LY20-8032-3
File No. S370-22

Program Product

VS APL Program Logic

Program Number 5748-AP1

Release 4

IBM

This publication was produced using the
IBM Document Composition Facility
(program number 5748-XX9)
and the master was printed on the
IBM 3800 Printing Subsystem.

Fourth Edition (August 1981)

This is a major revision of, and makes obsolete, LY20-8032-2.

This edition applies to Release 4 of VS APL, Program Product 5748-AP1, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

The changes for this edition are summarized under "Summary of Amendments" following the preface. Because the technical changes in this edition are extensive and difficult to localize, they are not marked by vertical bars in the left margin.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370 and 4300 Processors Bibliography, GC20-0001, for the editions that are applicable and current.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

PREFACE

This book contains information for programming support representatives and system programmers who maintain VS APL. When used with VS APL source-program listings, it enables them to understand the internal operation of VS APL and to maintain the system.

The book is divided into six sections:

- "Section 1. Introduction," is an overview of the VS APL program product.
- "Section 2. Method of Operation," contains Hierarchy Input Process Output (HIPO) diagrams that describe the functions performed.
- "Section 3. Program Organization," lists the entry points to routines in alphabetic order. It contains, for each entry point, a description of the function of its routine, the name of the module in which it is contained, the names of entry points from which it is called, and the names of entry points that it calls.
- "Section 4. Directory," lists the entry points in alphabetic order with the names of their containing modules and the number of the HIPO diagram referring to that module, if any. It also contains the same information in alphabetic order by module name.
- "Section 5. Data Areas," describes the VS APL workspace and the functions performed by the VS APL interpreter, and shows the formats of control blocks.
- "Section 6. Diagnostic Aids," describes serviceability aids and other information helpful in reading the program listings, and in detecting, tracing, and documenting problems in VS APL.

PREREQUISITE KNOWLEDGE

The prerequisite knowledge for using this publication is a basic understanding of VS APL concepts and other related information found in the _VS APL General Information_, _VS APL for CICS/VS: Terminal User's Guide_, _VS APL for CMS: Terminal User's Guide_, _VS APL for TSO: Terminal User's Guide_, and _VS APL for VSPC: Terminal User's Guide_.

PREREQUISITE PUBLICATIONS

- _VS APL General Information_, GH20-9064
- _VS APL for CICS/VS: Terminal User's Guide_, SH20-9167
- _VS APL for CMS: Terminal User's Guide_, SH20-9067
- _VS APL for TSO: Terminal User's Guide_, SH20-9180
- _VS APL for VSPC: Terminal User's Guide_, SH20-9066
- _VS TSIO Guide and Reference_, SH20-9107

RELATED PUBLICATIONS

- VM/370: Planning and System Generation Guide, GC20-1801
- OS/VS2 System Programming Library: System Generation Reference, GC26-3792
- Customer Information Control System/Virtual Storage (CICS/VS) Version 1, Release 5 General Information, GC33-0066
- Customer Information Control System/Virtual Storage (CICS/VS) Version 1, Release 5 Application Programmer's Reference Manual, GC33-0077
- Customer Information Control System/Virtual Storage (CICS/VS) Version 1, Release 5 Problem Determination Guide, SC33-0089
- VS APL for CICS/VS: Installation Reference Material, SH20-9181
- VS APL for CICS/VS: Writing Auxiliary Processors, SH20-9168
- VS APL for CMS: Installation Reference Material, SH20-9182
- IBM Virtual Machine/System Product Logic and Problem Determination Guide

  Vol.1: Control Program (CP), LY20-0892

  Vol.2: Conversational Monitor System (CMS), LY20-0893
- VS APL for CMS and TSO: Writing Auxiliary Processors, SH20-9068
- VS APL for TSO: Installation Reference Material, SH20-9183
- OS/VS2 TSO Terminal User's Guide, GC28-0645
- VS Personal Computing (VSPC) for OS/VS and DOS/VS: General Information, GH20-9070
- OS/VS1 and OS/VS2 MVS VS Personal Computing (VSPC) Logic, LY20-8072
- DOS/VS VS Personal Computing (VSPC) Logic, LY20-8039
- VS APL for VSPC: Installation Reference Material, SH20-9184
- VS Personal Computing (VSPC): Writing Processors, SH20-9074
- IBM Virtual Machine Facility/370: CP Command Reference for General Users, GC20-1820
- A Guide to Writing a Terminal Monitor and Program Command Processor, GC28-0648
- OS/VS2 System Programming Library: Supervisor, GC28-0628
- IBM System/370 Principles of Operation, GA22-7000

SUMMARY OF AMENDMENTS

RELEASE 4, AUGUST 1981

VS APL SESSION MANAGER

New Programming Feature

The VS APL Session Manager component of the program product is now available under CICS/VS.

AUXILIARY PROCESSORS

New Programming Feature

New auxiliary processors have been added to VS APL under CICS/VS and VSPC as follows:

AP 120: VS APL Session Manager Command, for CICS/VS
AP 126: GDDM, for CICS/VS and VSPC

VSAM Auxiliary Processor Enhancements

The functions of AP 123 are now available under CICS/VS and VSPC.

MAINTENANCE

As reflected in the Table of Contents, the Data Areas section has been restructured for ease of use as follows: Interpreter Data Areas, Executor Data Areas and Control Block Formats. Under Control Block Formats, data areas (and the system components which employ them) are ordered alphabetically.

RELEASE 4, MARCH 1981

VS APL UNDER TSO

New Programming Feature

VS APL under TSO (Time Sharing Option) is now included in the program product. Information about VS APL under TSO has been added to this book.

VS APL SESSION MANAGER

New Programming Feature

The VS APL session manager is a new component of the program product, and is available for users under CMS and TSO. It provides a set of commands by which the user may control the VS APL session, produces a record of the session (called a "session log"), and enables the user to scroll through the session log. A Method of Operation diagram has been added for it.

AUXILIARY PROCESSORS

New Programming Feature

New auxiliary processors have been added to APL under CMS and TSO. The lists of auxiliary processors and the Method of Operation diagrams have been amended to reflect these additions. The new processors are:

AP 120: VS APL Session Manager Command, for CMS and TSO
AP 121: APL Data File, for CMS
AP 126: GDDM, for CMS and TSO

VSAM Auxiliary Processor Enhancements

AP 123 will now support the following functions under CMS and TSO:

Record Search by generic key
Record search by key greater than or equal to
Access to relative record data sets
Direct access to entry-sequenced data sets
Direct query for record identification
Alternate indexing with duplicate key support
Reusable files

Documentation Change

The names of several auxiliary processors have been changed to reflect more clearly their functions as well as to provide consistency among subsystems. The following table gives the old and new names of each renamed auxiliary processor, by subsystem.

Subsystem	Old Name	New Name
CICS/VS	APL Format	APL Data File
	Command	CICS/VS Command
	Main Storage Access	Storage Display
CMS	CMS Stack Input	Alternate Input
	CMS FILEDEF I/O	QSAM
	CMS VSAM	VSAM
VSPC	APL Format	APL Data File
	EBCDIC Format	EBCDIC Data File
	Workspace Access	Storage Display

References to "distributed workspaces" have been changed to "workspaces," to avoid any confusion with the concept of distributed systems.

WORKSPACES

Specification Change

Several new workspaces have been added to VS APL, and some previously provided workspaces have been removed. The list of workspaces provided has been revised accordingly.

RELEASE 3, AUGUST 1978

VS APL SUPPORT FOR CICS

Under Release 3 of VS APL, support is now provided for the CICS/DOS/VS and CICS/OS/VS (VS1 and MVS) environments (in addition to the CMS and VSPC environments) as follows:

- CICS executor provides environment-dependent services for interpreter/translator.
- CICS shared storage manager (an integral component of CICS) manages communication between interpreter/translator and auxiliary processors.
- Extension of current auxiliary processor (command auxiliary processor, VSAM/ISAM file auxiliary processor, APL format auxiliary processor, and the full screen manager auxiliary processor) support to CICS.
- Addition of four new auxiliary processors (main storage access auxiliary processor, DL/I access auxiliary processor, transient data auxiliary processor, and the alternate input auxiliary processor) for the CICS environment.
- Addition of a new CICS APL library service program facilitates conversion of libraries and functions.
- Addition of three new distributed workspaces (DL/I support workspace, file support workspace, and an administrative workspace) for the CICS environment.

RELEASE 2, SEPTEMBER 1976

VSAM SUPPORT UNDER CMS

Under Release 2 of VS APL, support is now provided for VSAM when using CMS auxiliary processors.

DOS/VS SUPPORT FOR VSPC

New modules have been added to allow VSPC to run under DOS/VS. These new modules are similar in function to those for OS/VS VSPC. Modules in OS/VS are prefixed by the letters APLO; the new DOS/VS modules are prefixed by the letters APLD. Thus, unless explicitly indicated otherwise, modules indicated in this publication as beginning with APLO should be interpreted as if they began with APLD when working with VSPC under DOS/VS.

VS APL ASSIST

The "Diagnostic Aids" section explains how to handle possible problems with the VS APL Assist.

MAINTENANCE

A number of technical errors have been corrected in this edition. The "Program Organization" and the "Data Areas" sections have been updated considerably.

CONTENTS

FIGURES

SECTION 1. INTRODUCTION

The VS APL processor is an interactive program product that runs under the following systems:

- Customer Information Control System (CICS/VS)
- Conversational Monitor System (CMS)
- Time Sharing Option (TSO)
- Virtual Storage Personal Computing System (VSPC)

VS APL analyzes, stores, and executes source statements written in the VS APL language. In addition, it provides a facility for converting various workspaces to VS APL format.

VS APL PROCESSOR OVERVIEW

The VS APL processor consists of the following components:

- The translator
- The interpreter: exarch and appendage routines
- Four executors—VS APL CICS/VS, VS APL CMS, VS APL TSO, and VS APL VSPC
- The CMS/TSO shared storage manager
- The CICS/VS shared storage manager
- Auxiliary processors
- The APL Service Program Library
- Workspaces
- Workspace libraries
- Cross-system executor services
- VS APL session manager

VS APL Component Functions

The translator analyzes VS APL source statements entered at the terminal, and translates them into internal codes, either building them into defined functions for later execution or passing them immediately to the interpreter for execution.

The interpreter, comprising exarch and appendage routines, scans, analyzes, and executes tokenized statements. Exarch is available either as microcode or as assembler language modules. Appendage routines, available only as assembler language modules, run in conjunction with exarch.

The executor handles initialization of VS APL, and receives control from, and returns control to CICS/VS, CMS, TSO, or VSPC. It also issues supervisor service requests to CICS/VS, CMS, TSO, or VSPC as required by the VS APL processor and handles asynchronous events such as program checks, attention, and other interrupts.

The shared storage manager builds control blocks, sets shared memory, and issues system service requests in association with the shared variable facility of VS APL.

The auxiliary processors provide functions outside of the APL workspace environment by communication with the operating system access methods.

The conversion programs convert APL/360, APL/CMS, and APL Shared Variable (APLSV) workspaces to the VS APL format as required by CICS/VS, CMS, TSO, or VSPC.

In addition to these general conversion utilities, a CICS/VS-only library service program uses conversion output to import the converted workspaces from APLSV, APL/CMS, or APL/360 to VS APL format; a TSO internal APL file service program manages the import and export of APL object files to and from the APL user's TSO system; and a TSO converted workspace import program processes output from APL converted programs, and imports loadable workspaces for TSO.

Certain workspaces are provided with VS APL to aid the user in migration from APL/360, to help him in learning VS APL, and to do certain commonly-needed functions. They are tools to assist users in the use of VS APL.

The cross-system executor services represent a set of components which provides equivalent services to the CMS, TSO, or CICS/VS executor.

The VS APL session manager (optionally available to the APL user) provides common session support, for use with terminals of the IBM 3270 Information Display System under CMS, TSO, or CICS/VS.

VS APL Environment

UNDER CICS/VS: VS APL runs as a series of CICS/VS transactions. The following is a list of transactions by transaction ID.

- APL Specifies the APL user sign-on transaction
- APLU Specifies the user session transaction
- APLL Specifies the library access transaction
- APLT Specifies the non-GDDM terminal I/O transaction
- APLH Specifies the hardcopy processing transaction
- APLO Specifies the auxiliary processor 100 transaction
- APLX Specifies the GDDM terminal I/O transaction

Note that although these default transaction IDs are used throughout this book, an installation can define different transaction IDs.

UNDER CMS: The VS APL translator, interpreter, executor, and shared storage manager run as a CMS module.

UNDER TSO: The VS APL translator, interpreter, and executor run as a TSO command processor.

UNDER VSPC: The VS APL translator, interpreter, and executor run as a VSPC foreground processor.

PURPOSE AND FUNCTION OF THE VS APL PROCESSOR

Translator

The translator receives VS APL source statements as input, directs system commands to the proper routines, converts VS APL source statements to internal codes (tokens), and builds VS APL functions as required. The functions of the translator are to:

- Initialize the user's workspace
- Receive terminal input and determine its type and destination within the processor
- Prepare VS APL statements for execution
- Isolate and execute system commands
- Perform sequencing and control functions for the processor

The translator is divided into the following modules:

- Initialize workspace: APLITINI
- Input/output: APLITINP
- System commands: APLITCMC, APLITCMD, APLITCME, APLITCMF, APLITCMG, APLITCMI, APLITCML, APLITCMS, APLITCMT, APLITCPI, APLITCPO.
- Statement conversion: APLITFUN, APLITIDS, APLITLXS, APLITNCV, APLITPRL.
- Function definition: APLITFDC, APLITFDE, APLITFDN, APLITFDO, APLITHDR.
- Execution control: APLITERR, APLITEX.
- Subroutines: APLITFCH, APLITSUB, APLITUSG.
- Message text and default workspace values: APLITMSG
- Mark end of load module: APLITIHI
- Copyright statement: APLCOIBM

Interpreter

The interpreter receives tokenized VS APL statements as input. Its functions are to:

- Receive control from the translator; return control when input is exhausted or when a translator service is required
- Scan, analyze, and execute tokenized statements
- Format terminal output; request executor output
- Communicate with the shared storage manager when a shared variable is encountered

The interpreter is divided into the following modules:

- Exarch: APLIECMX, APLIEFCH, APLIEFNM, APLIEIDX, APLIEMND, APLIEPSI, APLIEREV, APLIERHO, APLIESCA, APLIESPA, APLIETAK, APLIEXAR, APLIEXFR.

- Appendage Routines: APLIACHK, APLIACIR, APLIADEC, APLIADOM, APLIAENC, APLIAFOR, APLIAGFM, APLIAGOU, APLIAGRD, APLIANAM, APLIAPRD, APLIAQFN, APLIARED, APLIAROT, APLIASCN, APLIASHF, APLIASHV, APLIASYV, APLIATAK, APLIATBC, APLIATRN, APLIATRS, APLIATSP.

