this case the NOT operator makes equality the "false" condition, and the STOP command would be ignored; any value of PTR other than the address of TABLE is "true," and the STOP command would be executed.

Example 3:

```
IF FIELD=X'FFFFFFFF'|FIELD=X'00000000'
DISPLAY FIELD;RUN
```

In this example, the OR operator specifies that satisfying either of two possible "true" conditions results in execution of the DISPLAY and RUN commands. If FIELD contains any value besides the two specified, the DISPLAY and RUN commands are ignored and a $ is written at the terminal to invite another input statement.

Example 4:

```
IF (FLAG1|X'FB')=X'FF'|(FLAG2|X'7F')
=X'FF' DUMP STATUS
```

In this example, the OR operation is performed on the contents of FLAG1 and the literal X'FB' and on the contents of FLAG2 and the literal X'7F' in a work area, and the results in each case are compared with the literal X'FF'. If either comparison (or both) results in an equality the DUMP command is executed. The use of parentheses determines that actual bit manipulation with the OR instruction will take place. The second of the three OR operators does not cause bit manipulation from the language syntax point of view (see "Boolean Operators").

RUN Command

The RUN command is used to relinquish control to TSS/360 for resumption of TSS/360 execution, without disconnecting the System Programmer. The format is:

Operation	Operand
RUN	[address]

address
: the symbolic or actual storage location, assumed to be an instruction address, where TSS/360 execution is to resume.

  Specified as: a symbol or L-notation.

  TSSS default: the point where TSSS received control, with TSS/360 restored.

Functional Description: Execution of the RUN command causes normal TSS/360 execution to resume; only the changes to the system that may have been effected by the System Programmer (including implanted ATs) affect system operation.

In RSS, the RUN command causes the old PSW that was stored when RSS received control to be loaded. If the RUN command has an operand, the PSW is first modified by (1) turning off the wait state bit if it is on, and (2) inserting the value of the operand into the instruction address field.

In VSS, the RUN command causes the task to be restored and control to pass to TSS/360. If the RUN command has an operand, the task's current VPSW is first modified by inserting the value of the operand into the instruction address field.

It is the System Programmer's responsibility to specify an appropriate value for the PSW being modified, if he supplies an operand with the RUN command.

When issuing a RUN command with an operand in RSS, caution should be taken to ensure that the address specified is within the addressability range of the supervisor.

Cautions: After one or more AT commands have been issued, a RUN command is required to execute the implanted ATs, except for a TSP's ATs in real storage (depending on location). The RUN command cannot be used within a dynamic statement to serve that purpose, and, in fact, a RUN in a dynamic statement is redundant unless it has an operand that differs from the default value.

When a RUN command is executed, TSS/360 status is restored as it was saved when TSSS received control, except for any changes made to that stored data by the System Programmer. It is the System Programmer's responsibility to alter stored register contents if necessary (e.g., a base register) when an operand is used with the RUN command.

Note: If a TSP uses the RUN command with an operand the address specified should not be within a virtual memory program which has a different privilege state than that of the program which was executing when VSS was invoked. The privilege state of each TSS/360 program may be found in the prologue to the CSECT listing for that program.

Programming Notes: A RUN command with an operand that occurs in a dynamic statement will cause any additional chained ATs to be ignored.

After execution of a RUN command (explicit or implied), a TSP can regain control at his terminal with an Attention. An MSP (connected by use of the external interrupt key) relinquishes the terminal to the Operator task when a RUN command is issued; to regain control (other than through an implanted AT) he must use the external interrupt key again.

Example:

```
RUN ENTRYPT
```

Execution of this command causes the address represented by the external symbol ENTRYPT to be placed in the instruction address of the current PSW for the MSP or current VPSW for a TSP, followed by the processing required to restore TSS/360 or the task, respectively, and exit to TSS/360.

STOP Command

The STOP command is used as the last command in a dynamic statement when control is to be returned to the current TSSS input device. The format is:

Operation	Operand
STOP	

There are no operands.

Functional Description: The STOP command terminates processing of the dynamic statement in which it appears; it should appear as the last command in a statement.

Programming Notes: If a STOP command occurs in a dynamic statement associated with an AT implanted in real storage by a TSP, the $ is written at the MSP terminal. All TSS/360 activity and all VSS activity is suspended until the MSP issues a RUN or DISCONNECT command.

If a STOP command is included in an immediate statement written at the terminal, the remainder of the statement is ignored and a $ is written to invite new input.

If a STOP command is encountered in a predefined statement set, reading of the card reader or tape drive is terminated and the System Programmer's terminal is solicited for input. If the mode is dynamic mode when the STOP command is executed, the SP may change the qualification. However, when an implied or explicit RUN is executed the qualification state reverts to that which was current when the associated AT was executed.

Example:

```
AT INTPROC COLLECT COLLAREA=$IPSW;STOP
```

This example repeats Example 2 under "AT Command." Execution of the STOP command results in the termination of processing of the dynamic statement and the writing of $ on the System Programmer's terminal to invite input.

CONNECT Command

The CONNECT command enables the Master System Programmer to connect a Task System Programmer at a logged-on terminal to that terminal's conversational task. The CONNECT command is not acceptable to the VSS language processor and cannot be used by a TSP. The format is:

Operation	Operand
CONNECT	taskid

taskid
: the task identification number of the task to which a TSP is to be connected.

  Specified as: any representation that can be resolved to an appropriate value (for example, an SP symbol or the system symbol $TSKID); usually a hexadecimal literal.

Functional Description: Execution of the CONNECT command causes activation of VSS within the specified task with a TSP connected to the SYSIN terminal. The MSP must

then issue a RUN or DISCONNECT command before any TSP activity is possible. When execution of the CONNECT command has been successful and the MSP has issued RUN or DISCONNECT, a $ is written at the TSP's terminal.

Caution: Specification of the taskid should be performed carefully. If an unintended value is supplied, it may represent the task ID of an unsuspecting conversational user. Or it may be the task ID of a nonconversational task, which would result in system error condition. If an invalid task ID is supplied, an RSS message is written.

Programming Notes: A conversational task's taskid is displayed at the user's terminal by TSS/360 upon successful completion of TSS/360 LOGON.

If any input from the TSP terminal is accepted prior to the writing of a $ on the terminal, that input is delivered to TSS/360.

Example:

```
CONNECT X'001C'
```

Execution of this command, issued by the MSP, results in the terminal dedicated to task 1C becoming a TSP-dedicated device, as soon as TSS/360 receives control from RSS.

Note that the operand of CONNECT must resolve to a two-byte field.

DISCONNECT Command

The DISCONNECT command is used to remove TSSS capabilities from a terminal that is dedicated to an MSP or a TSP. The format is:

Operation	Operand
DISCONNECT	

There are no operands.

Functional Description: When the DISCONNECT command is executed on behalf of the MSP, all of his ATs are removed, TSS/360 status is restored, and control reverts to TSS/360 at the point where it was interrupted by RSS activation.

When the DISCONNECT command is executed on behalf of a TSP, all of his ATs in virtual storage are removed, the status of the task is restored, and control reverts to TSS/360 with VSS no longer active in the task. The terminal is again dedicated to the task.

Note that although ATs are removed automatically at disconnect time, patches are not. However, the Patch Table is lost, as are all private SP symbols. It is the System Programmer's responsibility to save any information (for example, with the DUMP command) he wants to retain for further use.

Cautions: A DISCONNECT command is in the MSP domain if included in a dynamic statement associated with an AT implanted in real storage by a TSP; it disconnects the MSP and not the TSP.

The DISCONNECT command should not be used in a global dynamic statement.

Programming Notes: It is possible for the task with which a TSP is associated to be forced off the system or logged off. When this happens, the disconnect function is performed automatically before the logoff is complete.

If a TSP has defined global SP symbols during his terminal session, they remain in the Global Symbol Table after he disconnects and may be referenced by the global dynamic statements of other TSPs.

Example:

```
DUMP HEADER,STATUS,INFO,TABLE;DISCONNECT
```

Execution of this statement results in the contents of the four specified data fields being written on the output device specified in $DOUT, then performance of the DISCONNECT function. All of this System Programmer's ATs are removed (see "REMOVE Command"), and control then reverts to TSS/360 at the point where it was last interrupted by activation of TSSS.

CALL Command

The CALL command is used to initiate execution of a predefined set of TSSS command statements that have been stored as a deck of punched cards or on tape. The format is:

Operation	Operand
CALL	X'xxxx' C'xxxx' decimal int. sp symbol

xxxx
a device address -- physical path when preceded by X, symbolic device address when preceded by C.

Specified as: hexadecimal digits.

decimal int.
a device address which, when converted into hexadecimal, represents the physical path.

sp symbol
a data field containing a device address, with a type attribute of character to indicate symbolic device address. Any other type attribute indicates the physical path.

Specified as: a character string.

Functional Description: When the CALL command is executed (establishing call mode), statements are read from the card reader or tape drive specified by the operand and are processed exactly as if they had been keyed in at the terminal. A RUN, STOP, END, or DISCONNECT command terminates reading of the device.

Execution of a CALL command with an operand that specifies a different device also terminates reading of the current device.

Predefined statement sets may be "stacked" in such a way that one group of statements is executed sequentially until an appropriate command is reached (see above), and the following group is executed when another CALL command is issued.

If a CALL command occurs in a dynamic statement, execution of that command causes establishment of dynamic call mode. The specified device becomes the TSSS input device, and it is read as a continuation of the dynamic statement until a RUN, END, STOP, or DISCONNECT command is encountered.

Messages and any other output that would be delivered to the System Programmer's terminal when not in call mode are not affected by the change of input device; they still are delivered to the terminal (and the $ is written to invite input at the terminal).

For additional information on using predefined statement sets, see Part III.

Cautions: The continuation character (see "Terminal Usage" in Part III) cannot be used in predefined statement sets. However, tape blocks may be up to 256 characters long, whereas statements on cards can be no more than 80 characters long.

If a deck of cards is to be used in the 1056 Card Reader, it must comprise cards with an upper left corner cut. The 1056 Auto EOB switch must be on, or each card must have a 11-5-9 punch in column 80; otherwise, each read is 256 bytes of data instead of 80. (If a deck with 11-5-9 punches is used in a 2540 Card Reader, the character generated by that punch is ignored.) A predefined statement set in a 1056 must end with an END, STOP, RUN, or DISCONNECT command.

A CALL command should not be used in a TSP's global dynamic statement.

Programming Note: In addition to the commands in a set of predefined statements that terminate reading of the called device, an Attention (asynchronous interruption) from the System Programmer's terminal, a manual key external interrupt (MSP only), physical end-of-file (except 1056 Card Reader), or a permanent I/O error causes control to revert to the terminal.

Example 1:

```
CALL X'000C'
```

Execution of this command results in the device whose physical address is 00C being read for further TSSS input. Command statements are then read from that device until a RUN, END, CALL, STOP or DISCONNECT command (or one of the interrupts described under "Programming Note") is encountered.

Example 2:

```
AT LOC IF FOULUP=ON CALL INPUT
```

In this command statement, LOC is the symbolic address of an instruction location, FOULUP and ON are data fields whose contents are to be compared for equality, and INPUT is an SP symbol that represents a data field containing the address of a card reader or tape drive. After this AT is implanted at LOC, its execution causes the operand of the IF command to be evaluated and, if it is "true," the device named INPUT is read as the input device. If the operand of IF is "false," the CALL command is not executed; an implied RUN is executed instead.

END Command

The END command is used as the last command in a predefined statement set when the System Programmer wishes to have his terminal read again as the input device. The format is:

Operation	Operand
END	

There are no operands.

Functional Description: Execution of the END command causes reading of a card reader or tape drive to cease and the System Programmer's terminal to be solicited for additional input.

The END and RUN commands make it possible for the System Programmer to stack a set of predefined statements on a device, with control returned to the terminal or to TSS/360, respectively, at the end of each set. As the last command in the deck (or tape file), the END command causes control to return to the terminal without invoking the time-consuming error-recovery procedures that would result from a physical end-of-file condition.

Cautions: If an END command is included in an immediate statement written at the terminal, its execution will halt execution of the statement and will cause an error message to be written, followed by a $ to invite input.

An END command in a dynamic statement will cause execution of the statement to stop and an implied RUN to be executed. This applies to a dynamic statement in call mode (END appears in a predefined statement set that is executing as the result of a dynamic CALL command). Therefore, to return control to the terminal when in dynamic call mode, a STOP command should be used.

Example:

```
IF SWITCH=X'FF' RUN

DISPLAY C'SWITCH OFF';END
```

Assuming that the example comprises two records in a predefined statement set, execution of the first results in comparing the contents of the data field SWITCH with the literal X'FF' and, if equal, execution of the RUN command. This terminates the predefined statement set.

If the comparison yields an inequality, however, the RUN command is not executed and the next statement is read. This results in execution of the DISPLAY and END commands, the latter returning control to the terminal where a $ is written to invite input.

This section discusses general considerations for correct use of the Time Sharing Support System by a System Programmer. Applications for using TSSS are not considered here; the System Programmer should design and document his own procedures or have access to existing materials (card decks, tape volumes, or procedure descriptions).

CONNECTING THE SYSTEM PROGRAMMER

The several methods for connecting the Master System Programmer to RSS and Task System Programmers to VSS are summarized in Figure 7 and are described in detail below, except for use of the CONNECT command, which is defined in Part II. There can be only one MSP connected at a given time; several TSPs may be connected at a given time, but only one can be connected to a given task.

External Interruption Key

When the MSP is connected through use of the external interruption key, he has preempted the system operator's terminal. In this case the MSP relinquishes control when he issues the RUN command, thereby allowing the resumption of TSS/360 execution. He is still considered to be connected, but he cannot use the Attention key to communicate with RSS. (An Attention under those circumstances delivers an attention interruption to the operator task.) Except for the reactivation of RSS that occurs with the execution of an implanted AT, the MSP must again press the external interruption key to reactivate RSS and regain control at the system operator's terminal.

VSS Command

A system programmer (authority O or P) may invoke VSS within his conversational task and connect his terminal to VSS with the VSS command. He thus becomes a TSP. If the user does not have the correct authority, the command is rejected.

Operation	Operand
VSS	

Note: There are no operands.

When a TSP has been connected via use of the VSS command, the TSS/360 BACK command should not be used for that task. Also, the VSS command should not be used in the SYSIN data set for a task that has been made nonconversational with the BACK command.

Upon execution of the VSS command, a $ is printed at the terminal to invite TSSS input. This type of connection can also be made by an MSP on behalf of the TSP with the CONNECT command (see Part II, "CONNECT Command"). The CONNECT command should not specify a nonconversational task.

Action	Result[1]
External Interruption Key Pressed	MSP connected at operator's terminal; $ written at terminal to invite input; RSS mode; TSS/360 execution temporarily suspended.
VSS command at user terminal[2]	TSP connected at same terminal; $ written at terminal to invite input; VSS mode for the task; execution of task temporarily suspended.
CONNECT command issued by MSP followed by RUN or DISCONNECT[3]	TSP connected at terminal of specified task; $ written at terminal to invite input; VSS mode for the specified task; execution of task temporarily suspended.

[1] "Result" means the immediate effect of each action when it is initially taken.
[2] The VSS command is in the TSS/360 command system, and requires O or P authority for its use.
[3] The CONNECT command is RSS only; it can be issued only by the MSP.