Executor

The executor is used for communication between the VS APL processor and the CICS/VS, CMS, TSO, or VSPC system. Such services include terminal input and output and access to libraries. The individual executor module configurations differ from one another, depending on the system under which the processor is operating, but all four executors perform similar functions. These are:

- Establish the VS APL processor environment
- Manage asynchronous events; for example, attention signal from the terminal
- Execute VS APL processor service requests, including terminal I/O requests

The executors are divided into the following modules:

- **CICS/VS**: APLKADEF, APLKADSP, APLKAGBL, APLKAHST, APLKASON, APLKASTB, APLKDOPS, APLKEHCP, APLKEMGR, APLKLIBR, APLKAMIX, APLFXIIM, APLKIFIX, APLKISVI, APLKLIBA, APLKLIBB, APLKLIBC, APLKLIBF, APLKLIBG, APLKLIBU, APLKLIBV, APLKLTAB, APLKMSCA, APLKMSCB, APLKSSVP, APLKSSUB, APLKTCTL, APLKTRAN, APLKTREQ, APLKTRQO, APLKTSRV, APLKVOPS
- **CMS**: APLSCERR, APLSCFXI, APLSCINI, APLSCLIB, APLSCOPY, APLSCTIO, APLSCDAC, APLSCSSI, APLSCMSG, APLSCMSC, APLSCSHV, APLSCSVI, APLSCTBL, APLSCTYP, APLSCDPY
- **TSO**: APLYUCMD, APLYUDOC, APLYUDPY, APLYUERR, APLYUFXI, APLYUEXC, APLYUHSH, APLYUIIM, APLYUINI, APLYULIB, APLYULNE, APLYUMSC, APLYUMSG, APLYUOPT, APLYUPFK, APLYURVC, APLYUSCN, APLYUSHS, APLYUSHV, APLYUSSH, APLYUSVI, APLYUTIO, APLYUTRN, APLYUTYP, APLYUUSR, APLYUTBL
- **VSPC**: APLPAPAB, APLPAPCD, APLPAPFS APLPAPGB, APLPAPGC, APLPAPGD, APLPCOAP, APLPCOEX, APLPCTBL, APLPFXIM, APLPLIBS, APLPMISC, APLPSERR, APLPSHVR, APLPTYIO

Cross-System Executor Services

These services comprise the following components:

- GDDM Interface Services (GDDX)—provides a set of services for communication with the Graphic Data Display Manager (GDDM) when it is used in the session. GDDX is made up of the following modules: APLXGCOM (common), APLXGCHC (common), APLXGCAT (common), APLXGS (CMS), APLXGY (TSO), APLXGKU (CICS/VS), APLXGKT (CICS/VS), APLXGKR (CICS/VS), APLXGKRQ (CICS/VS), APLXGKRR (CICS/VS), and APLXGKON (CICS/VS).
- Main Storage Services—provides the calling routine with a system-independent interface for requesting GETMAIN and FREEMAIN services. There are three separate modules: APLXMYSG (TSO), APLXMSSG (CMS), and APLXMKSG (CICS/VS).
- Stack Management Services—provides a cross-module workstack facility which performs register saving and supplies module-level work areas. The module is APLXSTAK.

- APL Print Services—provides APL print (open, write, and close) support for CMS/TSO, and acts as an APL print support interface for CICS/VS. There are three separate modules: APLXPK (CICS/VS), APLXPS (CMS), and APLXPY (TSO).

- File System Services—processes file processing requests for the auxiliary processor AP 121 and the scrolling code in the executor. There are three separate modules: APLXFYFL (TSO), APLXFSFL (CMS), and APLXFKFL (CICS/VS).

- Common AP Services—provides a set of services between an auxiliary processor and the shared storage manager with a system-independent interface. There are four modules: APLXAC (CMS/TSO), APLXAK (CICS/VS), APLXASD (CMS) and APLXAYD (TSO).

- Wait-Post Services—provides wait-post services to CMS and TSO executor tasks, and acts as a system-independent interface to CICS/VS executor processes. There are three separate entry points: APLXWYWP (TSO), APLXWSWP (CMS), and APLXWKWP (CICS/VS).

- Abend Services—allows CMS, TSO, and CICS/VS tasks to attempt to recover from abends and program checks. There are three separate entry points: APLXBYAB (TSO), APLXBSAB (CMS), and APLXBKAB (CICS/VS).

- Dump Services—provides for caller-selected areas of storage to be printed to a particular destination in a CMS, TSO, or CICS/VS environment. There are two separate entry points: APLXDUMP (CMS and TSO) and APLXDKMP (CICS/VS).

- Translation Services—provides various supported translation services, as well as descriptions of request blocks for translation. The module is APLXTRAN.

- Conversion Services—provides data type conversions for numeric objects. The module is APLXVERS.

VS APL Session Manager

The VS APL session manager comprises the following executable modules which are used to process terminal tables requests from the CICS/VS, CMS, or TSO executor, or from an auxiliary processor:

APLACCBE, APLACDSL, APLACHLP, APLACNDP, APLACMSG, APLACMDX, APLACOPY, APLACPRM, APLACPRO, APLACQRY, APLACQUE, APLACRDA, APLACRSA, APLACSF, APLACXCM, APLADMSG, APLADTTM, APLAK, APLAKP, APLALINE, APLAS, APLASA, APLASP, APLAUSRX, APLAY, APLAYA, APLAYP

CMS/TSO Shared Storage Manager

For VS APL under CMS and TSO, the shared storage manager is logically a part of the respective executor. (In the case of VS APL under VSPC, it forms an integral component of VSPC.) The shared storage manager's principal function is to manage communication between the interpreter/translator and the auxiliary processors. The tasks performed are:

- Initialization for shared variable processing

- Processing of shared variable commands

- Termination of processing when the shared variable facility is no longer required

A common set of shared storage manager modules is employed for the CMS and TSO executors.

- APLSHACC, APLSHBPB, APLSHBVB, APLSHCPY, APLSHGET, APLSHOFR, APLSHPUT, APLSHQUE, APLSHREF, APLSHRET, APLSHSOF, APLSHSON, APLSHSPC, APLSHSRD, APLSHSUB

CICS/VS Shared Storage Manager

The shared storage manager for the CICS/VS executor is logically a part of the executor. It is composed of two modules.

- APLKSSUB, APLKSSVP

Auxiliary Processors

Auxiliary processors are non-APL programs that operate outside the APL environment. The auxiliary processor concept provides a method of extending the capability of the APL environment.

UNDER CICS/VS: Auxiliary processors provide selected data management services for APL files, VSAM and ISAM files, and DL/I data bases; allow a user to request a subset of CICS/VS services, display certain areas of main storage, read/write data in CICS/VS transient data destinations, specify an APL command or statement, provide for application control of the IBM 3270 display facilities, and to display alphameric and graphic data (including color and extended highlighting) via the graphic data display manager (GDDM). These auxiliary processors are:

Auxiliary Processors	Modules
AP 100 (CICS/VS Command)	APL100K, APL100KU, APL100KO
AP 102 (Storage Display)	APL102K
AP 120 (VS APL Session Manager Command)	APL120
AP 121 (APL Data File)	APL121K
AP 123 (VSAM)	APL123K
AP 124 (Full Screen Management)	APL124K, APL124KO
AP 125 (DL/I)	APL125K, APL125KD, APL125KO
AP 126 (GDDM)	APL126, APL126T
AP 132 (Transient Data)	APL132K
AP 139 (Alternate Input)	APL139K

UNDER CMS: Auxiliary processors provide selected data management services for CMS files, VSAM files, and OS files supported by QSAM. They also allow an APL application to specify terminal input data, to pass commands to CP or CMS, to specify an APL command or an APL statement that will be executed when terminal input is next requested, and to display alphameric and graphic data (including color and extended highlighting) via the graphic data display manager (GDDM). These auxiliary processors are:

Auxiliary Processors	Modules
AP 100 (CMS Command)	APL100
AP 101 (Alternate Input)	APL101
AP 110 (CMS File)	APL110
AP 111 (QSAM)	APL111
AP 120 (VS APL Session Manager Command)	APL120
AP 121 (APL Data File)	APL121
AP 123 (VSAM)	APL123
AP 126 (GDDM)	APL126

UNDER TSO: Auxiliary processors provide selected data management services for VSAM files, OS files supported by QSAM, and unkeyed, relative record, fixed-length files supported by BDAM. Thay also allow an APL application to specify an APL command or an APL statement that will be executed when terminal input is next requested, to issue TSO interactive commands, and to display alphameric and graphic data (including color and extended highlighting) via the graphic data display manager (GDDM). These auxiliary processors are:

Auxiliary Processors	Modules
AP 100 (TSO Command)	APLYU100
AP 101 (Alternate Input)	APLYU101
AP 102 (Storage Display)	APLYU102
AP 111 (QSAM)	APLYU111
AP 120 (VS APL Session Manager)	APL120
AP 121 (APL Data File)	APL121
AP 123 (VSAM)	APL123
AP 126 (GDDM)	APL126, APL126T
AP 210 (BDAM)	APLYU210

UNDER VSPC: Auxiliary processors provide selected data management services for VSPC library files and VSAM files maintained by the operating system, and provide for application control of the IBM 3270 display facilities. Under VSPC, the auxiliary processors are contained within the executor, and operate through modules APLPAPAB, APLPAPCD, APLPAPFS, APLPAPGB, APLPAPGC, APLPAPGD, APLPCOAP, and APLP126T. The auxiliary processors are:

Auxiliary Processors	Modules
AP 100 (VSPC Command)	APLPAPAB
AP 101 (Alternate Input)	APLPAPAB
AP 102 (Storage Display)	APLPAPAB
AP 121 (APL Data File)	APLPAPAB, APLPAPCD
AP 122 (EBCDIC Data File)	APLPAPAB, APLPAPCD
AP 123 (VSAM)	APLPAPAB, APLPAPCD
AP 124 (Full Screen Management)	APLPAPAB, APLPAPFS
AP 126 (GDDM)	APLPAPAB, APLPAPGB, APLPAPGC, APLPAPGD, APLP126T

APL Service Program Library

THE CONVERSION PROGRAMS: These members of the service program library construct VS APL workspaces from APL/360, APLSV, and APL/CMS dump tapes for CICS/VS, CMS, TSO or VSPC. They also provide user profile and directory information for VSPC.

The configuration of the conversion programs is as follows:

- **CMS (APL/360 and APLSV)**: APLCCULL, APLCDISP, APLCFUNC, APLCGRUP, APLCIBNM, APLCINIT, APLCLEAR, APLCMISC, APLCPARM, APLCRPRT, APLCSAVE, APLCSHIP, APLCSPIE, APLCTBCD, APLCVARB, APLCWKSP, APLCWSFN.

- **CMS (APL/CMS)**: APLQDISP, APLQFUNC, APLQGRUP, APLQIBNM, APLQINIT, APLQLEAR, APLQMISC, APLQPARM, APLQRPRT, APLQSAVE, APLQVARB, APLQWKSP, APLQSPIE.

- **CICS/VS, TSO, VSPC (OS/VS1 and OS/VS2)**: APLOCULL, APLODIRE, APLODISP, APLOFUNC, APLOGRUP, APLOIBNM, APLOINIT, APLOLEAR, APLOMISC, APLOPARM, APLORPRT, APLOSAVE, APLOSHIP, APLOSLST, APLOSPIE, APLOTBCD, APLOTIDY, APLOVARB, APLOWKSP, APLOWSFN.

- **CICS/VS (DOS/VS)**: APLDCULL, APLDDIRE, APLDDISP, APLDFUNC, APLDGRUP, APLDIBNM, APLDINIT, APLDLEAR, APLDMISC, APLDPARM, APLDRPRT, APLDSAVE, APLDSHIP, APLDSLST, APLDSPIE, APLDTIDY, APLDTBCD, APLDVARB, APLDWKSP, APLDWSFN.

In addition to the above modules, each version of the conversion program also contains these translator and interpreter modules: APLIEREV, APLIESPA, APLITFDC, APLITHDR, APLITIDS, APLITLXS, APLITNCV and APLCOIBM.

When used with CICS/VS, TSO, or VSPC, the conversion program runs in the batch environment of the host operating system (OS/VS1, OS/VS2, or DOS/VS). Under CMS, it runs as a separate program invoked from a CMS EXEC procedure and under control of the CMS nucleus.

OTHER SERVICE PROGRAMS: Other members of the service program library are the following:

FOR CICS/VS: An APL library service program imports and exports workspace and auxiliary processor 121 files, copies APL user libraries and initializes APL data sets during CICS/VS installation. This program comprises the following modules:

APLKDALD, APLKDAUT, APLKDCMD, APLKDCPY, APLKDDOS, APLKDDSI, APLKDDSO, APLKDEXP, APLKDIMP, APLKDINT, APLKDLBI, APLKDLBO, APLKDMSG, APLKDPIN, APLKDSPG, APLKDTPO, APLKDTRM, APLKDSCN, APLKDEXC, APLKDFMT, APLKVALD, APLKVCMD, APLKVOPI, APLKVDSI, APLVDSO, APLKVEXP, APLKVIMP, APLKVINT, APLKVLBI, APLKVLBO, APLKVMSG, APLKVPIN, APLKVSPG, APLKVTPO, APLKVTRM, APLKVSCN, APLKVEXC, APLKVFMT

FOR TSO: A workspace manages importing (addition) and exporting (off loading) of APL objects to and from the APL user's library under TSO. The workspace, WSINFO, contains additional information on this workspace.

In addition to the workspace, there is an APL TSO converted workspace import program which processes output from APL converted programs and imports loadable workspaces for the TSO system. This single load module is invoked as a batch job; its name is APLYUCNV.

Workspaces

The environment for VS APL is established by an area of storage called a workspace. The size of the workspace is determined by the installation and the limits of the host system (CICS/VS, CMS, TSO, or VSPC). The workspace contains the user's programs, the values of variables, the user status, and the current input to or output from the interpreter. The workspace, therefore, is the means of communication between the executor, the translator, and the interpreter.

Workspace Libraries

Three libraries of workspaces are provided with VS/APL, as follows:

- Library 1: (workspaces of general usefulness for all systems)
  - WSINFO—summary of all workspaces.
- Library 2: (auxiliary processor workspaces)
- Library 314159: (special workspaces—CICS/VS only)
  - ADMIN—monitors and controls use of the APL system under CICS/VS, and maintains the APL directory.

Each of these workspaces has three functions or variables that describe what it contains and how it is used. They are:

- ABSTRACT—brief description of workspace contents.
- DESCRIBE—what the workspace contains, in detail.
- HOW—how to use the workspace.

PHYSICAL CHARACTERISTICS OF THE VS APL PROCESSOR

Object Modules

The VS APL processor is distributed in the form of separate object modules as described under "Purpose and Function of the VS APL Processor" in this section.

Load Modules

UNDER CICS/VS: VS APL (except for the library service and conversion programs) is stored in the CICS/VS load library as a set of independent load modules. Each load module is identified to CICS/VS by an entry in the CICS/VS processing program table (PPT).

The following load modules are built from multiple source modules:

APLINTRP contains the interpreter modules (APLIxxxx), APLFXIIM and APLCOIBM

APLKADSP contains the CICS/VS executor modules APLKAMIX, APLASCHD APLKADSP, APLKIFIX, APLKLIBC, APLKLIBU, APLKMSCA, APLKMSCB, APLKISVI, APLXGCHC, APLXGCOM, APLXGKON, APLACRCP, APLACCBE, APLACDSL, APLCCHLP, APLACMDX, APLACMSG, APLACNDP, APLACOPY, APLACPRM, APLACPRO, APLACQRY, APLACQUE, APLACRDA, APLACRSA, APLACSF, APLACXCM, APLADMSG, APLADTTM, APLAK, APLAKP, APLALINE and APLAUSRX

APLKASON contains APLKMIX and APLKASON

APLKLIBG contains APLKLIBA, APLKLIBG, APLKLIBV and APLKLTAB

APLKEHCP contains APLKEHCP and APLKTRAN (also included in the APLKASTB load module)

APLKTCTL contains APLKTCTL and APLKTCWR

APLKSPRG contains the library service program modules listed under the section entitled "Purpose and Function of the VS APL Processor." Note that modules beginning with APLKV are in a load module for OS/VS only, and that modules beginning with APLKD are in a load module for DOS/VS only.

APLXGKT contains APLXGKT and APLAKP

APL100K contains APL100K and APL100KU

APL120 contains APL120, APLASCHD, APLAK and APLAKP

APL124K contains APL124K and APL124KO

APL125K contains APL125K and APL125KD (or APL125KV)

APL126 contains APL126, APL126T

APLKASTB contains APLKASTB, APLKAGBL, APLKDOPS (or APLKVOPS), APLKEMGR, APLKLIBB, APLKLIBR, APLKLIBF, APLKSSUB, APLKSSVP, APLKTRAN, APLKTREQ, APLKTRQO, APLKTSRV, APLXAK, APLXDKMP, APLXFKFL, APLXMKSG, APLXSTAK, APLXTRAN, APLXVERS, APLXGKU, APLXPK, APLASCHD, APLAKP, APLXGKRQ, APLXGKR, and APLXGKRR

All other modules are stored as separate load modules (APLKADEF, APLKAHST, APL100KO, APL102K, APL121K, APL123K, APL132K, APL139K, and APLKPARM).

UNDER CMS: The executor, translator, interpreter, and shared storage manager object modules are link-edited and loaded as one load module (VSAPL). Optionally, auxiliary processors may also be included in this load module. A second load module (startup module APL) is generated for discontiguous segment determination.

The conversion program object modules are link-edited and loaded as one load module for each of the conversion programs. The load module names are: APLCVCMS (convert APL/360 and APLSV workspaces under CMS), APLCVRPQ (convert APL/CMS workspaces under CMS), APLCVOS (convert APL/360 and APLSV workspaces under OS/VS1 or OS/VS2), and APLDVDOS (convert APL/360 and APLSV workspaces under DOS/VS).

UNDER TSO: The executor, translator, interpreter, and shared storage manager object modules are link-edited and loaded as one load module with the name VSAPL.

UNDER VSPC: The executor (which includes the auxiliary processors), translator, and interpreter object modules are link-edited and loaded as one load module.

Flow of Control

Flow of control among VS APL modules is determined by the VS APL source statements received at the terminal or contained within the workspace as function definitions that are to be executed.

The major flow of communication between components is shown in Figure 1. The flow of communication to the auxiliary processors and shared storage manager is not applicable. Under VSPC, where the auxiliary processors are contained within the executor, the shared storage manager is a component of the host system (VSPC).

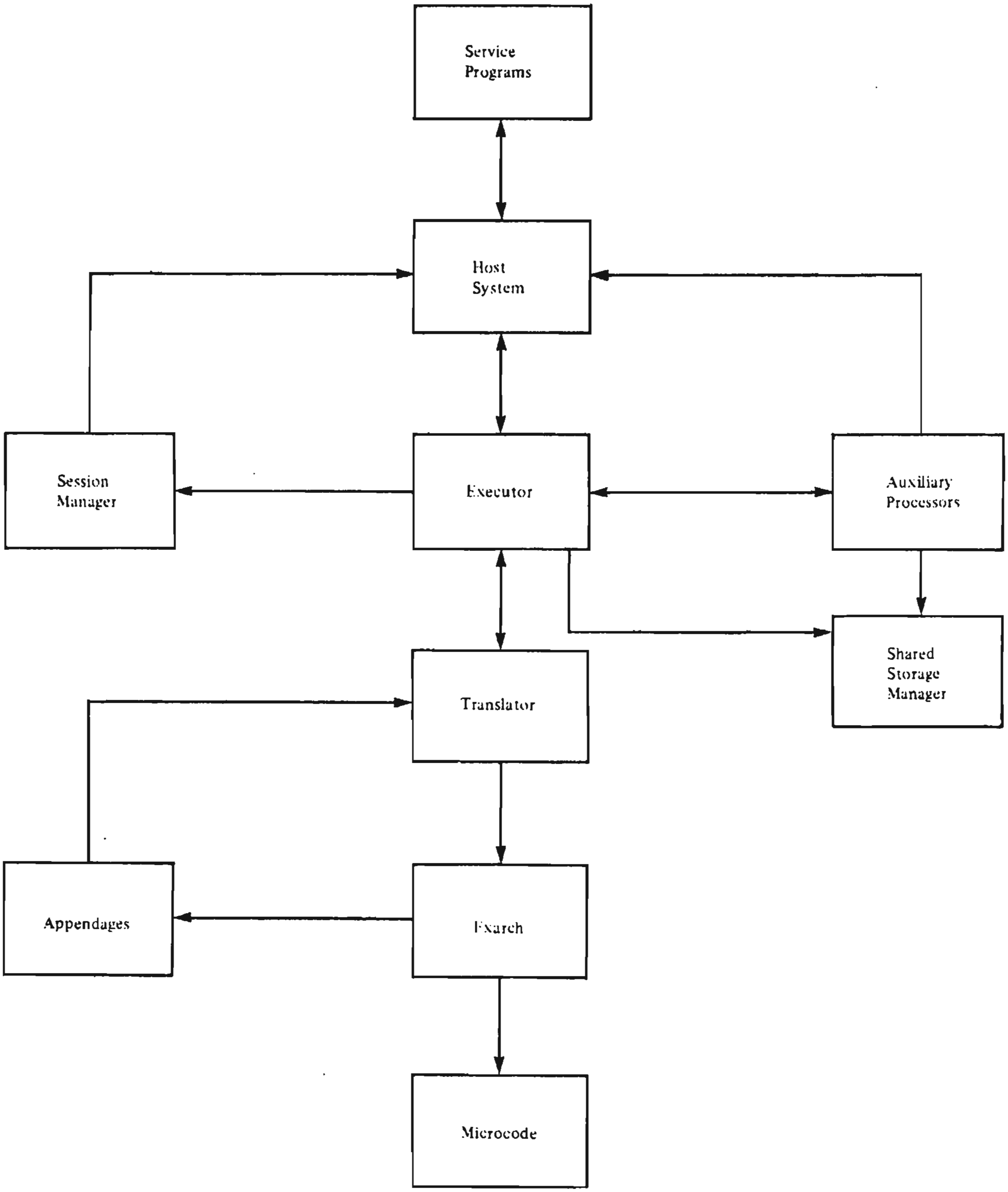


Figure 1. VS APL Processor Communication Overview

OPERATIONAL CONSIDERATIONS

Data Set Information

IN CICS/VS: For DOS/VS, the executor modules reside in the system or private core image library; for OS/VS1, they reside in LINKLIB or in a CICS/VS load libary; for OS/VS2, they reside in LPALIB or in a CICS/VS load libary. User workspaces reside in the APLLIB VSAM entry sequenced data set for DOS/VS, OS/VS1, and OS/VS2.

IN CMS: In CMS, VS APL modules, files, and workspaces exist as individual files on VM mini-disks. If VS APL is used in a discontiguous shared segment (DCSS), then the module images also reside in the CP system storage.

IN TSO: Load modules can reside in LPALIB or in another load libary. User workspaces reside in sequential (BSAM) data sets.

IN VSPC: The processor resides in the VS system library; in OS/VS1 it resides on the SYS1.LINKLIB library; in OS/VS2 it resides in the SYS1.LPALIB library. User workspaces reside on the SYSLIB2 VSAM entry sequenced data set (OS/VS1 and OS/VS2).

Installation

VS APL under CMS, CICS/VS, TSO, or VSPC is installed by standard operating system installation tools. These are SMP (for OS/VS systems), MSHP (for DOS/VS systems), and PLC (for CMS systems).

Control Information

UNDER CICS/VS: The VS APL processor is started either from a terminal or from another transaction.

UNDER CMS: The VS APL processor is entered by means of a command given from the terminal or from an EXEC procedure. The APL initialization routine, after analyzing the command parameters, uses a CMS EXEC called APLEXIT to establish the APL environment. APLEXIT EXEC is invoked again at termination.

UNDER TSO: The VS APL processor is started by a TSO command processor invoked by entering its name (APL) through the terminal or from a CLIST.

UNDER VSPC: The VS APL processor is started at user logon time by the VSPC online program if the user's profile specifies VS APL, or by the "ENTER APL" command issued at a later time during the user's session.

SYSTEM CONFIGURATION

Processors

UNDER CICS/VS: VS APL operates on all compatible processors supported by CICS/VS under DOS/VS, OS/VS1, or MVS.

UNDER CMS: VS APL operates on all compatible processors supported by CMS under the Virtual Machine Facility/370 (VM/370).

UNDER TSO: VS APL operates on all compatible processors supported by TSO under the MVS operating system.

UNDER VSPC: VS APL operates on all compatible processors supported by VSPC under OS/VS1 or MVS.

Access Methods

UNDER CICS/VS: VSAM and SAM are the only required access methods, although access to ISAM files through the CICS/VS interface is also supported. VSAM requirements include control interval processing as well as essentially all of the VSAM support available under CICS/VS.

UNDER CMS: The standard CMS file access macros are used to access CMS files. Access to VSAM is also supported. For a description of these macros see <u>IBM Virtual Machine Facility/370: CP Command Reference for General Users</u>.

UNDER TSO: VS APL employs BSAM, BPAM, VSAM, QSAM, and BDAM files for APL applications.

UNDER VSPC: VS APL uses the VSPC library management function, based on the Virtual Storage Access Method (VSAM), for all library support. It supports all DASD devices supported by VSAM. Auxiliary processors may also provide other access method support.

Terminals

Refer to the following manuals for a description of the terminals supported under VS APL:

- <u>VS APL for CICS/VS: Terminal User's Guide</u>
- <u>VS APL for CMS: Terminal User's Guide</u>
- <u>VS APL for TSO: Terminal User's Guide</u>
- <u>VS APL for VSPC: Terminal User's Guide</u>

Supervisor Service Calls

UNDER CICS/VS: Most APL supervisor services are requested through CICS/VS interfaces. VSAM control interval processing is performed using operating system services directly. In some cases, VS APL uses CICS/VS control blocks and macros that are not a part of the CICS/VS external interface.

UNDER CMS: The VS APL executor routines issue CP and CMS commands; CMS macros, such as DMSFREE and DMSFRET; and simulated OS macros, such as WAIT, POST, STIMER, and STAX. The executor also makes use of some CMS routines whose address constants are found in the CMS NUCON macro. Hexadecimal location 440 in the CMS NUCON macro is reserved for a pointer to the VS APL global table (GLBLTABL).

UNDER TSO: VS APL makes use of the services described in <u>A Guide to Writing a Terminal Monitor and Program Command Processor</u>. The primary TSO services used are DAIR and TGET/TPUT. MVS operating system services are also used.

UNDER VSPC: VS APL makes use of the service calls provided through the defined foreground interface to VSPC. These calls are described in "Method of Operations" (Diagram 1.1: "Communication with VSPC").

ERROR HANDLING

Customer Information Control System (CICS/VS)

UNDER CICS/VS: The integrity of the VS APL user's variables and functions is protected by the VS APL executor itself. Errors of a single user or program errors within a processor cannot interfere with another user. VS APL executor and interpreter routines operate in problem program state.

VS APL under CICS/VS provides an internal dump facility for the user's workspace and the areas associated with it. A dump is requested automatically by the VS APL processor to provide information about certain types of processor-related system errors.

VS APL under CICS/VS intercepts both processor page faults and program checks. Program checks are passed back to the processor to take appropriate action and to issue appropriate diagnostic and error messages.

Conversational Monitor System (CMS)

UNDER CMS: The integrity of VS APL is protected by the Virtual Machine Facility/370 (VM/370), CMS, and the VS APL executor routines. VM/370 ensures that no errors of a single user and no errors of the VS APL interpreter or executor routines can affect any other user.

VS APL executor and interpreter routines operate in the virtual supervisor state. The executor routines provide their own storage protection as well as data protection for non-interpreter routines. Program checks are intercepted by the VS APL executor routines and passed back to the interpreter through the defined interface. This allows the interpreter to issue appropriate diagnostic and error messages.

VS APL executor routines check VM/370 system messages and return codes after issuing system service requests.

A STAE exit is provided to allow dumping of storage for problem determination. The STAE exit stops the virtual machine so that the user can enter CP commands to display storage and help isolate problems.

For noncatastrophic errors, diagnostic information is printed at the user's terminal and the active workspace is cleared.

Time Sharing Option (TSO)

UNDER TSO: The integrity of VS APL is protected by both the Multiple Virtual Storage (MVS) and the VS APL executor routines. MVS ensures that no errors of a single user and no errors of the VS APL interpreter or executor routines can affect any other user.

VS APL employs ESTAE, SPIE, and ATTACH with the ESTAI option to gain control when MVS detects an error. In addition, the auxiliary processors set up the DCB ABEND exits. Program checks are intercepted by the VS APL executor routines and passed back to the interpreter through the defined interface; this allows the interpreter to issue appropriate diagnostic and error messages.

The basic thrust of error recovery in VS APL under TSO is to get the active workspace saved in the CONTINUE workspace, and to cause TSO to reinvoke a clean copy of VS APL which will in turn reload the CONTINUE workspace and continue processing. There are two principal kinds of abends: 1) X22 and X3E abends brought about by operator cancel, timing, TCAM error, etc. In these instances, the CONTINUE workspace is saved normally; 2) all other abends constitute error situations in which the CONTINUE workspace is marked nonloadable.

VS Personal Computing (VSPC)

UNDER VSPC: The integrity of the VS APL user's variables and functions is protected by VSPC itself. Errors of a single user or program errors within a processor cannot interfere with another user.

VSPC provides an internal dump facility for the user's workspace and the areas associated with it. A dump is requested automatically by the VS APL processor to provide information about certain types of processor-related system errors.

VSPC intercepts both processor page faults and program checks. Program checks are passed back to the processor through the defined interface to allow the processor to take appropriate action and issue appropriate diagnostic and error messages.

COMPONENT AND MODULE NAMING CONVENTIONS

Object modules are identified by 5- to 8-character names that describe them by component and function.

Object module names, except for the shared storage manager, conform to the following convention:

- A 3-character prefix of: APL
- Followed by a component identification, described in Figure 2.
- Followed by an abbreviation identifying the function of the module.

Entry point names conform to the same convention as module names, except that, in some cases, the 3-character 'APL' prefix is omitted.

The conversion modules for DOS/VS differ from those for OS/VS. These modules are functionally the same, but the DOS/VS modules are designed to interface with DOS/VS and the OS/VS modules with OS/VS. The OS/VS modules begin with the characters APLO; the DOS/VS modules begin with the characters APLD. To avoid unnecessary repetition in this publication, only the OS/VS names are used in this publication wherever possible. Unless explicitly noted otherwise, substitute the prefix APLD for APLO when using this publication for DOS/VS VS APL.

Identification	Component	
A	Session Manager	
C	Conversion Program (CMS and TSO—APL/360 and APLSV workspaces)	
D	Conversion Program (DOS/VS)	
I	Interpreter	
IA	Appendage Routines	
IE	Exarch	
K	Executor (CICS/VS) with shared storage manager, and library service program	
KD	DOS/VS system-dependent code (CICS/VS DOS/VS)	
KV	OS/VS system-dependent code (CICS/VS OS/VS)	
O	Conversion Program (OS/VS1 and OS/VS2)	
P	Executor (VSPC) with auxiliary processors	
Processor Number (nnn)	Auxiliary Processors (CMS and TSO)	
Processor Number (nnn followed by K)	Auxiliary Processors (CICS/VS)	
Processor Number (nnn)	Auxiliary Processors (Common)	
Q	Conversion Program (CMS—APL/CMS workspaces)	
SC	Executor (CMS)	
SH	CMS and TSO shared storage manager	
X	Common Services	
	XA	Common AP Services
	XB	Common Abend Services
	XD	Common Dump Services
	XF	File System Services
	XG	GDDM Interface Services
	XM	Main Storage Services
	XP	APL Print Services
	XS	Stack Management Services
	XT	Translate Services
	XV	Conversion Services
	XW	Wait Post Services
YU	Executor (TSO)	

Figure 2. Object Module Component Name Identification

In this section, Hierarchy Input Processing Output (HIPO) diagrams are used to describe the functions of VS APL.