Figure 7. Methods of connecting the TSSS user

TERMINAL USAGE

From the time the System Programmer is connected until his TSSS terminal session ends, he may use his terminal to enter TSSS command statements. Aside from differences in language syntax and the functions being performed, terminals are used similarly for TSSS and TSS/360, as described in the appropriate TSS/360 publications. Note that the remote 1052 Printer-Keyboard and 2741 terminal generate end-of-block (EOB) automatically when the RETURN key is pressed, and the 1052-7 does not. Regardless of terminal type, RETURN as used here implies EOB unless otherwise specified.

The following considerations apply uniquely to TSSS:

- Maximum statement length is 255 characters (plus EOB). With the 1052-7 Printer-Keyboard (CPU console), as many lines as desired may be entered prior to EOB, if no more than 255 characters are included. The remote 1052 and 2741 terminals generate EOB automatically with each carriage return. More than one line, up to a total of 255 characters, may be included in an input statement from a remote terminal through use of the continuation character, a hyphen (-), followed immediately by RETURN. The hyphen may appear anywhere in a statement and is not counted as an input character. Note: The continuation character may be used in the same way with the 1052-7, although it is not required.

- The character # ("pound sign") is interpreted by TSSS as a backspace character when the input device is a 1052-7 Printer-Keyboard. In that case the effect of pressing the # key is the same as pressing the backspace key on any other terminal, except that the typing element does not reverse direction. (If a backspace key is installed on a given 1052-7, either the backspace or # may be used interchangeably.)

  For each occurrence of the word "backspace" in the following paragraphs, "#" can be used interchangeably to cover the use of a 1052-7 Printer-Keyboard.

  With the Model 33 or 35 KSR teletypewriter, the left-arrow key is used as a backspace key for TSSS operations.

- Upon recognizing the end of an input statement (EOB, not preceded by a hyphen or backspace), TSSS executes the immediate portion of the statement and then writes a $ at the terminal to invite additional input, unless the statement contains an immediate RUN, CALL, or DISCONNECT command.

- Mode switching is accomplished (1) with several of the TSSS commands (defined in Part II of this publication and summarized in Appendix A), and (2) under certain circumstances with a CPU external interruption key (MSP only), Attention key (any TSP), and RETURN key (TSP only).

  For the MSP, if the mode is run mode (TSS/360 is executing), a CPU external interruption key is used to regain control at the Operator's terminal.

  Any TSP whose task's VSS is in run mode may press his Attention key to regain control at his terminal. If a $ appears VSS has been reactivated. If an underscore appears the VSS Command is required to reactivate the task's VSS routines.

- A TSP can reply to an invitation for input with RETURN only and thereby produce a TSS/360 Attention for the task; it is a special type of implied RUN. This action -- EOB constituting the only input -- is called the "void command."

  When in run mode, an Attention (causing establishment of VSS mode) followed by RETURN (reestablishing run mode and delivering a TSS/360 Attention) provides a TSP full capability for running the conversational task. However, at this time the TSP, now the task user, should first issue a TSS/360 RUN command to return the task to the point at which the initial VSS invocation interrupted it. (This holds true for any case in which an Attention is issued while a TSP is connected but VSS is not active.)

- An Attention when in RSS mode or VSS mode is recognized only if (1) call mode processing is in progress, (2) a DISPLAY or DUMP command is being executed (or is yet to be executed in the current command statement) or (3) the system is reading or writing to the SP terminal. An Attention causes termination of call mode or termination of the display or dump, followed by a message being written at the System Programmer's terminal and a $ to invite input. If not in call mode and with no current I/O or CALL, DISPLAY, or DUMP command in the current statement, normal end-of-statement processing ensues without regard to the Attention.

  Note: The System Programmer may think run mode is the current state when

actually a dynamic statement is being executed by TSSS. An Attention at that time produces the results described above.

- Pressing the external interruption key twice on the active CPU in RSS mode causes "RSS restart," which is described later in Part III under "General Operating Considerations." If the external interruption key of another CPU is pressed, the interruption is ignored.

- Errors in typing may be corrected prior to ending a line through use of the backspace key. The backspace deletes each character that is backspaced over, after which new characters may be substituted. For example, if "DISPALY XYZ" was typed, backspacing over "ZYX YLA" and then typing "LAY XYZ" results in the correct spelling of the DISPLAY command being placed in the input buffer.

- An entire line may be deleted by entering a backspace immediately followed by RETURN (EOB). If the preceding line ended with the continuation character, that line is not deleted. Note that an attempt to delete and not replace the last character of a line results in deletion of the entire line when RETURN (EOB) is pressed. In many cases, the deleted character may be replaced with a blank or a hyphen (continuation character) to prevent deletion of the line.

PREDEFINED STATEMENT SETS

Operation of TSSS via predefined statement sets is identical to terminal operation in most respects but has unique aspects as well. The language syntax is unchanged; any free-form statement that follows the prescribed format is acceptable. Some variation in the meaning of the execution of mode switching commands may occur, however; the differences are included in the command definitions in Part II and are summarized in Appendix A.

Other considerations for using predefined statement sets are:

- The continuation character is not used. Cards are read as 80-character records; tape blocks may be up to 256 characters in length.

- Cards for either a 2540 Card Read Punch or a 1056 Card Reader should be punched with the EBCDIC punch code. The optional code acceptable to TSS/360 is not correctly interpreted by TSSS. See "CALL Command" for special considerations to be observed when the input device is a 1056 Card Reader.

- When a tape drive is specified as the input device, no labels are processed. If the volume used contains one or more header labels, either the tape should be positioned manually to a valid TSSS command statement or execution of the CALL command will recognize a label as an invalid statement, which causes an error message to be written at the terminal. Additional CALL commands can then be issued until a valid statement is reached.

- Any tape block following a predefined statement set (e.g., a trailer label) that is not preceded by a RUN, END, or DISCONNECT command is read as an invalid statement, causing an error message to be written at the terminal. When a tape mark is reached, control returns to the terminal. No rewind is performed by TSSS.

- Pressing the Attention key or performing RSS restart (described under "General Operating Considerations") while in call mode terminates reading of the called device and returns control to the System Programmer's terminal. The effect on the command being executed is the same as is described under "Terminal Usage."

- Predefined statement sets on tape may be created by any available means. The TSSS SET command can be used for card-to-tape or terminal-to-tape operations. In either case, the tape drive is specified using the $IO system symbol as the first parameter in the operand. A 2540 Card Read Punch is specified as the second parameter for card-to-tape; a character literal is used as the second parameter for terminal-to-tape.

  The SET command does not write a tape mark. A predefined statement set created on tape with the SET command must always end with an END, STOP, RUN, or DISCONNECT command.

Figure 8 shows examples of each type of operation. In the second example, note the use of multiple operands with the SET command, the use of pairs of apostrophes within a character literal, and the use of a hyphen as a continuation character. (Two of the character literals are examples that appear under "COLLECT Command" in Part II.)

```
SET $IO(X'0182')=$IO(X'000C')
```

```
SET $IO(X'0182')=C'AT INTPROC COLLECT COLLAREA=$IPSW',$IO(X'0182')=C'AT NEWADDR COLL-
ECT COLLAREA=C''ASYNC'',COLLAREA=$APSW',$IO(X'0182')=C'RUN'
```

Figure 8. Card-to-Tape and Terminal-to-Tape Statement Examples

GENERAL OPERATING CONSIDERATIONS

Normal TSSS operations are performed in accordance with the information provided in the language description (Part II) and in the preceding portion of Part III. In the following paragraphs, additional information: (1) summarizes the unique aspects of operating with global qualification, (2) describes RSS restart, and (3) describes the several possible courses of action taken by TSSS when error conditions are encountered.

GLOBAL QUALIFICATION

Global operations are different for the MSP and a TSP; the two cases are discussed separately.

Global Operations for the MSP

By implanting ATs in shared virtual storage, the MSP causes all tasks using that code to initiate execution of RSS operations. Global qualification is not required; an AT in shared virtual storage is always global when implanted by RSS. If global qualification is specified, the task receiving the AT implantation is the current task by default.