HIPO is a method for graphically describing the internal functions of a program without regard for the way in which the functions are implemented or for the physical organization of the program. A HIPO package contains a visual table of components and a set of method of operation diagrams illustrating the functions of a program, in this case, the VS APL processor. The visual table of components (see Figure 3) shows the contents of each diagram and how it is related to the other diagrams in the set. The graphic symbols used in Method of Operation diagrams are identified in Figure 4. The method of operation diagrams are grouped by function.

The method of operation diagrams themselves are divided into four distinct areas of information: input, process, output, and extended description (diagram notes). The input information, on the left side of the diagram, describes the input to the process or function being described. The process information, the central portion of the diagram, describes processes that make up the function. The output information, on the right side of the diagram, illustrates the output from the process. The extended description information following the diagram is used to provide additional detail or to outline how the function was implemented. This section also contains references to the module that performs all or part of the function involved, and any references within the remainder of this publication where additional information may be found.

0.0: VS APL Processor Overview

    1.0*: Host System Communication

        1.1: Communication with VSPC

            1.1.1: Shared Variable Processing (VSPC)

        1.2: Communication with CMS

            1.2.1: Shared Storage Manager (CMS and TSO)

            1.2.2: Auxiliary Processors (CMS)

        1.3: Communication with CICS/VS

            1.3.1: Shared Storage Manager (CICS/VS)

            1.3.2: Auxiliary Processors (CICS/VS)

        1.4: Communication with TSO

            1.4.1: Auxiliary Processors (TSO)

            1.4.2*: Shared Storage Manager (TSO) (see Diagram 1.2.1)

    2.0: Input Recognition, Translation, and Routing

    3.0: Function Definition and Editing

        3.1: Function Editing

        3.2: Function Definition

    4.0: Statement Execution

        4.1: Statement Scan, Syntax Analysis, and Execution

            4.1.1: Function Call and Function Exit Processing

            4.1.2: Branch Processing

            4.1.3: Primitive Function Processing

            4.1.4: Miscellaneous Processing

            4.1.5: Shared Object Processing

        4.2: Return Code Processing

    5.0: System Command Execution

    6.0: Workspace Conversion

* No diagram is provided for this component.

Figure 3 (Part 1 of 2). Table of Components

7.0: CICS/VS Library Service Program

8.0*: Host-Independent Executor Services

8.1*: APL GDDM Interface Services Subcomponent (GDDX)

8.2: VS APL Session Manager Executor Scheduler

8.2.1: VS APL Session Manager Executor Processor

8.3*: Common Auxiliary Processor Services

8.3.1: Common Auxiliary Processor Services Under CMS & TSO

8.3.2: Common Auxiliary Processor Services Under CICS/VS

8.4*: Common Auxiliary Processors

8.4.1: VS APL Session Manager Command auxiliary processor for CICS/VS, CMS, and TSO

8.4.2: GDDM Auxiliary Processor for CICS/VS, CMS, and TSO

8.4.3: APL Data File Auxiliary Processor for CMS and TSO

8.4.4*: VSAM Auxiliary Processor (see Diagram 1.2.2 or 1.4.2)

* No diagram is provided for this component.

Figure 3 (Part 2 of 2). Table of Components

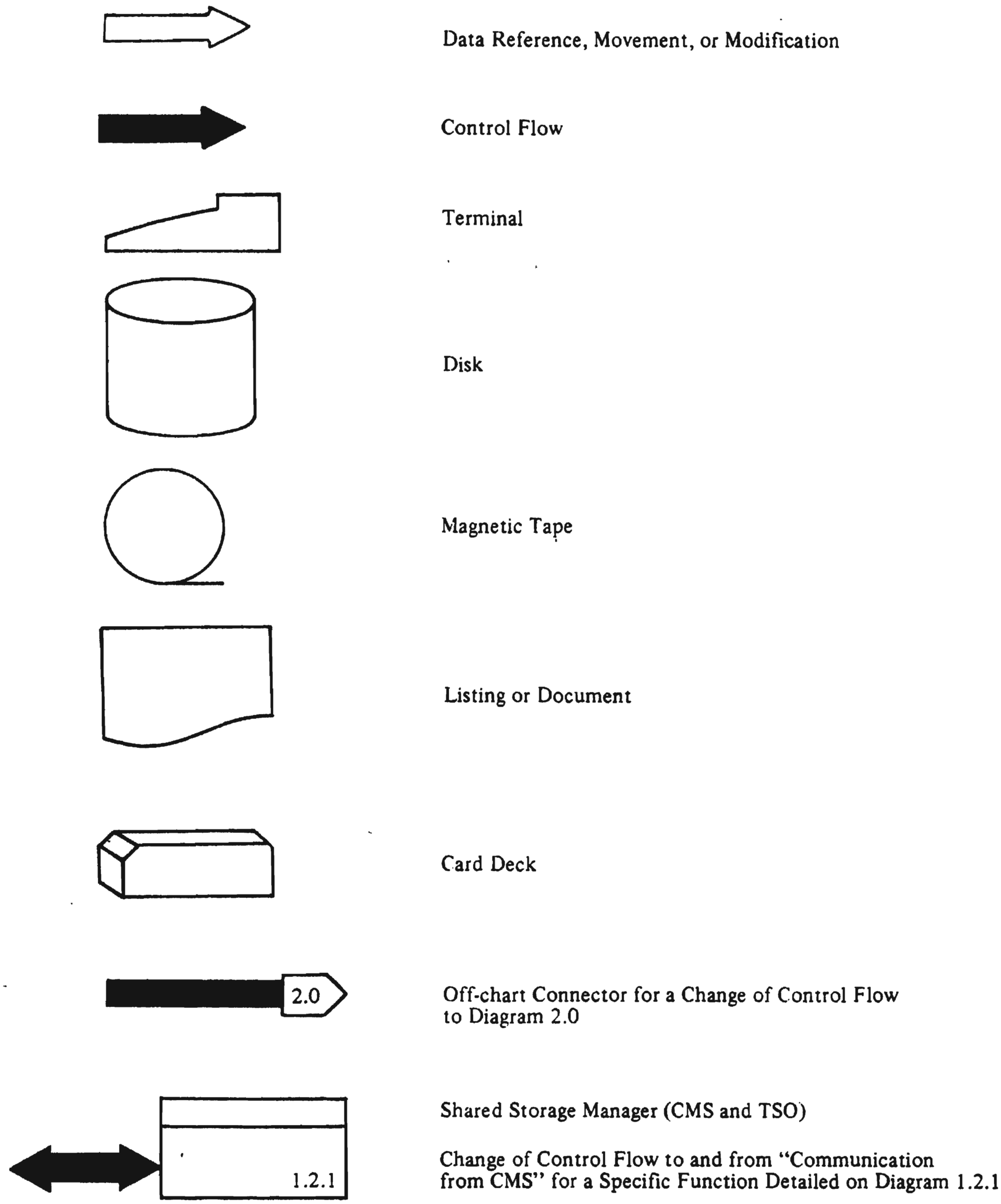
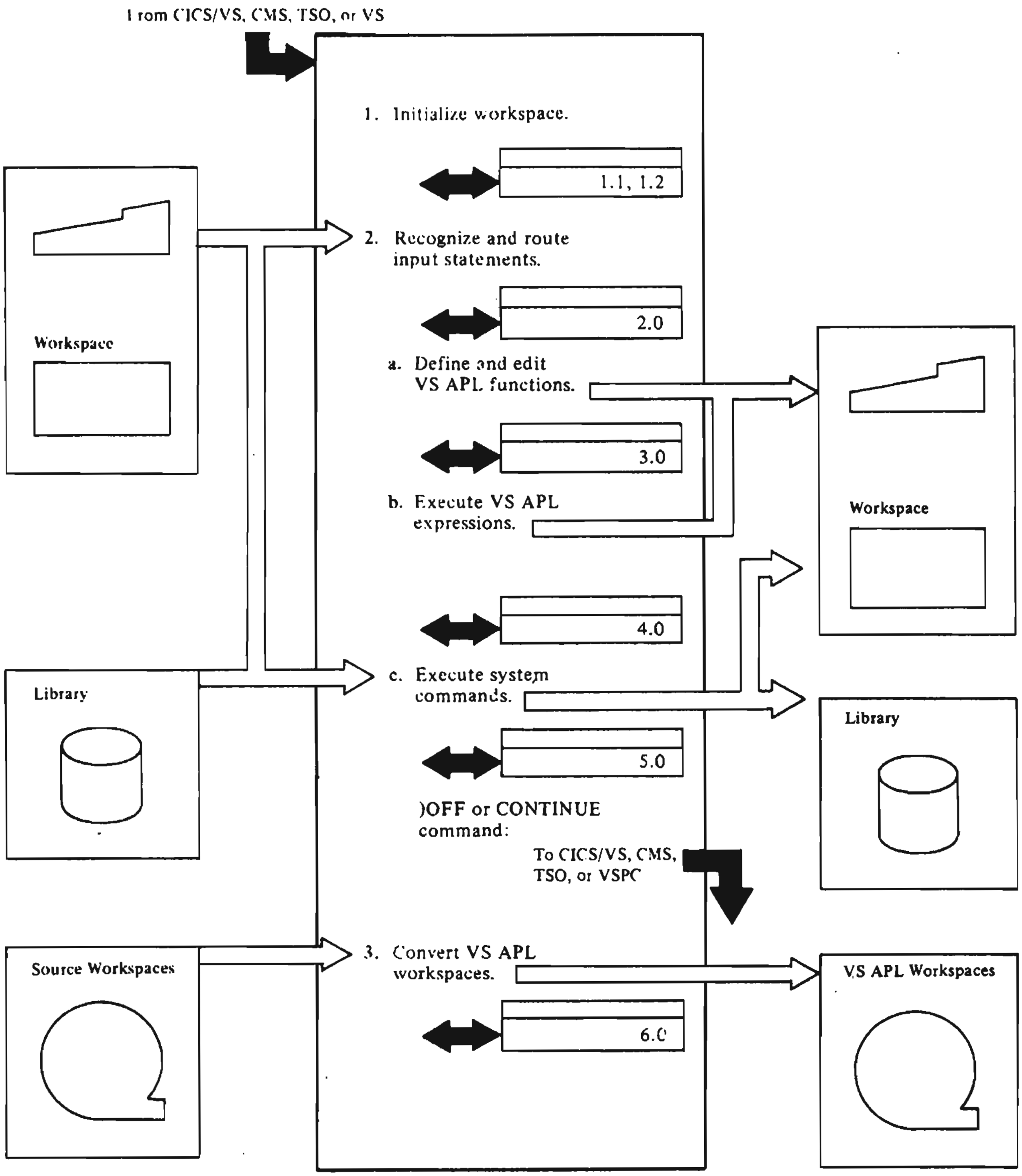


Figure 4. Graphic Symbols Used in Method of Operation Diagrams

DIAGRAM 0.0: VS APL PROCESSOR OVERVIEW



Notes for Diagram 0.0

EXECUTOR

1. When control is received from the host system (VSPC, CICS/VS, TSO, or CMS) the workspace is initialized.

TRANSLATOR

2. Input is received from the terminal. The contents of the input line and the status of the workspace determine the destination of the line. [Executor]

   a. If the first nonblank character in the line is a del, or if the workspace is in function edit status, the line is routed to the function definition and edit routines.

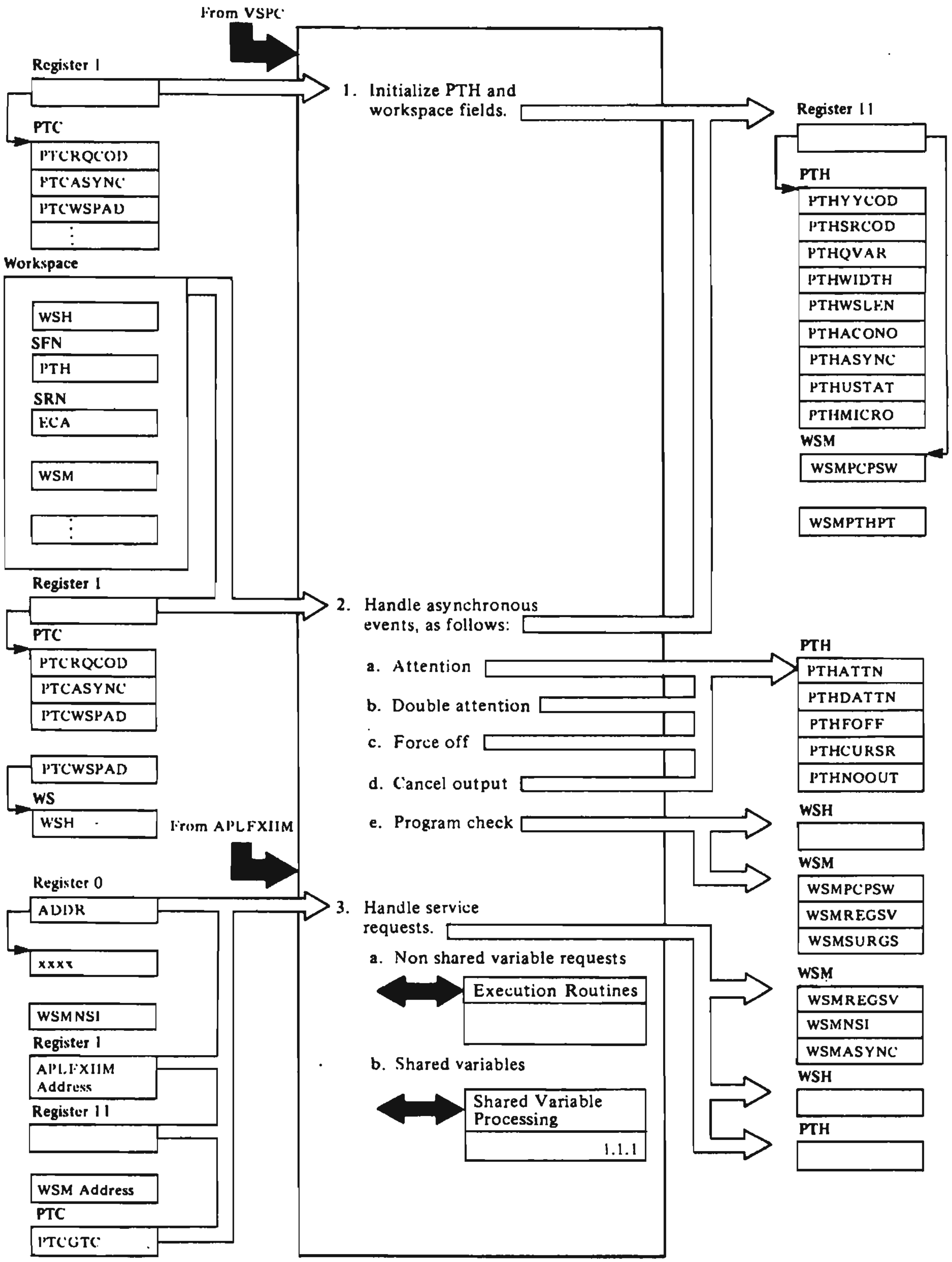
   b. If the workspace is not in function edit status, and the first nonblank character of the line is neither a del nor a right parenthesis, the line is routed to the statement execution routines. [Interpreter]

   c. If the workspace is not in function edit status, and the first nonblank character of the line is a right parenthesis, the line is routed to the command processor routines.

   The above process is continued until an)OFF or)CONTINUE command is input. The control is returned to the host system.

3. The conversion program, run as a batch job, converts source workspaces (APL/360, APLSV, or APL/CMS) to VS APL workspaces.

DIAGRAM 1.1: COMMUNICATION WITH VSPC



Notes for Diagram 1.1

APLPCOEX

1. For initialization, the executor receives a default size workspace from VSPC. The length is indicated in PTCWSLEN and pointed to by PTCWSPAD. The executor takes the top 2K bytes for its own use and always informs the interpreter that the workspace is after this 2K byte area. [APLPCENT]

   The executor, within the 2K byte block, sets fields in the executor control area (ECA); that is, it initializes the ECADUMP field to 0, sets the ECASTAT field to indicate that the executor has been called, sets the ECAPTC field to the address of the PTC area, and initializes ECAMICRO for microcode assist.

   The executor then initializes fields of the PTH and sets the WSMPTHPT field to the address of the PTH. It initializes WSMPCPSW=0. It then places the service request YYON in PTHYYCOD and passes control to the interpreter at its entry point APLIINIT.

2. For asynchronous event handling, the executor receives an indication from VSPC, determines the type of event, and processes it as follows:

   a. Attention—sets PTHATTN on, except if already on, then sets PTHDATTN on, sets type element to zero position, PTHCURSR field to 0, and WSHPFLG1.WSHPATN to 1. Sets WSMASYNC fields correspondingly and returns to VSPC to be dispatched at the point of interrupt.

   b. Double attention—sets type element to zero position, PTHATTN.PTHDATTN bits to 1, WSHPFLG1.WSHPDATN bit to 1, and PTHCURSR field to 0. Sets WSMASYNC fields correspondingly and returns to VSPC to be dispatched at the point of interrupt.

   c. Force off—sets PTHFOFF bit to 1 for logoff by the interpreter and sets WSHPFLG1.WSHPSTRM to 1. Sets WSMASYNC fields correspondingly and returns to VSPC to be dispatched at the point of interrupt.

   d. Cancel output—sets PTHNOOUT on, sets WSMSASYNC fields, PTHCURSR field to 0, and returns to VSPC to be redispatched at the point of interrupt, with WSHPFLG1.WSHPCNCL=1.

   e. Program check—saves registers in WSMSURGS field and PSW in WSHPSWSV field. For interpreter program check, registers and the PSW are moved from the WSHREGSV and WSHPSWSV fields, respectively, to the WSMREGSV and WSMPCPSW fields, respectively. The program check is acknowledged (WSHPFLG1.WSHPPCHK=1), the YYPRGX command is simulated, and control is passed to the APLIINIT routine (the interpreter). (See Diagram 2.0: "Input Recognition, Translation, and Routing.") For microcode assist initialization error, when microcode is not installed, the program check is acknowledged, the PTHMICRO bit is set, and control is returned to VSPC for redispatch at the point of interrupt. For executor program checks or program check loops in the interpreter, messages are issued, a dump is taken, and the WENDR error exit is taken.

APLPFXIM

3. For service requests, addressability to the PTH, WSH, SFN, and ECA is set up by backing up 2K bytes from the address of the WSM. The interpreter's registers are saved (except for YYDUMP request), the address of the next sequential instruction is saved in WSMNSI, and the request code is entered in the PTHYYCOD field. Control is then passed to the appropriate request handling routine. [APLFXIIM]

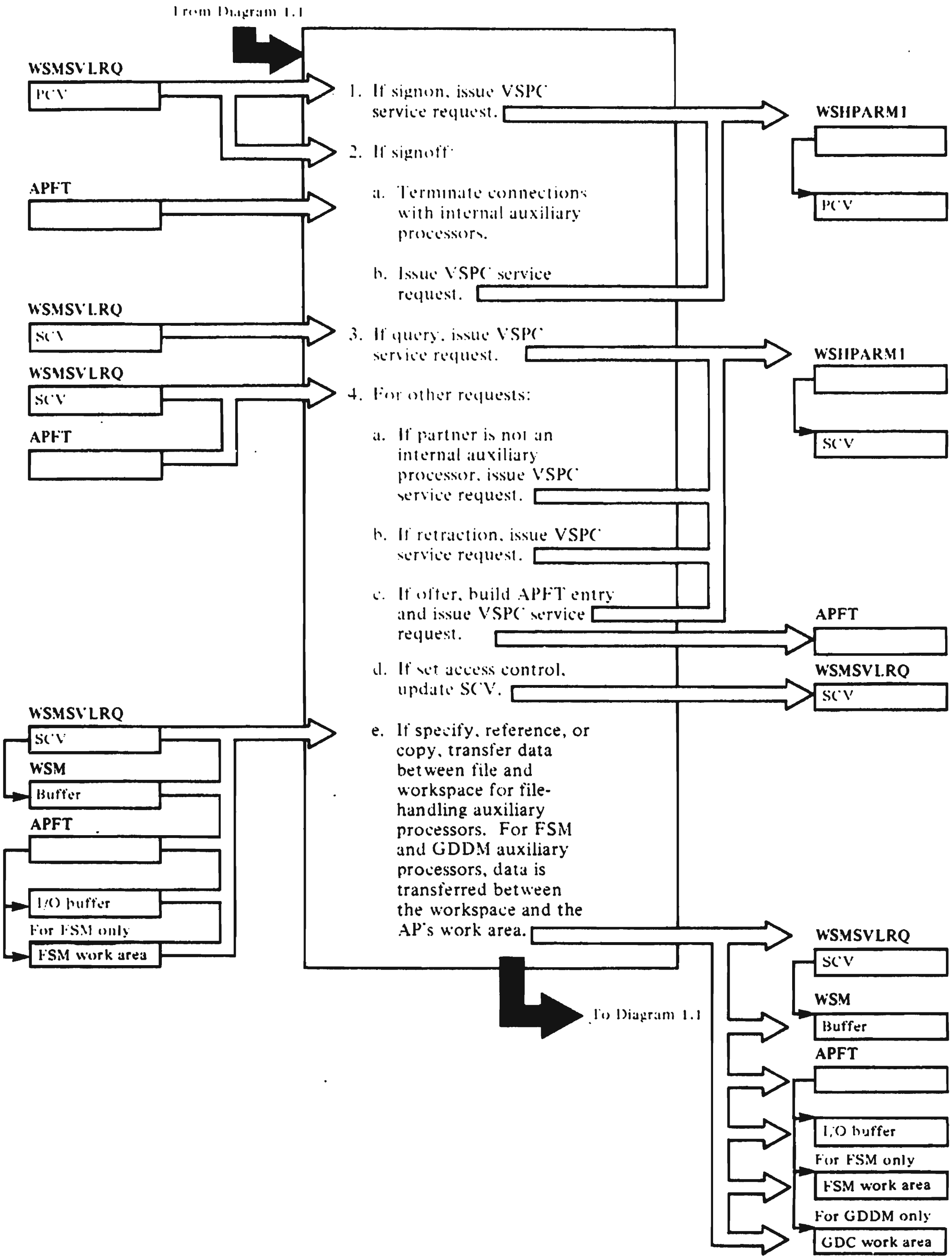
   The execution routine returns with the service request return code set in PTHSRCOD. Control is passed to the interpreter at its entry point, APLIINIT, where the interpreter's environment is restored and control is returned to the instruction following the service request.

   For a description of the service request codes and the names of the VSPC executor routines that handle them, see "Values, Parameters, and Return Codes for

Service Requests" under "Service Request Calls" in "Section 6. Diagnostic Aids."

For shared variable processing, control is passed to APLPSHVR to route the request to the VSPC shared storage manager or to the internal auxiliary processors. (See Diagram 1.1.1: "Shared Variable Processing (VSPC).")

DIAGRAM 1.1.1: SHARED VARIABLE PROCESSING (VSPC)



Notes for Diagram 1.1.1

Return and reason codes for each request are passed to the interpreter in PTHSRCOD.

APLPSHVR

1. For sign-on, the user's ID, shared variable quota, and space quota are placed in the PCV. A VSPC service request SSON is issued. [PCSON]

APLPAPAB

2. For sign-off, each active APFT entry is cleared. If the VSAM file is open, a VSPC service request VCLOSE is issued. If FSM was active, a TFSCRN EXIT request is issued. If GDDM was active, GDDMSOFF is called. [APLPAPSF]

APLPSHVR

The user's ID is placed in the PCV, and a VSPC service request SSOF is issued. [PCSOFF]

APLPSHVR

3. For a query, the user's ID is placed in the SCV, and a VSPC service request SQRY is issued. [PCSQUERY]

APLPSHVR

4. For set access control, copy, reference, retract, or specify, the APFT entries are searched to determine if the partner is an internal auxiliary processor (one distributed as part of VS APL). [INTAPCHK]

   For offer, SCVPART in the SCV is checked to determine if the offer is to an internal auxiliary processor. [PCSOFFER]

   a. If the partner is not an internal auxiliary processor, the user's ID is placed in the SCV, and the appropriate VSPC service request is issued. [PCSACC, PCSCOPY, PCSREF, PCSRET, PCSSPEC, PCSOFFER]

   APLPAPAB

   b. Retraction when partner is an internal auxiliary processor. If variable CTL: If file is open, a VSPC service request VCLOSE or DCLOSE is issued; for FSM auxiliary processor, a TFSCRN EXIT is issued; for GDDM auxiliary processor, GDDMCRET is called, and if no more paths remain, then GDDMSOFF is also called.

      An SCV is built, including flag SCVFDOFR, which indicates that both partners have retracted, and a VSPC service request SRET is issued. The APFT entry is updated if the other variable for a connection is active; the entry is cleared if it is not. [APLPAPRT]

   APLPAPAB

   c. Offer to internal auxiliary processor. The APFT entries are searched to determine if this is a new connection or the second variable for an existing connection. Accordingly, a new APFT entry is built, or the existing APFT entry is modified. For offers to the FSM internal auxiliary processor, only one connection is allowed at any one time. For offers to the GDDM internal auxiliary processor, a maximum of seven connections are allowed at any one time. In addition, concurrent sharing with the FSM and GDDM internal auxiliary processors is not allowed. An SCV is built, including flag SCVFDOFR, which indicates that both partners have offered, and a VSPC service request SOFR is issued. [APLPAPOF]

      For the VSPC command and alternate input auxiliary processors, the initial value of the variable (if any) is checked and the return code is set in the APFT (in case the user references the variable).

   APLPAPAB

   d. Set access control when partner is an internal auxiliary processor. SCVACV in the SCV is set to binary '1111'. [APLPAPAC]

   APLPAPAB

   e. Copy when partner is an internal auxiliary processor. The return and reason codes that indicate that the latest value is in the workspace are placed in PTHSRCOD. [APLPAPPR]

      Reference or specify when partner is an internal auxiliary processor (finite state machine logic, driven by APFIFO action stack in APFT): If an interlock exists, a VSPC service request TWAIT is issued. [APLPAPPR]

User specifies the CTL: When partner is file-handling auxiliary processor, and the VSPC file is open for sequential input, then if the value is null, a VSPC service request DCLOSE is issued; otherwise, the value is ignored. [APUSCTL]

If the APFIFO action stack in the APFT contains a pending "AP references CTL" action (the usual case), the finite state machine logic in APLPAPPR will proceed to call the APARCTL subroutine immediately after the APUSCTL subroutine. The APARCTL subroutine contains the entire processing logic of the VSPC command, alternate input, and storage display auxiliary processors. For the VSPC command auxiliary processor, the VSPC service request WCMD is issued; for the alternate input and storage display auxiliary processors, the processing consists of analyzing the request in the user's variable and then copying data from one place to another within the workspace. For the other internal auxiliary processors, the APARCTL subroutine analyzes the user's request and calls the appropriate routine to process it. Routines in module APLPAPCD are called to handle requests for the APL data file, EBCDIC data file and VSAM auxiliary processors. Routines in module APLPAPFS are called to handle requests for the FSM auxiliary processor. The routine GDDMRCTL in module APLPAPGC is called to handle requests for the GDDM auxiliary processor.

APLPAPCD

User specifies CTL and VSPC file is open for sequential output. If the value is null, a VSPC service request DCLOSE is issued. Otherwise, data is transferred from the workspace to the I/O buffer, and a VSPC service request DWRITE is issued. [PWRITE]

User specifies CTL and the VSPC file is open for direct input or update. If the value is null, VSPC service request DCLOSE is issued; otherwise, the value is examined to determine whether the request is to read or write. [APIO]

If the request is to read, a VSPC service request DREAD for specified record is issued. Data is left in I/O buffer until the user references DAT. [PRDDIR]

If request is to write, data is transferred from the workspace to the I/O buffer, and a VSPC service request DWRITE for a specified record is issued. [PWRITE]

User specifies CTL and the VSPC file is not open. An appropriate VSPC service request corresponding to the user's request is issued. [APCREATE, APFILSIE, APSHARE, APPASSWD, APOPEN, APDROP]

User specifies CTL and partner is VSAM file auxiliary processor. An appropriate VSPC service request corresponding to the user's request is issued. If the request is to write, data is first transferred from the workspace to the I/O buffer. If the request is to read, the data is left in the I/O buffer until the user references DAT. [APVIO]

User references CTL and the VSPC file is open for sequential input. VSPC service request DREAD is issued, and data is transferred from the I/O buffer to the workspace. [PRDSEQ]

APLPAPFS

User specifies CTL and partner is FSM auxiliary processor. If not already obtained in previous connections, FSM auxiliary processor obtains storage out of user's VSPC workspace quota, size depending on number and characteristics of FSM fields defined, for use as FSM work area. [FSMFORMT]

If user issues request to read from display screen, VSPC TSFSM READ service request is issued. For read and read-format requests, data is transferred to workspace when user next references DAT. [FSMREAD, FSMGET, FSMRFORM]

If user issues a request to write to display screen, data is transferred to FSM work area and VSPC TSFSM WRITE service request is issued. [FSMWRITE]

If user issues a request to format, modify field characteristics, modify field intensity, set cursor position, or sound alarm, the request data is recorded in FSM work area to be communicated to VSPC at the next display screen read or write request. [FSMFORMT, FSMMTYPE, FSMMINT, FSMSETC, FSMBUZZ]

If request is to make hard copy of display screen data, VSPC TSFSM PAGE service request is issued. [FSMHCOPY]

APLPAPGC

If this is the first invocation of GDDMRCTL for this path (connection via a CTL-DAT pair), the GDDXINIT routine in module APLPAPGD is called to initialize the path.