In this situation, the MSP will not know the identification of the task. Therefore, removing the AT with the REMOVE command might not be possible because defaulting the virtual memory qualification again would probably occur when a different task is current. (Another step, such as "DISPLAY $TSKID," can be taken to produce a printed record of the task identification for subsequent use.)

The global dynamic statement is stored and subsequently executed by RSS, so RSS must be able to resolve all symbols that occur in the statement.

Global Operations for a TSP

Global qualification has different significance for a TSP. A TSP's global ATs are recorded in the Global AT Table, and his globally defined SP symbols are recorded in the Global Symbol Table. Both of these tables are shared by all TSPs. When a global AT is executed by any task, the Global AT Table and Global Symbol Table (if there are any SP symbols in the dynamic statement) are referenced by the VSS of that task.

Generally, a TSP establishes global qualification only (1) to issue a global AT, which automatically assigns global qualification to the associated dynamic statement, and (2) to define a global SP symbol.

A global dynamic statement can meaningfully contain only a subset of TSSS commands for execution by all tasks executing the global AT. Commands that require I/O to be performed (e.g., DUMP, DISPLAY) are suppressed when the current task does not have the required device allocated to it. Note, however, that an assumed device (one not specified in the command operand) may be available for each task to which a TSP is connected. For example, a display can be performed by the VSS of each task with a TSP connected when a global AT is executed, since a TSP terminal is already allocated to the task executing the AT.

Detection of an error condition during execution of a global dynamic statement causes an error message to be written only if a TSP is connected to that task. For any other task, the message is suppressed and the command string, in most cases, is aborted, although all applicable chained AT command strings will be executed. As a result, managing space in a collection area, say, with a conditional statement (e.g., COLLECT AREA=DATA;IF $P(AREA)=FULL DISPLAY AREA) may be difficult, since, in this example, the number of executions of the COLLECT and IF commands versus the number of suppressions of the DISPLAY command might not be known.

Note that if the RUN command is used within a global dynamic statement, its operand must specify an instruction address that is valid for all tasks that will execute the command. (If RUN has no operand, it is redundant.)

RSS RESTART

When RSS is executing, the MSP may cancel the current operation and initiate an RSS restart by pressing the external interrupt key twice on the CPU that is executing

RSS. A message indicating RSS restart in progress appears on the MSP terminal.

After performance of RSS restart the MSP tables (i.e., the AT Table, SP Symbol Table and the Patch Table) appear as they did the last time that RSS was activated. (They reflect the situation immediately prior to the last RUN issued on behalf of RSS.) The implanted AT SVCs and patches have not been removed.

Note: The initial external interruption while RSS is active is queued, but restart does not occur until the second external interruption is received. In this way TSSS guards against an accidental or unnecessary restart. If an external interruption is pending after the issuance of a RUN or DISCONNECT command, TSSS honors it by printing a $ at the MSP terminal to invite input.

ERROR CONDITIONS

A number of error conditions may be detected by TSSS, such as receipt of an unexpected program interruption. In some cases, it is possible for the current operation to be aborted and the error then ignored; the System Programmer retains control. In other cases the error is serious enough that TSSS is unable to recover.

Error Recovery and TSSS Messages

When it is possible for TSSS to retain control after encountering an error condition, an error message is written at the System Programmer's terminal. The System Programmer is then free to retry the operation or to proceed with another command statement. Messages are diagnostic only; they do not require a predetermined response. It is the System Programmer's responsibility to determine his course of action after receiving an error message.

TSSS issues error messages whenever possible regardless of whether recovery is achieved. These messages and their meanings appear in Appendix B. Class 0 messages originate with the RSS or VSS I/O system; Class 1 messages inform the System Programmer of error conditions resulting from processing his input (e.g., syntax errors, AT Table overflow); Class 2 messages reflect TSSS logic errors; and Class 3 messages indicate an RSS load/unload failure. (RSS is partially resident and partially transient; VSS resides in each task's Initial Virtual Storage.)

A system error condition (e.g., an unexpected program interruption) results in a message, and a "system logic error." In this case, TSSS does not retain control. TSS/360 is set in a wait state with only external interruptions enabled, and TSSS executes an implied RUN command.

Error Recovery with TSS/360 Aborted

Each time RSS is activated (via an external interruption key, a service request from VSS, or execution of an AT implanted in real storage), if the configuration is duplex the activating CPU places the other CPU in wait state. If this operation fails, RSS places "its" CPU in wait state and considers TSS/360 to be aborted.

When this type of abort condition occurs, the MSP can press the STOP key on the other CPU and then perform an RSS restart. Subsequently, RSS will execute but without the AT, CONNECT, DISCONNECT, and RUN functions.

No Recovery

Some unrecoverable errors result in an exit to the TSS/360 System Error Processor.

If the task to which a TSP is connected is abnormally terminated by TSS/360, a DISCONNECT is performed automatically for that TSP. Note, however, that only his virtual storage ATs are removed; any ATs implanted in real storage by that TSP can be removed only by the MSP. Note also that the TSS/360 ABEND messages are delivered to the task's SYSIN terminal.

This command summary consists of two parts: First, the formats of all commands are repeated, and under each format the restrictions on usage of the command are summarized. Figure 9 summarizes the results of execution of the mode-changing commands CALL, END, RUN, and STOP.

Operation	Operand
AT	address[,...]

An AT cannot be placed at the location of any privileged instruction in virtual storage or of any Execute, Diagnose, or SVC instruction, unless that SVC constitutes an AT implanted by the same System Programmer.

An AT is rejected when there no longer is sufficient space in the appropriate AT Table.

Operation	Operand
CALL	{X'xxxx' / C'xxxx' / decimal int. / sp symbol}

The CALL command operand must specify a card reader or a tape drive, and for a TSP the device must have been allocated to the task by TSS/360.

Operation	Operand
COLLECT	sp symbol={data field / literal}[,...]

The data being collected by the COLLECT command (specified by the second parameter in the operand) cannot be more than 4096 bytes in length.

Operation	Operand
CONNECT	taskid

The CONNECT command is invalid for a TSP. The CONNECT command is executed by RSS; therefore, the TSP being connected cannot use his terminal until the MSP issues a RUN or DISCONNECT command.

Format 1:

Operation	Operand
DEFINE	symbol[.(o,l,t,s)][,...]

Format 2:

Operation	Operand		
DEFINE	symbol=	{external symbol / sp symbol / system symbol / address}	[.(o,l,t,s)][,...]

A DEFINE command is rejected when there is insufficient space in the SP Symbol Table or, under global qualification for a TSP, in the Global Symbol Table.

Operation	Operand
DISCONNECT	

No restrictions.

Operation	Operand
DISPLAY	{data field / literal}[,...]

No restrictions.

Operation	Operand
DUMP	{data field / literal}[,...]

The data field $DOUT must have been set to the device address of a printer or tape drive prior to execution of a DUMP command. For a TSP, the device must have been allocated to the task by TSS/360.

Operation	Operand
END	

If an END command appears in a predefined statement set executing in dynamic call mode, it results in an implied RUN.

Operation	Operand
IF	expression

No restrictions.

Operation	Operand
PATCH	data field$_1$={data field / literal}[,...]

A patch cannot be placed where another patch exists if both patches have the same starting address. The length of a patch cannot exceed 4096 bytes.

Operation	Operand
QUALIFY	system symbol

The operand of a QUALIFY command issued by a TSP cannot specify the virtual memory of any task other than the one he is connected to.

Operation	Operand
REMOVE	{$AT / $PATCH}[.location][,...]

If a TSP implants an AT in real storage or inserts a patch dynamically with an AT in real storage, he cannot remove the AT or patch with execution of the REMOVE command in VSS if "location" is defaulted; each AT or patch to be removed must be specified.