If DAT variable was specified by the user, it is referenced; the CTL variable. The CTL variable specified by the user is analyzed, and the appropriate series of GDDM requests are built. The GDDX routine in module APLPAPGD is called to issue each GDDM request that is built. The output parameters from all the GDDM requests are accumulated and formatted into numeric and character output buffers, which are later transferred into the user's workspace by the GDDMSCTL and GDDMSDAT routines (in module APLPAPGB), respectively. Certain GDDM auxiliary processor requests are internal to the auxiliary processor and do not involve issuing a GDDM request; these internal requests are handled entirely within the GDDMRCTL routine. [GDDMRCTL]

APLPAPGD

A path control block index is allocated. If this is the first path to be allocated, GDDM is initialized by issuing SPINIT and FSQERR requests to GDDM via the VSPC service request TGDDM. [GDDXINIT]

Request built by the caller is analyzed for "pass through" or "special case" processing. Special processing is performed for page, query error, and hardcopy requests. If the request built by the caller requires a "page select" operation, then a GDDM FSPSEL request is chained to the front of the caller's request. The VSPC service request TGDDM is issued to pass the required request(s) to GDDM, and the VSPC return and reason codes are analyzed and converted to standard GDDM return and reason codes. [GDDX]

APLPAPAB

User references CTL (all other cases). Return and reason codes from prior request are transferred to the workspace. For GDDM auxiliary processor the GDDMSCTL routine is called if return code vector buffer exists. [APURCTL]

User specifies DAT. Event is recorded in APFT entry. No further action is taken until the user issues a write request. [APUSDAT]

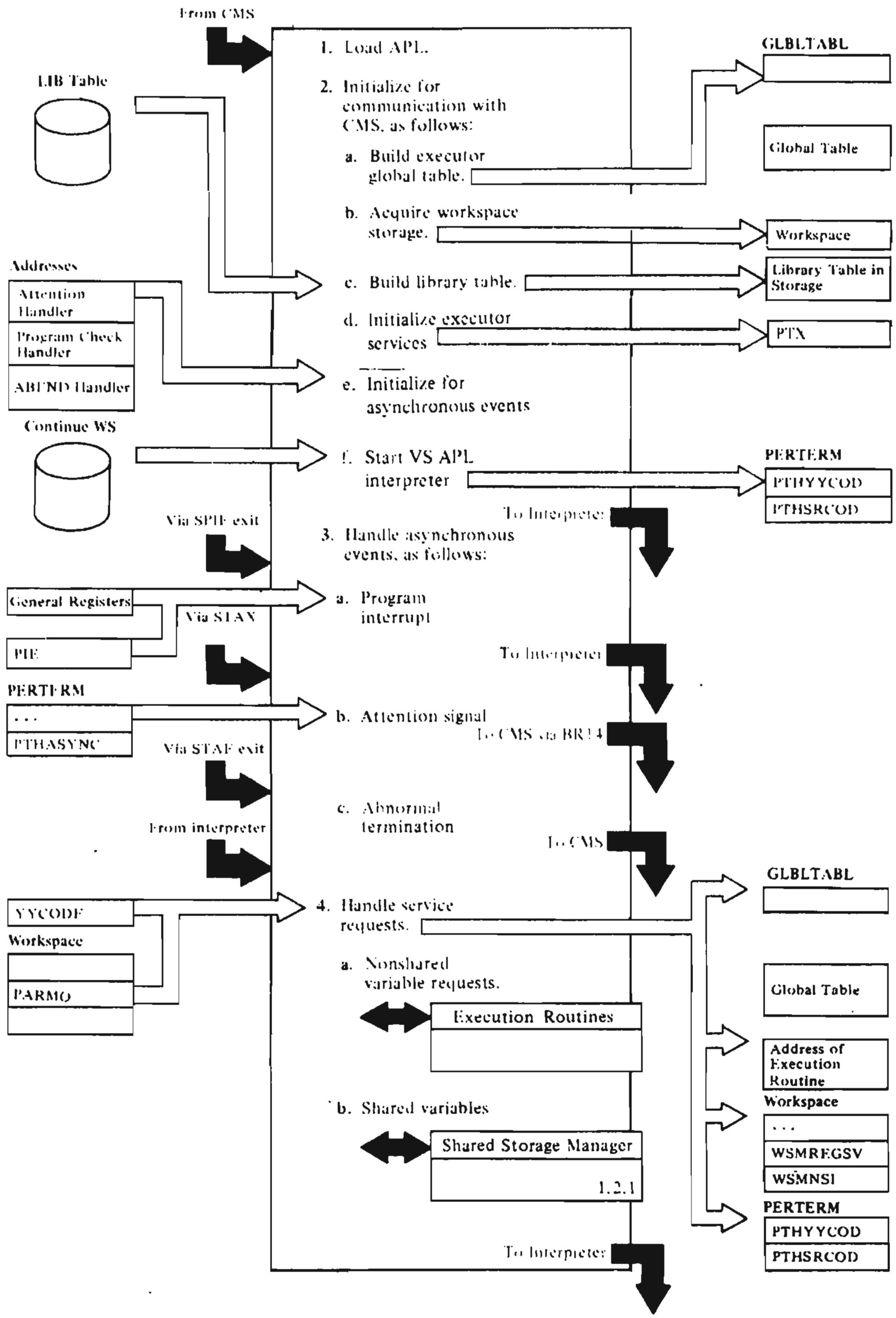
User references DAT. Data from prior read request is transferred from the I/O buffer or FSM work area to the workspace. For GDDM auxiliary processor, the GDDMSDAT routine is called. [APURDAT]

APLPAPGB

CTL variable data in the GDDM numeric output buffer is copied into the user's workspace and is converted to VS APL "variable descriptor" format in the process. The GDDM numeric output buffer is deallocated. [GDDMSCTL]

DAT variable data in the GDDM character output buffer is copied into the user's workspace and is converted to VS APL "variable descriptor" format in the process. The GDDM character output buffer is deallocated. [GDDMSDAT]

DIAGRAM 1.2: COMMUNICATION WITH CMS



Notes for Diagram 1.2

APLSCINI

1. Attempt to locate a VM DCSS for APL.

   If a suitable DCSS is found, load it using Diagnose 64, and transfer to it.

   If DCSS is not to be used, then perform a LOADMOD VSAPL and transfer to it.

APLSCINI

2. The initialization process (module APLSCINI) performs the following functions at VS APL startup:

   a. Gets space for and initializes the executor global table. This table (mapped by the APLCMSGL macro) contains the PERTERM terminal buffers and key switches and pointers. It is always pointed to from location X'440' (GLBLTABL).

      Scans the startup parameter list. Loads text files for any auxiliary processors.

      After the parameter list is scanned, the APLEXIT user EXEC is invoked to establish the VM environment.

      Sets STAE and STAX exits.

   b. Allocates space for shared memory, auxiliary processor work areas (512 bytes per auxiliary processor), and the workspace. Gives back free space to CMS.

   c. Reads the library table file (APLIBTAB APLIBTAB) and builds the incore library table.

      Determines if VS APL microcode assist is to be used.

      Calls the shared storage manager to initialize any auxiliary processors.

      Initializes pointers and keys in the incore workspace.

   d. Initializes executor services for the stack manager, the file subsystem, and the session manager.

      Determines if the CONTINUE workspace is to be auto-loaded.

      Determines if terminal is display or typewriter and initializes accordingly.

   e. Sets SPIE exit.

   f. Places service request YYON in PTHYYCOD and passes control to SCAPL. From there, control is passed to the interpreter at its entry point APLIINIT.

APLSCERR, APLSCTYP

3. Asynchronous handling applies to program checks, attention exits, and abends.

   Program checks (SPIE exit): [SCSPIE]

   These are handled in module APLSCERR, routine SCSPIE (except during VS APL startup, when it is handled by routine SPIEXIT in module APLSCINI).

   Routine SCSPIE does the following:

   - Saves the program check registers and PSW in WSMSURGS and WSMSUPSW (in the workspace).
   - If the program check occurred in supervisor code, prints messages S631S, S633I, S634I, S635I, and abnormally terminates VS APL with the user code 1xx, where xx is the program check code in decimal.
   - If the program check occurred in the shared storage manager or an auxiliary processor, prints message APL114I and handles the check as an interpreter program check.
   - If the program check occurred in the interpreter, checks to see if the interpreter is in a program check loop (prints message APL101I and ABEND if so). If not in a loop, moves the registers from WSMSURGS to WSMREGSV and the PSW from WSMSUPSW to WSMPCPSW and WSMNSI.

If a program check loop occurs, the registers and the PSW for the next-to-last program check will be in WSMREGSV and WSMPCPSW/NSI and the registers and PSW for the last fatal program check will be in WSMSURGS and WSMSUPSW.

Attention exits (STAX exit):

The STAC exit is in APLXGCAT. APLXGCAT saves information about the attention and transfers control to the address in PTXATTN. This will point either to a session manager routine or to SCATTN.

Asynchronous interrupts for the active workspace are handled by routine SCATTN in module APLSCTYP, which does the following:

- Sets attention bit(s) in the PERTERM.
- If attention is pressed during wait for message response, completion of time delay, or shared variable request, posts an ECB.
- If attention is pressed during terminal output and/or function execution, returns the print element to position 0.
- Returns to point of interrupt.

Abends (STAE exit):

There are two types of STAE exits. The subsystem STAE exit is established by processors calling APLXBSXT (in APLSCSVI). When a subsequent abend occurs, the subsystem exit (BSXTSTXE) schedules a retry routine and then passes control to it with diagnostic information in the BND.

If no exit has been requested, message APLS620E is issued, and the processor is marked nondispatchable.

These are handled in module APLSCERR by routine SCSTAE, which does the following:

a. Prints messages S644E and S632D.

b. Address stops the virtual machine to allow the user to dump storage and do problem determination in CP mode. At the time of the address stop, the following information is relevant:

Reg.	**Contents**
R2	Contains the address of the 104-byte STAE work area.
R8	Contains the ABEND code.
R10	Contains the address of the VS APL supervisor global table.
R11	Contains the address of the VS APL incore workspace slot. (If the ABEND code is 1xx, then the workspace has the program-check PSW and registers.)

APLSCFXI

4. Service request handling: [routine APLFXIIM]

   Service requests allow the interpreter to interact with its environment (for example, type a line, load a workspace). Any module in the interpreter may issue a service request. The linkage is:

```
L          R1,=V(APLFXIIM)
BALR       R0,R1
DC         AL2(YYCODE)
```

   Routine APLFXIIM is in executor module APLSCFXI. It does the following for every service request:

   a. Saves the general registers in WSMREGSV (except for YYDUMP, for which we want to preserve the contents of WSMREGSV), the floating registers in WSMREGF0, F2, F4, F6, and the address of the caller's resume point (R0+2) in WSMNSI. (All WSM fields are in the workspace.)

   b. Changes the protect key in the PSW from X'D' (the interpreter protect key) to X'E' (the executor key).

      Changes the storage key of the first 4K bytes of the workspace from key X'D' to X'E' so the executor can store data there.

c. Adds the processor time used by the interpreter to an accumulated-processor-time field (CMSCPUAC) for the quad-AI system variable.

d. Stores the YYCODE (which determines the type of request) in the PERTERM (field PTHYYCOD).

e. Looks up the request type in YYTABL (module APLSCFXI) and gets the address of the execution routine.

f. Calls the execution routine to execute the service request. The execution routine will return with the service request return code set in PTHSRCOD.

g. Updates the WSMASYNC bits in the workspace to reflect the latest status of asynchronous events (for example, attention).

h. Changes the storage key of the first 4K bytes of the workspace from X'E' back to X'D' so the interpreter can store data there.

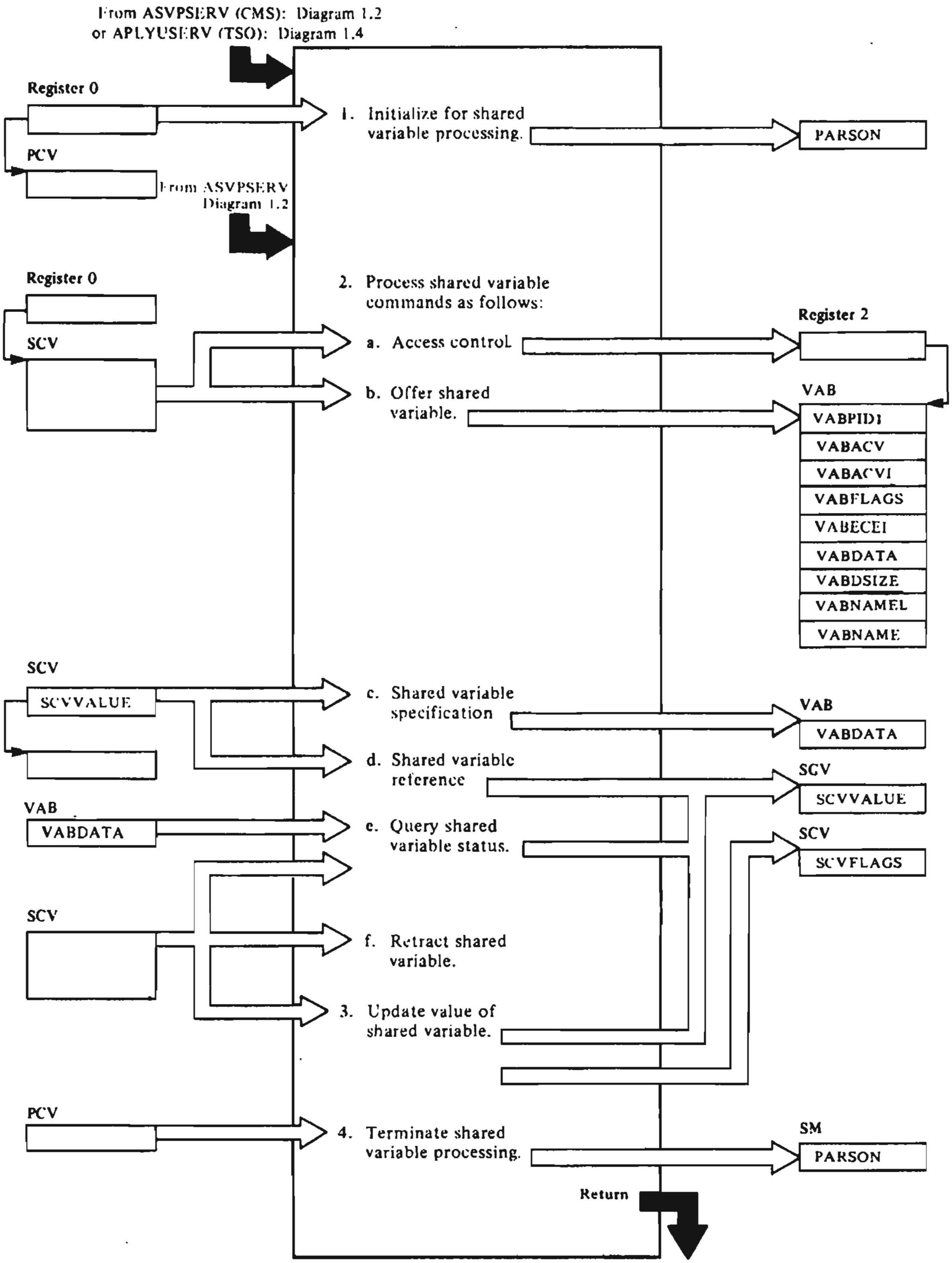
i. Sets the current time in CMSHOLDT so that processor time for the interpreter can be accumulated for quad-AI.

j. Goes back to interpreter in PSW key X'D' at its entry point, APLIINIT. There, the interpreter's environment is restored, and control is returned to the instruction following the service request.

For a description of the service request codes and the names of the CMS executor routines that handle them, see "Values, Parameters, and Return Codes for Service Requests" under "Service Request Calls" in "Section 6. Diagnostic Aids."

For **shared variable** processing, control is passed to ASVPSRVC and then to ASVPSERV to route the request to the appropriate routine in the shared storage manager. (See Diagram 1.2.1: "Shared Storage Manager.") After control returns to ASVPSERV, each auxiliary processor whose wait has been satisfied receives control. Control is then returned to the interpreter.

DIAGRAM 1.2.1: SHARED STORAGE MANAGER (CMS AND TSO)



Notes for Diagram 1.2.1

Return codes from each step are passed in registers 15 and 0.

APLSHGET

1. This function occurs as a result of an explicit request for the shared variable processor or implicit request through a shared variable command. Space is obtained from shared memory for the processor control block. [PRB]

   APLSHBPB

   The PRBID, PRBSPACQ, PRBVARSQ, and PRBECB fields of the PRB are set with data from the processor control vector. [PCV]

   APLSHSON

   The count of processors using the shared variable facility is updated in the PARSON field of the shared memory data area.

2. Shared variable commands are processed as follows:

APLSHSRD

a. Access control [⎕SVC]. The address of the field in the VAB data area that corresponds to the SCV fields of the offered shared variable is returned in register 2.

   APLSHACC

   The ACV, VABACV, and SCVACV fields of the VAB area are set to allow access control.

APLSHGET

b. Offer [⎕SVO]. Space is obtained from shared memory for the variable control block. [VAB]

   APLSHBVB

   Fields of the VAB are set for initial offer.

   APLSHOFR

   For counter offer or general offer, the fields are updated.

c. Specification of a shared variable. A new value for a shared variable is processed as follows:

   APLSHSRD

   The address of the VAB field corresponding to the SCV fields of the shared variable is returned in register 2.

   APLSHPUT

   Space used by the previous value is freed.

   APLSHGET

   Space required for the new value is acquired.