Operation	Operand
RUN	[address]

If a TSP provides an operand with a RUN command, real memory qualification will be ignored; the specified instruction address is subject to dynamic address translation.

Operation	Operand
SET	data field$_1$={data field / literal}[,...]

The length of a data field specified in a SET command operand cannot exceed 4096 bytes, unless both parameters in the operand specify I/O devices and the second (the input device for this function) specifies a card reader or tape drive.

Operation	Operand
STOP	

No restrictions.

	Type of Input			
Command	Terminal Statement	Dynamic Statement	Immediate Call Mode Statement	Dynamic Call Mode Statement
CALL	Change input device, go to immediate call mode.	Go to dynamic call mode.	Change input device, same mode.	Change input device, same mode.
END	Stop processing input, error message, $ at SP terminal.	Implied RUN.	Control and $ to SP terminal.	Implied RUN.
STOP	Stop processing input, $ at SP terminal.	Control and $ to SP terminal.	Stop processing input, $ at SP terminal.	Stop processing input, $ at SP terminal.
RUN	Go to run mode.	Go to run mode.	Go to run mode.	Go to run mode.

Figure 9. Effect of execution of mode-changing commands

TSSS messages are written at the System Programmer's terminal, with the exception of Messages 00D and 00E. Nearly all of these messages are the result of error conditions encountered by the program. No prompting or confirmation messages are issued; no messages require a predetermined response from the System Programmer.

There are four message classes, each of which is used to specify a particular type of error condition:

- Class 0 messages result from errors detected by the RSS or VSS I/O system.
- Class 1 messages result from System Programmer errors.
- Class 2 messages result from system errors; these may be either TSSS logic errors or TSS/360 errors such as inaccurate tables.
- Class 3 messages result from failure of the RSS load or unload function.

In almost all cases, detection of an error that causes a message to be written also causes the current operation to be canceled. Depending on the type, severity, and location of the error, a different operation may subsequently be successful.

The output format for TSSS messages is:

CcHccxyy text of 24 characters or less

c
: represents an alphabetic character; the first five characters of a message constitute the module identification of the module that detected the error condition.

x
: represents the message class; its value is 0, 1, 2, or 3.

yy
: represents the message number, in hexadecimal, beginning with 00 for each class.

In addition to the basic message line, Class 0 messages include some combination of the following, each on a separate line and labeled for identification: symbolic device address, seek address, physical path, CSW, PSW, sense data.

If a message results from an error encountered during execution of a dynamic statement, the source statement is printed out below the message (or, if Class 0, below the I/O data described above). Note, however, that in dynamic call mode, only the stored dynamic statement is printed out, not the statements of the predefined statement set.

The following is a complete list of TSSS messages. Class number and message number are shown in each case; in output format each message line also would include a module ID. A brief explanation of the message follows the text. Note that Class 0, 2, and 3 messages in many cases require knowledge of the internal structure of TSSS to be fully understandable. The TSSS Program Logic Manual, GY28-2022, can be used to obtain that knowledge.

The action taken by TSSS after writing a given message is listed separately. It is the same in most cases and is listed as "standard," which is defined as follows: The current operation is canceled at the point where the error was encountered, and after the message is written, TSSS invites input from the terminal unless the error was encountered during the processing of a global dynamic statement for a task with no TSP connected. In the latter case, the standard system action is to execute an implied RUN. (Note that a STOP or DISCONNECT command appearing in a dynamic statement may not be executed because of an error encountered during execution of an earlier command in the same statement.)

If an RSS error message cannot be written at the Operator's terminal, RSS exits to the TSS/360 System Error Processor. If a VSS message cannot be written to a TSP terminal (the error occurs while VSS is active in a task to which a TSP is connected), VSS calls the TSS/360 System Error Processor. When the task's VSS routines are returned to, the TSP is disconnected.

0 00 INVAL OPERATION FOR DEVC

Explanation: I/O system was requested to perform an impossible operation, such as reading a printer.

System Action: Standard.

0 01 DEVICE NOT SUPPORTED

Explanation: Self-explanatory.

System Action: Standard.

0 02 INVALID LENGTH SPECIFIED

Explanation: Module requesting I/O specified invalid length parameter.

System Action: Standard.

0 03 DEVICE NOT OPERATIONAL

Explanation: Condition code 3 received when Start I/O executed.

System Action: Standard.

0 04 NO SSDAT DEVICE ENTRY

Explanation: Requested device not represented in SSDAT.

System Action: Standard.

0 05 PAGE BOUNDARY VIOLATION

Explanation: Data crosses page boundary on single I/O request.

System Action: Standard.

0 06 PERMANENT I/O ERROR

Explanation: Error recovery retry attempts have failed, or CSW indicates unrecoverable error.

System Action: Standard, in most cases. If the failing device is the input device while in call mode (card reader or tape drive), the terminal becomes the input device again.

0 07 SYSTEM LOGIC ERROR

Explanation: An internal logic error has occurred in TSSS. Should not happen.

System Action: Standard.

0 08 DEVICE NOT ALLOCATED

Explanation: Program check, interruption code X'30', received. (VSS only)

System Action: Standard.

0 09 LOAD REAL ADDRESS FAILED

Explanation: Hardware failure of LRA instruction. (RSS only)

System Action: Standard.

0 0A INVAL FORMS MOTION BYTE

Explanation: First byte in a print line is not valid as TSSS carriage control data.

System Action: Standard.

0 0B PAGELST ENTRIES EXCEED 8

Explanation: Requested channel program exceeds TSS/360 limitation of eight entries in page table in IORCB. (VSS only)

System Action: Standard.

0 0C INVALID SSDAT ENTRY

Explanation: The table entry contains incompatible or meaningless data.

System Action: Standard.

0 0D INTERVENTION REQUIRED

Explanation: If message 00E (below) cannot be used because the Operator's terminal is not ready, and if the Operator's terminal is not a 1052-7 (the audible alarm would be sounded if it were), this message is written to the MSP terminal.

System Action: An attempt is made repeatedly to initiate the I/O operation on the device.

0 0E INTERVENTION REQUIRED

Explanation: This self-explanatory message is written to the Operator's terminal when a TSSS device requires it.

System Action: An attempt is made repeatedly to initiate the I/O operation on the device.

0 0F NO PHYSICAL PATH EXISTS

Explanation: The SSDAT does not define a physical path for a given symbolic device address.

System Action: Standard.

0 10 CHANNEL PROG EXCEEDS MAX

Explanation: The number of CCWs required to fulfill the I/O request is greater than available space for building the channel program.

System Action: Standard.

0 11 SP ATTENTION RECEIVED

Explanation: An Attention by a remote MSP or a TSP, issued while RSS or VSS I/O is executing at the terminal, is acknowledged.

System Action: The System Programmer's terminal is solicited for input.

1 00 INVAL SYST SYMBOL PARAM

Explanation: Parenthesized parameter of a system symbol is incorrectly specified. Does not apply to $RM or $VM.

System Action: Standard.

1 01 NO 'PATCH' AT THIS LOC

Explanation: The $PATCH system symbol's parameter does not specify a valid patch location.

System Action: Standard.

1 02 NO 'AT' AT THIS LOCATION

Explanation: The $AT system symbol's parameter does not specify a valid AT location.

System Action: Standard.

1 03 UNDEFINED ALIAS SYMBOL

Explanation: The symbol used as the second parameter in format 2 of the DEFINE command cannot be resolved.

System Action: Standard.

1 04 TSP ALREADY CONNECTED

Explanation: The operand of a CONNECT command specifies a task to which a TSP is already connected. (RSS only)

System Action: CONNECT command is rejected; the MSP input device is read again.

1 05 INVALID DEVICE SPECIFIED

Explanation: Self-explanatory.

System Action: Standard.

1 06 UNDEFINED SYMBOL

Explanation: Search of appropriate symbol tables found no match.

System Action: Standard.

1 07 WRONG NO. OF PARAMETERS

Explanation: Self-explanatory.

System Action: Standard.

1 08 SP SYMBOL TABLE FULL

Explanation: Self-explanatory. If this was a format 1 use of the DEFINE command, there may be sufficient space for a format 2 DEFINE command or a format 1 command that requests less storage.

System Action: DEFINE command rejected. The input device is read again.

1 09 INVALID TASK ID

Explanation: The task identification number supplied with a CONNECT or QUALIFY command was not found in a search of the TSI chain. (RSS only)

System Action: Standard.

1 0A INVALID VM ADDRESS

Explanation: Self-explanatory.

System Action: Standard.

1 0B INVALID $ID PARAMETER

Explanation: A dictionary search failed to find an appropriate symbol.

System Action: Standard.

1 0C INVALID CPU SPECIFIED

Explanation: Incorrect specification of a parameter with the $RM system symbol.

System Action: Standard.

1 0D INVALID $DOUT DEVICE

Explanation: Self-explanatory.

System Action: Standard.

1 0E INVALID MEMORY QUALIFIER

Explanation: Self-explanatory.

System Action: Standard.

1 0F BAD PARAM;OPERATION DONE

Explanation: Operation requested was incorrectly specified but performed anyway. Result may be unpredictable.

System Action: Processing continues as if no error occurred.

1 10 OPERATION NOT PERFORMED

Explanation: Operation requested could not be performed because of incorrect specification.

System Action: Standard.

1 11 INVALID LITERAL

Explanation: Incorrect specification of a literal, which can include "L-notation."

System Action: Standard.

1 12 INVALID SYMBOL

Explanation: A symbol contains an invalid character or too many characters.

System Action: Standard.

1 13 INCORRECT SYNTAX

Explanation: Self-explanatory.

System Action: Standard.

1 14 INPUT STRING TOO LONG

Explanation: Self-explanatory.

System Action: Standard.

1 15 PATCH TABLE OVERFLOW

Explanation: Self-explanatory; possibly a shorter patch will be accepted.

System Action: PATCH command rejected; input device is read again.

1 16 REDUNDANT PATCH

Explanation: A patch already exists at specified location.

System Action: PATCH command rejected; input device is read again.

1 17 INVALID RSS ADDRESS

Explanation: A segment 2 or 3 address cannot be resolved with the RSS page tables.

System Action: Standard.

1 18 INVALID RM ADDRESS

Explanation: A segment 0 or 1 address cannot be resolved with the RSS page tables.

System Action: Standard.

1 19 INVALID 'AT' LOCATION

Explanation: An excluded instruction resides at specified location.

System Action: AT command rejected; input device is read again.

1 1A 'AT' TABLE OVERFLOW

Explanation: Self-explanatory; possibly a shorter dynamic statement would be accepted.

System Action: AT command rejected; input device is read again.

1 1B INVALID INPUT CHARACTER

Explanation: Self-explanatory.

System Action: Standard.

1 1C INVALID RECORD ADDRESS

Explanation: Self-explanatory; this occurs when a dump or display involves reading from a direct access device.

System Action: Standard.

1 1D IMPROPER CALL COMMAND

Explanation: Self-explanatory.

System Action: Standard.

1 1E INVALID REMOVE OPERAND

Explanation: Operand not $AT or $PATCH, or accompanying parameter is invalid.

System Action: Standard.

1 1F END COMMAND INVALID

Explanation: Incorrect use of the END command.

System Action: Standard.

1 20 IMPROPER SUBSCRIPT

Explanation: Self-explanatory.

System Action: Standard.

1 21 SP ATTENTION RECEIVED

Explanation: An Attention by the System Programmer, issued while RSS or VSS was executing, is acknowledged.

System Action: The interrupt may have caused call mode or a display or dump to be terminated; see "DISPLAY Command" and "DUMP Command." The System Programmer's terminal is solicited for input.

1 22 COLLECT PARAM 1 INVALID

Explanation: The parameter that specifies the collection area in a COLLECT command operand is invalid.

System Action: Standard.

1 23 INVALID LENGTH

Explanation: More than 4096 bytes specified by a SET, PATCH, or COLLECT command.

System Action: Standard.

1 24 OVERFLOW CONDITION

Explanation: Either truncation was performed during execution of a SET or COLLECT command, or a fixed-point overflow exception resulted from an arithmetic operation.

System Action: Processing continues as if no error occurred.

1 25 IMPROPER IF

Explanation: The logical expression that constitutes the IF operand cannot be evaluated as either true or false.

System Action: Standard.

1 26 ATTRIBUTE ERROR

Explanation: One or more parameters incorrectly specified with immediate attribute designation.

System Action: Standard.

1 27 RANGE OPERATOR IN ERROR

Explanation: Self-explanatory.

System Action: Standard.

1 28 DUMPED TO xxxxxxxx EOR

Explanation: During execution of the DUMP command when the output device is a tape drive, the end of the tape reel has been reached. xxxxxxxx represents the storage address (in hexadecimal) of the last byte of data written to the tape.

System Action: Standard. If the remainder of the dump is to be written, the reel should be changed on the current output device (or the value of $DOUT changed) and a new DUMP command issued.

1 29 INVALID READ ONLY PAGE

Explanation: The PUT operation was discontinued because the virtual memory page was in TWAIT migration, in transit, or unprocessed by the dynamic loader.

System Action: Standard. The operation should be tried again after issuing the RUN command.

2 00 AT RELOCATION AREA OVFLO

Explanation: The eight AT relocation areas for this type of AT are all locked, and an attempt was made to use them. See Appendix D for a definition of the AT relocation areas and a description of their use.

System Action: The System Programmer's terminal is solicited for input. A recovery attempt can be made (see Appendix D). Alternatively, other commands can be issued. However, any ATs that reference the locked AT relocation area cannot be executed; if encountered, those ATs will cause this message to be issued.

2 01 UNEXPECTED 'AT' SVC

Explanation: A spurious AT SVC execution occurred, for which there is no entry in the AT Table.

System Action: After this message is printed, Message 20E and the accompanying PSW are printed, followed by an implied RUN at the instruction address following the spurious AT SVC.

2 02 DEVICE NOT IN SSDAT

Explanation: A routine seeking a symbolic device address in the SSDAT found no entry for the device.

System Action: Standard.

2 03 PAGE NOT IN TSS XPT

Explanation: An addressed page flagged "not in core" could not be found in the External Page Table. (RSS only)

System Action: Standard.

2 04 TSS RSPI CHAIN ERROR

Explanation: Search for an address in shared virtual storage failed because of erroneous pointers in Resident Shared Page Index. (RSS only)

System Action: Standard.

2 05 SCB COUNT IN ERROR

Explanation: Language processing has detected an invalid count of Symbol Control Blocks.

System Action: Standard.

2 06 INVALID SCB TYPE

Explanation: Internal logic error; detected during language processing.

System Action: Standard.

2 07 LOAD REAL ADDRESS FAILED

Explanation: Hardware failure of the LRA instruction, which was executed on behalf of VSS. (VSS only)

System Action: Standard.

2 08 ERROR COND-'AT' REMOVED

Explanation: (1) If this message was immediately preceded by another message, a system error detected during AT processing caused the previous message to be written; the offending AT has now been removed. (RSS only) (2) If this message is written without another diagnostic message immediately preceding it, it means that a branch or LPSW instruction overlaid by an AT SVC could not be simulated after the associated dynamic statement was executed. The AT has been removed.

System Action: Message 20E is written (AT SVC PSW FOLLOWS), and an implied RUN is executed. Note that in the case described by explanation (2), a program check can be expected upon execution of the previously overlaid instruction.

2 09 INVALID PAGING DEVC TYPE

Explanation: Self-explanatory. (RSS only)

System Action: Standard.