   APLSCSVI (CMS), APLSHSPC, APLYUSVI (TSO)

   The new value is entered in the VABDATA field of the VAB. If necessary, the shared variable partner is posted.

APLSHSRD

d. Reference. The address of the VAB field corresponding to the SCV fields of the shared variable is returned in register 2.

   APLSHREF

   The latest value of the variable is moved from shared memory to the buffer.

   APLSCSVI (CMS), APLYUSVI (TSO)

   If necessary, the shared variable partner is posted.

   The storage block for the data is freed if both partners of the shared variable have obtained the data.

e. Query [⎕SVQ]. For request for partner identification and offer numbers or for variable names and offer numbers, a list is constructed in the buffer whose address is in the SCVVALUE field. For request for single variable, information is entered in the SCV fields.

APLSHSRD

f. Retract. The address of the VAB field corresponding to the SCV fields of the shared variable is returned in register 2.

APLSHRET

SCVFLAGS are updated to reflect the degree of coupling.

APLSHSUB

VAB and PRB fields are updated to reflect retraction.

APLSHPST

If necessary, the shared variable partner is posted.

APLSHPUT

Shared memory used by the VAB is returned.

APLSHSRD

3. The address of the VAB field corresponding to the SCV fields of the shared variable is returned in register 2.

APLSHCPY

The value of the data is moved from shared memory to the buffer whose address is in SCVVALUE.

4. For logoff from the shared variable processor, processing occurs, as follows:

APLSHSOF

The number of processors in the PARSON field of shared memory is decremented.

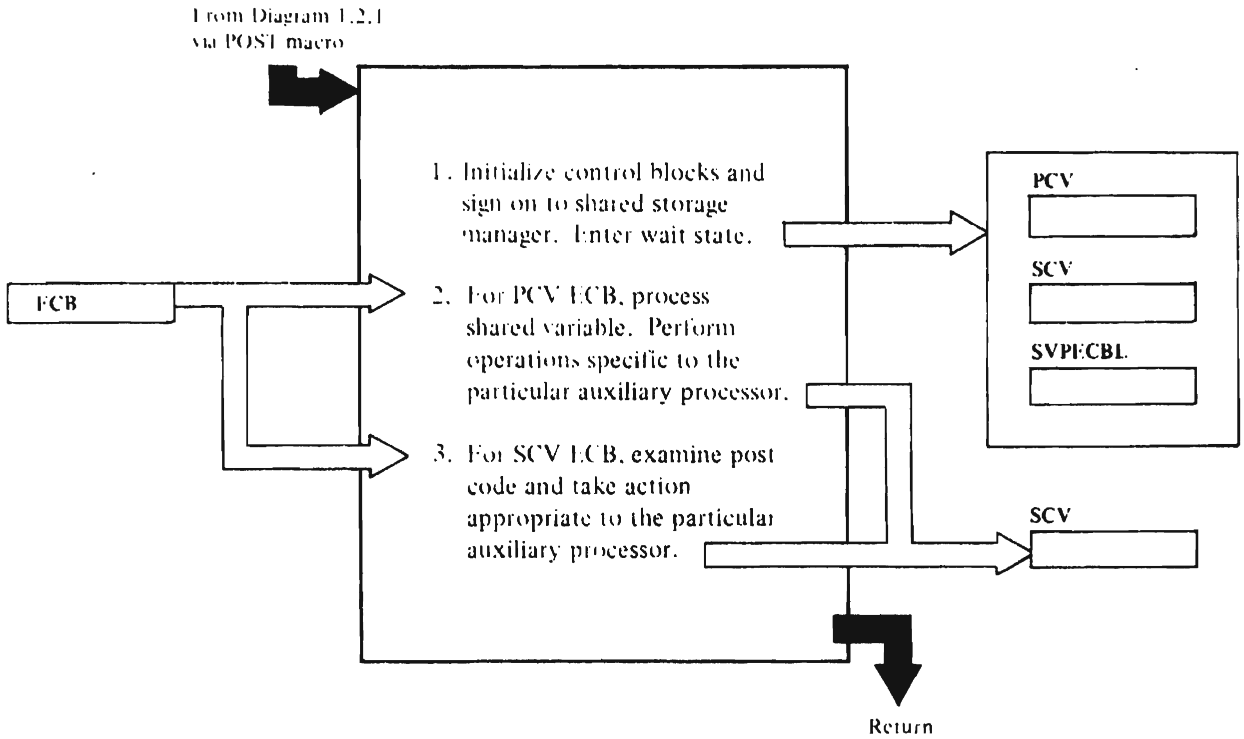
APLSHSUB

Each variable offered by the processor is retracted.

APLSHPUT

Shared memory used by the PCV block is released.

DIAGRAM 1.2.2: AUXILIARY PROCESSORS (CMS)



Notes for Diagram 1.2.2

1. Initialization

Initialization is performed as follows: The process control vector is completely filled in. The SCVID and SCVECB fields of the shared control vectors are filled in. The addresses of the ECBs are placed in SVPECBL. The PCV ECBSW switch is set in SVPECBL to identify the PCV ECB. The auxiliary processor signs on to the shared storage manager.

The auxiliary processor waits for an ECB to be posted. [WAIT]

APL100: CMS COMMAND

2. For PCV ECB

When control is returned from the wait state and the PCV ECB is posted, the following occurs:

An SCV is assigned to be used by the shared variable. [SCVLOOP]

A query is issued to find the variable's name. [DOQUERY]

A counter-offer is issued to complete the sharing of the variable. [QUERYSUB]

The variable is referenced. [GETNXVAR]

The type of command to be executed (CP or CMS) is determined. [REFOK]

The auxiliary processor waits for an ECB to be posted. [WAIT]

3. For SCV ECB

When control is returned from the wait state and an SCV ECB is posted, the post code is examined. [CHKPSTCD]

Processing then occurs as follows:

If the partner referenced the variable, the return code is specified. [RCODE]

If the partner set the access control vector, this event is ignored. [CLRPSTCD]

If the partner retracted the variable, the variable is retracted and the SCV is made available for another variable. [RETRACT]

If the partner specified the variable, the variable is referenced and the command is executed. [GETNXVAR, TRANZCOD]

The return code is specified. [RCODE]

The auxiliary processor waits for an ECB to be posted. [WAIT]

APL101: ALTERNATE INPUT

2. For PCV ECB

When control is returned from the wait and the PCV ECB is posted, processing occurs as follows:

An SCV is assigned to be used by the shared variable. [SCVLOOP]

A query is issued to find the variable's name. [DOQUERY]

A counter-offer to complete the sharing of the variable is issued. [QUERYSUB]

The variable is referenced. [GETNXVAR]

The stacking and conversion options are determined. [REFOK]

The auxiliary processor waits for an ECB to be posted. [WAIT]

3. For SCV ECB

When control is returned from the wait state and an SCV ECB is posted, the post code is examined. [CHKPSTCD]

Processing occurs as follows:

If the partner referenced the variable, the return code is specified. [RCODE]

If the partner set the access control vector, this event is ignored. [CLRPSTCD]

If the partner retracted the variable, the variable is retracted and the SCV is made available for another variable. [RETRACT]

If the partner specified the variable, it is referenced and converted. The line is stacked according to the options determined in step 2 above. [GETNXVAR]

The return code is specified. [RCODE]

The auxiliary processor waits for an ECB to be posted. [WAIT]

APL110: CMS FILE

2. For PCV ECB

When control is returned from the wait and the PCVECB is posted, processing occurs as follows:

An SCV is assigned to be used by the shared variable. [SCVLOOP]

A query is issued to find the variable's name. [DOQUERY]

A counter-offer to complete the sharing of the variable is issued. [QUERYSUB]

The variable is referenced. [GETNXVAR]

The conversion option is determined, and the file name is placed in the FSCB. [INIT]

Whether the file exists or not is determined, and the rest of the FSCB is filled in. [TRYFILE]

The auxiliary processor waits for an ECB to be posted. [WAIT] 3. For SCV ECB

When control is returned from the wait and an SCV ECB is posted, the post code is examined. [CHKPSTCD]

Processing continues as follows:

If the partner referenced the data variable, the file is read and the data converted according to the options determined in step 2 above. [PARTREF]

The converted data is then specified. [SPEC1]

If the partner referenced the control variable, the return code from the last operation involving the data variable, the read-pointer, the write-pointer, and the number of records to be processed are specified as a 4-element integer vector. [RCODE]

If the partner set the access control vector, this event is ignored. [CLRPSTCD]

If the partner retracted the variable, the file is closed, the variable is retracted, and the SCV is made available for another variable. [RETRACT]

If the partner specified the data variable, it is referenced and converted according to the options determined in step 2 above. [GETNXVAR,CONVERT]

The converted data is then written to the CMS file. [WRITE1]

If the partner specified the control variable, the read and write pointers and number of records to be processed are altered as specified. [SETCTL]

The auxiliary processor waits for an ECB to be posted. [WAIT]

APL111: QSAM

2. For PCV ECB

When control is returned from the wait and the PCV ECB is posted, processing occurs as follows:

An SCV is assigned to be used by the shared variable. [SCVLOOP]

A query is issued to find the variable's name. [DOQUERY]

A counter-offer to complete the sharing of the variable is issued. [QUERYSUB]

The variable is referenced. [GETNXVAR]

The conversion option is determined and the file name is placed in the DCB. [CHKPARM]

The auxiliary processor waits for an ECB to be posted. [WAIT]

3. For SCV ECB

When control is returned from the wait and an SCV ECB is posted, the post code is examined. [CHKPSTCD]

Processing continues as follows:

If the partner referenced the data variable, the file is read and the data converted according to the options determined in step 2 above. [PARTREF]

The converted data is then specified. [SPEC1]

If the partner referenced the control variable, the return code from the last operation involving the data variable is specified. [RETNCODE]

If the partner set the access control vector, this event is ignored. [CLRPSTCD]

If the partner retracted the variable, the file is closed, the variable is retracted, and the SCV is made available for another variable. [RETRACT]

If the partner specified the data variable, it is referenced and converted according to the options determined in step 2 above. [GETNXVAR,CONVERT]

The converted data is then written to the OS file. [WRITE]

If the partner specified the control variable, it is referenced and ignored. [CLRPSTCD]

The auxiliary processor waits for an ECB to be posted. [WAIT]

APL123: VSAM

2. For PCV ECB

When control is returned from the wait and the PCV ECB is posted, processing occurs as follows:

An SCV is associated with the variable. [SCVLOOP]

A query is issued to find the variable's name. [DOQUERY]

If the name does not begin with CTL or DAT or if it is greater than 11 characters or if the name is already shared, the offer is not accepted.

A counter-offer to complete the sharing of the variable is issued. [INIT]

After a counter-offer, the auxiliary processor waits. [WAIT]

3. For SCV ECB

When control is returned from the wait and an SCV ECB is posted, the post code is examined. [CHKPSTCD]

Processing continues as follows:

If the partner retracted the variable, the sharing of the variable is terminated, the file is closed (if it was opened), and the SCV is made available for another variable. Processing of any outstanding offer is attempted. [RETRACT]

If the partner specified the control variable, an appropriate action is performed:

a. For an OPEN request, the file is opened if available but not opened if already open. [VOPEN]

b. For a CLOSE request, the file is closed. [VCLS]

c. For READ, the file is read, and the data is specified into the DAT variable. [FILREAD]

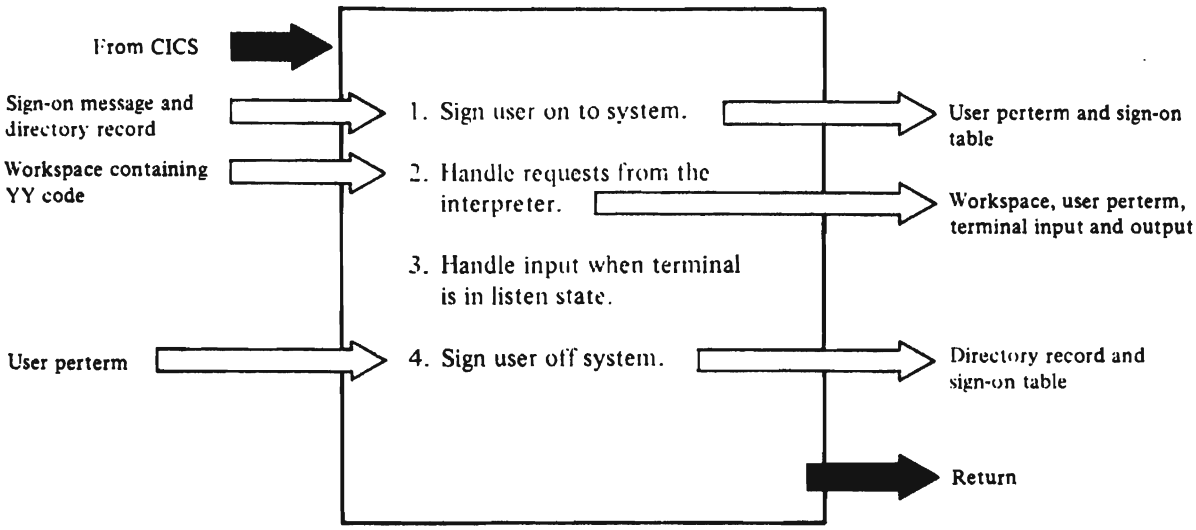
d. For WRITE, the DAT variable is referenced, and its data written to the file. [FILWRITE]

e. For ERASE and POSITION, the appropriate action is taken. [VERASE, VPOS]

f. For KEYFEEDBACK, the key of the record last processed is specified in the DAT variable. [KEYFDBK]

   The control variable is specified with a 2-element return code for all operations.

DIAGRAM 1.3: COMMUNICATION WITH CICS/VS



Notes for Diagram 1.3

APLKASON, APLKAGBL, APLKLIBB

1. APLKASON calls APLKAGBL which determines whether the global table is active and, if not, loads the global modules and calls APLKLIBB to initialize the library control blocks.

   Using the sign-on message as input, APLKASON initializes a perterm for the user.

   Using the user profile directory record as input, APLKASON then performs user and terminal verification, attaches the user task (APLKADSP), and exits. Output is the user perterm (PTH, PTX, PTK, and PRO control blocks).

   APLKADSP, APLKIFIX

   APLKADSP sets up the user task environment, including APLKWAIT and APLKEXIT macro services and dependent process control. APLKADSP then starts the interpreter process by calling APLKIFIX, which sets up the interpreter interface and calls APLASCHD and APLKLIBC.

   APLASCHD

   Initializes the terminal.

APLKIFIX

2. Accepts requests from the interpreter in the form of YY codes passed in the workspace and, based on the type of request, routes control as follows:

Module	Entry Point(s)	Function
APLASCHD	TYO, TYI, TYOI	Terminal Services for I/O
APLKISVI	SON, SOFFER, SRET, SQUERY, SACC, SSPEC, SREF, SCOPY, SOFF	Shared variable services
APLKLIBU	COPI, COPO, COPZ, LOAD, COPA, SAVE, DROP, LIB, CLEAR, WSID, PASS	Library services
APLKMSCA	TIME, QAI, DELAY, DUMP, SYSER, CMD	Time and error services
APLKMSCB	QZ, ATOFF, TABS, WIDTH, MBL, TRAN, QUOTA, OFF	Miscella-neous local and unsup-ported services

APLASCHD

Performs terminal I/O.

APLKEMGR, APLKEHCP

The destination manager. Provides an interface to CICS transient data and to 3270 printer terminals.

APLKLIBU, APLKLIBF, APLKLIBG, APLKLIBV, APLKLIBA, APLKLIBB, APLKLIBR

The library manager. Provides access to VS APL workspaces and files, all of which are stored in the APL library. In performing these operations, these modules call on CICS file services and DOS/VS or OS/VS VSAM services.