2 0A ADDR EXCEEDS 2301 RANGE

Explanation: Invalid relative page for 2301 drum. (RSS only)

System Action: Standard.

2 0B ADDR EXCEEDS 2311 RANGE

Explanation: Invalid relative page for 2311 disk. (RSS only)

System Action: Standard.

2 0C ADDR EXCEEDS 2314 RANGE

Explanation: Invalid relative page for 2314 disk. (RSS only)

System Action: Standard.

2 0D SYSTEM LOGIC ERROR

Explanation: An internal logic error has occurred in TSSS. Should not happen.

System Action: Standard.

2 0E AT SVC PSW FOLLOWS

Explanation: This message immediately follows Message 201 or Message 208. This message may also follow the diagnostic message that is issued when a system attempt to remove an AT is unsuccessful (see Message 208). In each case, the AT can be identified by the instruction address in the PSW (VPSW in VSS) that is written immediately following this message.

System Action: An implied RUN is executed.

3 00 SYSTEM UNLOAD FAIL-READ

Explanation: Permanent I/O error while attempting to read in TSS/360 resident supervisor pages during RSS unload function. (RSS only)

System Action: Exit to TSS/360 System Error Processor.

3 01 SYSTEM UNLOAD FAIL-WRITE

Explanation: Permanent I/O error while attempting to write RSS to its external residence device. (RSS only)

System Action: Exit to TSS/360 System Error Processor.

3 02 TSSS SYSERR-UNLOAD FAIL

Explanation: System logic error makes unload function impossible. (RSS only)

System Action: Exit to TSS/360 System Error Processor.

3 03 SYSTEM LOAD FAIL - READ

Explanation: Permanent I/O error while attempting to read in a transient RSS page. (RSS only)

System Action: Exit to TSS/360 System Error Processor.

3 04 SYSTEM LOAD FAIL - WRITE

Explanation: Permanent I/O error while attempting to write TSS/360 resident supervisor pages to temporary external location. (RSS only)

System Action: Exit to TSS/360 System Error Processor.

3 05 TSSS SYSERR - LOAD FAIL

Explanation: System logic error makes load of transient RSS modules impossible. (RSS only)

System Action: Exit to TSS/360 System Error Processor.

3 06 RSS RESTART IN PROGRESS

Explanation: Restart has been requested by the MSP by pressing the CPU external interrupt key.

System Action: RSS restart is performed. Upon completion, RSS retains control and invites input at the terminal by printing a $.

3 08 SYSTEM LOGIC ERROR

Explanation: A TSSS module has encountered a program interruption which is not a paging interruption. For example, a bad operation code may have been encountered.

System Action: Standard.

APPENDIX C: OUTPUT FORMATS

This appendix describes the format of each type of output generated by TSSS except messages. The message output format is described in Appendix B, "TSSS Messages."

DISPLAY Command Output Formats

Execution of a DISPLAY command whose operand specifies a data field in real or virtual storage (with exceptions noted below) results in output at the System Programmer's terminal in a format that varies according to the type attribute of the data field. Starting with the first line, every second line of data is preceded by the storage address of the first byte of data on that line. The formats are as follows:

- hexadecimal -- four words consisting of eight digits each, separated by spaces
- decimal integer -- three words, each consisting of ten digits plus an algebraic sign, separated by spaces
- character -- a continuous character string, with a period (.) inserted in any position for which there is no printable EBCDIC graphic

If more than one operand is used with a DISPLAY command, the display for each operand begins on a new line and is formatted according to type as described above.

If a display of the result of an arithmetic or Boolean operation is performed, the address preceding the data represents a TSSS buffer or working storage location.

If a range is specified as the DISPLAY command operand and the two parameters have different type attributes, the display is in hexadecimal.

If a DISPLAY operand specifies an I/O device, type is assumed to be hexadecimal. The address preceding each line of data is relative to the beginning of the block, starting with zero unless an offset was specified.

Exceptions: The system symbols $AT, $PATCH, $MAP, and $ID used as operands of DISPLAY commands cause different formatting to be performed, as described under other headings in this appendix.

If a character, decimal integer, or hexadecimal literal is the operand of a DISPLAY (or DUMP) command, no storage address precedes the literal on the print line.

If the system symbol $DOUT is used as the operand of a DISPLAY command, the output format is:

```
┌──────────────────────────────┐
│      $IO(C'xxxx',,,,yy)      │
└──────────────────────────────┘
```

xxxx
: represents the hexadecimal data contained in the data field $DOUT, assumed to be the symbolic device address of a printer or tape drive.

yy
: represents the mode set byte for a tape drive. For a nine-track tape drive or for a printer, the value is normally zero.

The storage address of the $DOUT data field does not precede the display.

DUMP Command Output Formats

The output resulting from execution of a DUMP command is identical to that described under "DISPLAY Command Output Formats," except for the following:

- Each line of data is preceded by a storage address.
- Each line contains twice as much data; that is, eight eight-digit words of hexadecimal data, six ten-digit words of decimal integer data, or a correspondingly longer character string.
- Each operand of a DUMP command causes a new dump to be initiated, starting a new page with a header line that reads "RSS STORAGE PRINT" or "VSS STORAGE PRINT" as appropriate. Three skip lines follow the header line. Dump output may be labeled with a subheading by use of the $DHDR system symbol.
- If the DUMP operand specifies a direct access device, the cylinder/track/record address appears on a separate line above the first data line.

The logical record format for all dumps is as follows:

Byte 1: forms motion byte

Bytes 2-132: print data

Hexadecimal output resulting from the execution of a DUMP command includes character-format data colinear to the hexadecimal data representation. The logical record length of a TSSS dump with character and hexadecimal data is 132 bytes, and the block size is 4092 bytes (31 logical records per physical record).

The line format for hexadecimal dumps is as follows:

Print positions 1-8: data address

Print positions 13-20, 23-30, 33-40, 43-50: hexadecimal data

Print positions 51-54: blank

Print positions 55-62, 65-72, 75-82, 85-92: hexadecimal data

Print position 92: an asterisk

Print positions 99-130: character data

Print position 131: an asterisk

The same formatting exceptions apply to both the DISPLAY and DUMP commands.

The system symbols $TASK and $STATUS, used as operands of the DUMP command, result in different formatting, which is described in this appendix under the particular headings.

Printing TSSS Dump Tapes

The procedure for printing a TSSS dump tape is as follows:

Since the tape is unlabeled, it is first necessary to issue the DDEF command with the following operands:

```
DDNAME=name,PS,DSNAME=dsname,

DCB=(RECFM=FB,LRECL=132,BLKSIZE=4092,

    DEN={0|1|2},TRTCH={C|E|T}),

UNIT=(TA,tape type),VOLUME=(,volume
serial)

LABEL=(,NL),DISP=OLD
```

(For detailed description of the above command, see Command System User's Guide, GC28-2001.)

The PRINT command is then issued, and must adhere to the following form:

Operation	Operand
PRINT	DSNAME=data set name [,STARTNO=first volume] [,ENDNO=last volume] [,PRTSP=EDIT][,ERASE={Y|N}] [,ERROROPT={ACCEPT|SKIP|END}] [,FORM=paper form] [,STATION=station id]

DSNAME
identifies the data set that is to be printed.

Specified as: a fully qualified data set name.

STARTNO
specifies the sequential position of the first dump to be printed.

Specified as: a decimal number between 1 and 131.

ENDNO
specifies the sequential position of the last dump on the tape to be printed. If STARTNO and ENDNO are entered as the same number, the one volume specified is printed.

Specified as: a decimal number between 1 and 131.

System default: If STARTNO and ENDNO are defaulted, all dumps on the tape are printed.

PRTSP
this parameter is required for the printing of a multiple-volume data set.

Specified as: EDIT

System default: only the first volume is printed.

ERASE
specifies that the catalog entry created by the PRINT command is to be deleted. If it is desirable to keep the catalog entry without having to DDEF again, this parameter should be omitted.

Specified as: Y or N

System default: N; there is no deletion of the catalog entry.

ERROROPT
designates the action to be taken when an uncorrectable error is found while reading a data set record.

Specified as:
ACCEPT -- error record will be accepted

SKIP -- error record will be skipped

END -- print operation will be terminated

System default: END is assumed.

FORM
designates the form number of the printer paper to be used.

Specified as: one-to-six alphameric characters.

System default: the installation's standard printer form is used.

STATION
indicates identity of a remote station to which the output is to be directed.

Specified as: one-to-eight alphameric characters.

System default: station ID from Task Common is used.

$AT and $PATCH Output Formats

When the system symbol $AT or $PATCH is used as the operand of a DISPLAY or DUMP command, the output appears in a special format. Information extracted from the AT Table or Patch Table is formatted into one or more print lines (50 bytes per line for a display, 101 bytes per line for a dump) without a storage address preceding the formatted data.

Each execution of the DISPLAY or DUMP command (there is one execution per operand) causes a header line to precede the data, identifying the data fields by relative position. If the $AT or $PATCH operand has no accompanying parameter, all outstanding ATs or patches are formatted under the single header line.

When $AT is the operand, the information written to the output device consists of:

- the qualification of the AT address, expressed as the characters RM or VM
- the hexadecimal address of the implanted SVC
- the qualification of the AT command string
- the character-string text of the stored dynamic statement.

The header under which this data is written appears as:

```
QUALIFICATION ADDRESS QUALIFICATION COMMAND TEXT
```

When $PATCH is the operand, the information written to the output device consists of:

- the qualification of the patch, expressed as the characters RM, VM, or EXT (external device)
- the address of the patch
- the original data at the patch location
- the patch data.

The address is a hexadecimal storage address for RM or VM qualification; for EXT qualification, the address specifies a location on a direct access device in the following format:

meaning of data	symbolic device address	cylinder	track	record	offset
length in hex digits	4	2	2	2	4

The header under which this data is written appears as:

```
QUALIFICATION   ADDRESS   ORIGINAL DATA   PATCH DATA
```

$MAP Output Format

When the $MAP system symbol is used as the operand of a DISPLAY or DUMP command, the output consists of a group of symbol-address elements in the following format:

Data	symbol	blank	hex address	blanks
No. of print Characters	8	1	8 or 9	2 or 3

Each element is 20 characters long.

Real Storage: The address field is eight digits long and is following by three blanks. One display line contains two elements (40 characters); one print line contains five elements (100 characters). In either case, in RSS the elements are ordered by address, starting with the lowest, whereas in VSS the elements are ordered alphabetically.

In the case of a dump, the storage map is preceded by a header line that reads RM STORAGE MAP.

Virtual Storage: When the storage map element is a CSECT or PSECT name, the address is eight digits long and is followed by three blanks. When the storage map element is an entry point within the control section, the address portion consists of eight hexadecimal digits followed by an asterisk, after which there are only two blanks.

One display line contains two elements (40 characters); one print line contains five elements (100 characters). The elements are ordered by control section address, starting with the lowest, with each group of entry points in a control section occurring in the same order in which they are found in the Control Section Dictionary.

In the case of a dump, the storage map is preceded by a header line that reads VM STORAGE MAP.

$ID Output Format

When the $ID system symbol is used as the operand of a DISPLAY or DUMP command, the output format is identical to the $MAP output format. The only difference is that $ID output consists of one element. Note that the input address supplied by the System Programmer is not included in the output format (unless it happens to correspond to the symbol being returned).

$STATUS Output Format

When the $STATUS symbol is used as the operand of a DUMP command to produce a formatted dump of all system status indicators, the output data is formatted as follows:

Print Line Number	Contents
1	Primary Header
2	$DHDR
3	Task ID, CPU ID
4	Current PSW
5, 6, 7	General Registers
8, 9, 10	Control Registers
11, 12	Floating Point Registers
13, 14, 15	Old PSWs
16	Channel Address Word, Channel Status Word
17-21	TSI
22-34	XTSI Header

$TASK Output Format

When the $TASK symbol is used as the operand of a DUMP command to produce a dump of all task status indicators, the output data is formatted as follows:

Print Line Number	Contents
1	Primary Header
2	$DHDR
3	Task ID, User ID
4	Task's Current PSW
5, 6, 7	General Registers
8, 9, 10	Control Registers
11, 12	Floating Point Registers
13, 14, 15, 16	Old Virtual PSWs
17-21	Task's TSI
22-34	Task's TSI Header

APPENDIX D: AT RELOCATION AREAS

If a System Programmer is to recover from the error condition signaled by Message 200, AT RELOCATION AREA OVFLO, he must understand the information provided in this appendix.

AT Processing

Control-program processing of an AT (as opposed to processing of the associated stored dynamic statement) involves the use of an "AT relocation area." An RSS real-storage CSECT comprises eight contiguous areas for ATs in real storage; two virtual storage CSECTs, one for use by RSS, the other for use by VSS, each comprises eight contiguous AT relocation areas for ATs located in a task's virtual storage. The CSECT names and formats for these AT relocation areas are depicted in Figure 10.

An AT is an SVC instruction that overlays a TSS/360 instruction at a specified location and is accompanied by a block of control information that includes a copy of the overlaid instruction. To resume TSS/360 execution at the point of interruption, after execution of an AT's dynamic statement, a copy of the overlaid instruction is executed in an AT relocation area. (It is simulated if it is a branch or LPSW instruction.) The contiguous "return SVC" instruction (see Figure 10) then makes it possible for the PSW instruction address to be set to the appropriate TSS/360 next sequential instruction.

Use of AT Relocation CSECTs

An AT relocation area is filled in dynamically with each execution of an AT SVC, locked, and then unlocked upon exit to TSS/360. During normal operations, only the first AT relocation area in a CSECT is used, since processing of each AT is normally completed before another AT SVC is executed.

The instruction overlaid by an AT SVC can produce a program check when executed, of course. Control then goes to the TSS/360 program interrupt processing routines. If an AT located in that portion of the resident supervisor also produces a program check, an infinite loop could result. However, as soon as all eight AT relocation areas are locked, the next execution of the offending AT results in Message 200 being written.

Recovery

If the System Programmer wishes to attempt recovery from the error condition described above, he can do the following:

1. DISPLAY the AT relocation CSECT to identify the ATs involved.
2. REMOVE the offending ATs.
3. Unlock at least one AT relocation area.

CEHJAB (RSS CSECT[1])

lock byte	overlaid instruction	return SVC	instr. length	AT SVC address	pointer to lock byte	unused
2	2 to 6	2	2	4	4	0 to 4

length in bytes (total = 20)

CZHZAB (VSS CSECT[1])

lock byte	overlaid instruction	return SVC	instr. length	AT SVC address	pointer to lock byte	character identifier[2]	unused
2	2 to 6	2	2	4	4	8	0 to 4

length in bytes (total = 28)

[1] Each CSECT contains eight contiguous, identical AT relocation areas.

[2] Character identifier = CEHJAVAT for an RSS-implanted AT.
Character identifier = CZHZAVAT for a VSS-implanted AT.

Figure 10. Formats of AT Relocation Areas

The command statement for zeroing the entire AT relocation CSECT in RSS is:

```
SET CEHJAB.(,160)=X'00'
CZHZAC.(,224)=X'00'
```

For the VSS CSECT the statement is:

```
SET CZHZAB.(,224)=X'00'
```

Note: If the RSS AT relocation CSECT becomes fully locked due to errors encountered with a TSP's ATs in real storage, Message 200 is written at the MSP terminal (Operator's terminal if no MSP is connected).

Where more than one page reference is given, the first page number indicates the major reference.

GC28-2006-1

IBM S/360 TSSS Printed in U.S.A. GC28-2006

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]