APLKISVI, APLKSSVP, APLKSSUB, APLKADEF

The shared storage manager and the interpreter interface to the shared storage manager. Provides communication between auxiliary processors and APL users and manages the use of shared memory.

APLKMSCA, APLKMSCB

Performs miscellaneous services for the interpreter. In performing these services, the CICS dump services and global task timer services may be called.

APLKDOPS, APLKVOPS, APLKASTB

Performs services dependent on use of the operating system. Modules APLKDOPS (for DOS/VS) and APLKVOPS (for OS/VS) provide VSAM macros, handling of VSAM and ISAM return codes, and timer support for time slicing. APLKASTB provides support for DOS/VS page fault overlap conditions.

Note: The following information applies to both steps 1 and 2 of diagram 1.3.

APLKLIBO, APLKLIBV, APLKLIBA

Services a library request made by APLXLIBF or APLKLIBU. These modules execute as separate CICS/VS tasks started by APLKASTB. APLKLIBG gains control first, and performs most of the services for APLKLIBU. For APLKLIBF services, it calls APLKLIBV. Either APLKLIBG or APLKLIBV may call APLKLIBA to allocate or deallocate space in the library.

APLKASON, APLKTCTL, APLKTCWR, APLXGKT

3. APLKASON is initiated as a sign-on attention transaction if the user sends input when the terminal is in listen state (in other words, when APL has more work to do for the user, but no terminal read or write operations are outstanding).

   If the APL user is already signed on, APLKASON give control to the APLXGKT if GDDM is being used, control is given to APLKTCTL.

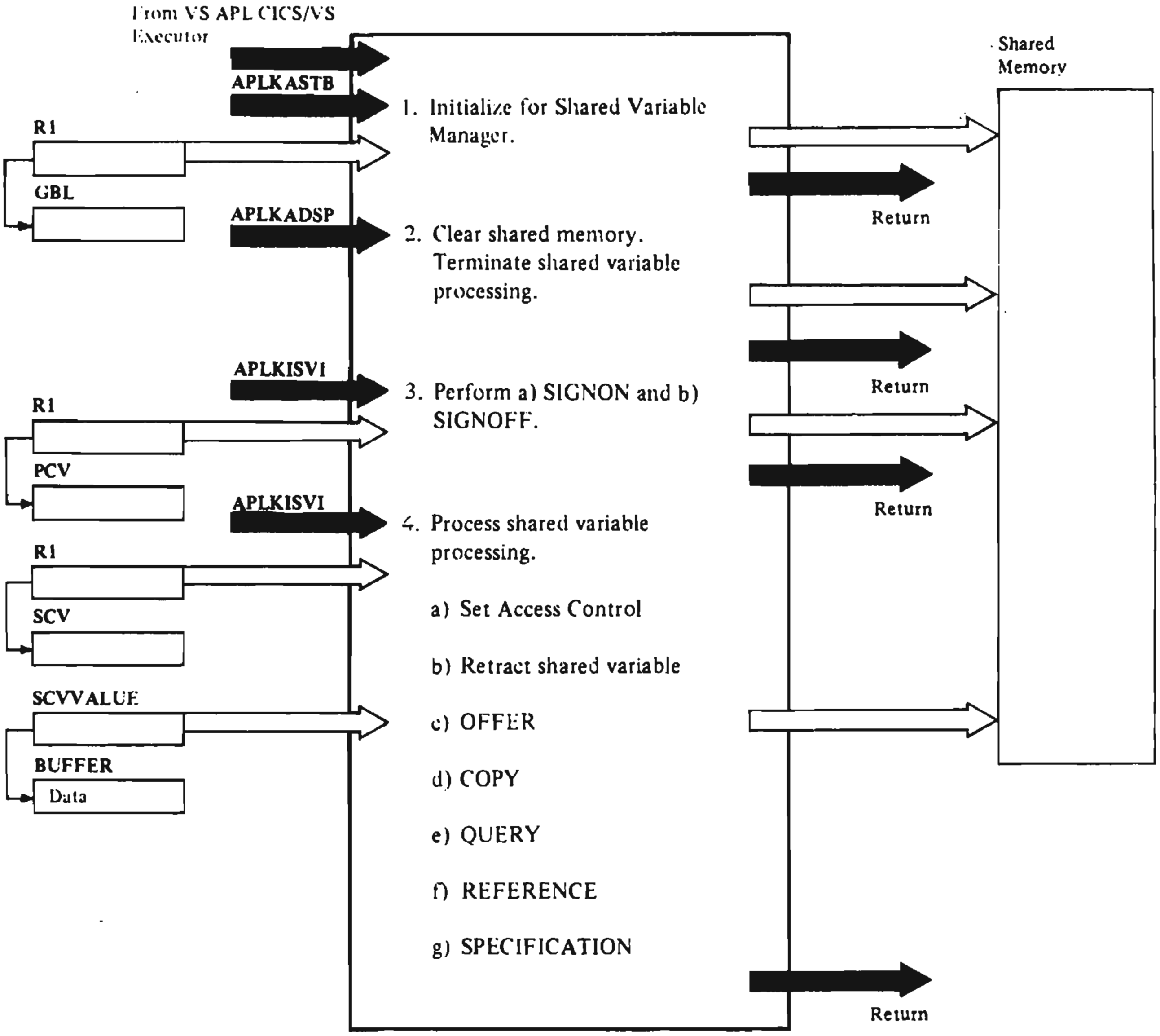
APLKLIBC, APLASCHD, APLXMSCB

4. APLXMSCB controls sign-off processing, calling on library and session manager termination routines to assist in sign-off processing. APLKLIBC cleans up the workspace storage and APLASCHD initiates session manager and terminal cleanup as described in diagrams 8.1 and 8.2.

   APLKIFIX, APLKADSP, APLKAGBL

   When APLKIFIX receives control from APLKMSCB after a YYOFF request, it exits to APLKADSP; APLKADSP then terminates any processing being done by dependent auxiliary processors, deletes the user's sign on entry from the sign on table, and, if no other users are signed on to the system, causes the global task to terminate processing (unless independent auxiliary processors are still using the shared storage manager).

DIAGRAM 1.3.1: SHARED STORAGE MANAGER (CICS/VS)



Notes for Diagram 1.3.1

The return code and the reason code are passed in R15 and in R0. For tasks 2, 3, and 4 on entry, R0 has the request code.

APLKSSUB

1. Obtains space for and initializes the shared memory (SM). The CICS/VS service DFHSC TYPE = GETMAIN, CLASS = PROGRAM is employed to derive the storage. [APLKSINI]

APLKSSVP

2. Storage used by the shared variable processor (SM) is released and shared variable processing is terminated. [APLKSSR]

APLKSSVP [APLKSSR]

3. Options when the terminal is in listen state:

   a. SIGN-ON: A processor control block (PERPROC) is obtained from SM for the user. The user ID, shared variable number quota, and space quota are placed in the PERPROC.

   b. SIGN-OFF: All of this user's shared variables are retracted, and PERPROC is released.

APLKSSVP [APLKSSR]

4. Options when the user signs off the system:

   a. SET ACCESS CONTROL: The access control vector (ACV) for a shared variable is altered. The effective ACV is returned.

   b. RETRACT: Retracts the sharing of a single variable. If the partner has already retracted, the PERSHARE for this variable is released.

   c. OFFER: Offers to share a single variable with another processor. If it is not a counter-offer, a share entry (PERSHARE) for this variable is obtained from the SM.

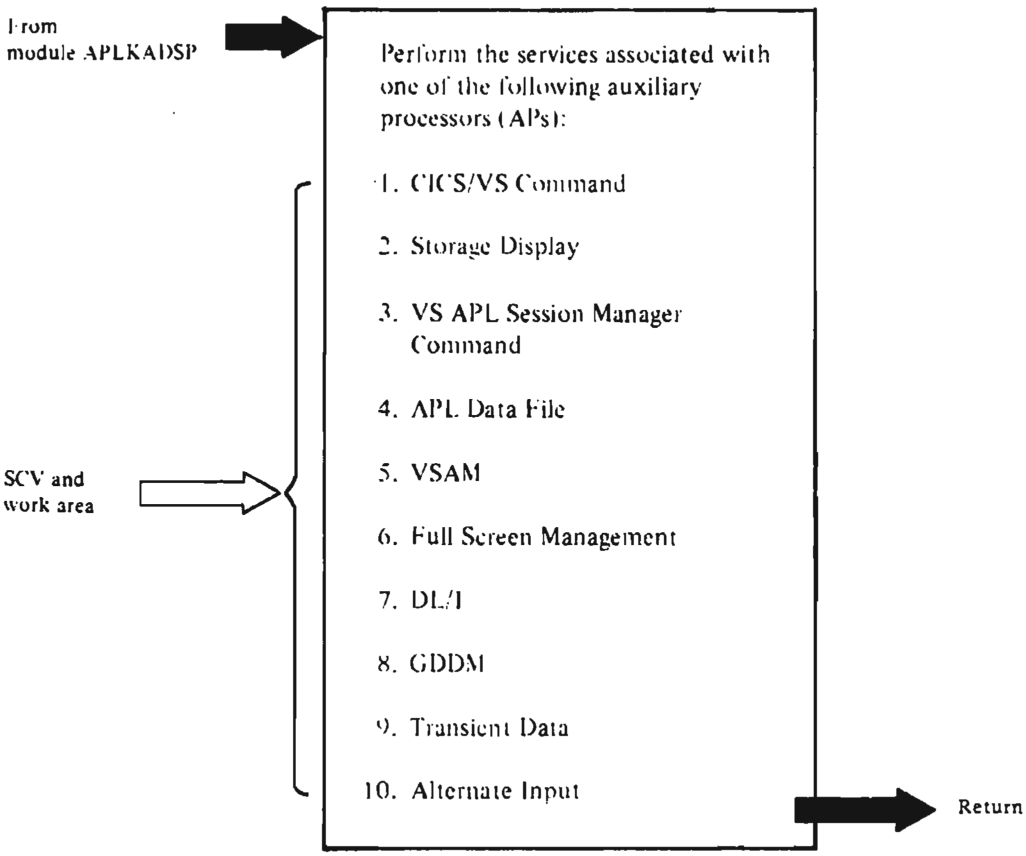
   d. COPY: Copies the latest value of a shared variable. The access state of the variable is not changed.

   e. QUERY: Obtains information about shared data items.

   f. REFERENCE: References the latest value of a shared variable.

   g. SPECIFICATION: Specifies a new value for a shared variable.

DIAGRAM 1.3.2: AUXILIARY PROCESSORS (CICS/VS)



Notes for Diagram 1.3.2

APL100K, APL100KO

1. APL100K issues CICS/VS commands, and attaches APL100KO, which starts CICS/VS transactions.

AP102K

2. Displays storage for the user.

AP120

3. See Notes for Diagram 8.4.1.

APL121K

4. Creates, writes, updates, reads, and/or deletes APL object files. Uses library services (part of the CICS/VS executor) to access the APL library.

APL123K

5. Using CICS/VS file services, reads from and writes to VSAM and ISAM data sets.

APL124K

6. Permits APL functions to format and control the user display terminal. Calls the terminal manager (part of the CICS/VS executor) to provide physical terminal services.

APL102K

7. Displays storage for the user.

APL125K

8. Provides an interface to CICS DL/I services for the CICS/VS user.

APL126

9. See Notes for Diagram 8.4.2.

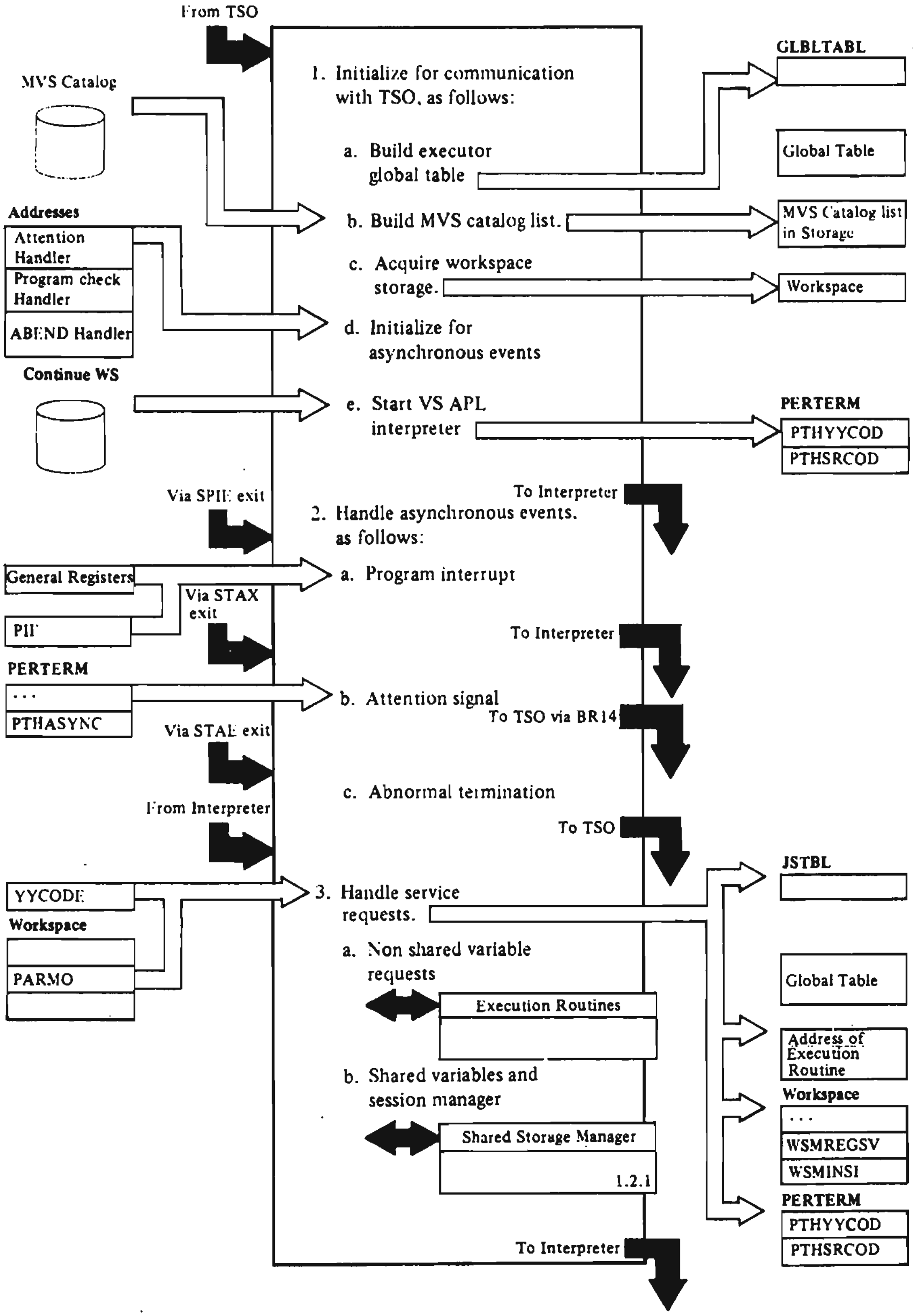
APL132K

10. Accesses CICS/VS transient data, including both intrapartition and extrapartition destinations (for example, sequential devices). Communicates with transient destinations through the destination manager (part of the CICS/VS executor).

APL139K

11. Passes user-supplied data from the shared storage manager to the session manager.

DIAGRAM 1.4: COMMUNICATION WITH TSO



Notes for Diagram 1.4

APLYUINI

1. The initialization process (module APLYUINI) performs the following functions at VS APL startup:

   Gets space for and initializes the executor global table. This table (mapped by the APLTSOGL macro) contains the PERTERM, terminal buffers, switches and pointers. It is always pointed to from all VS APL tasks from each task's TCBFSA field.

   Scans the invocation parameters and sets session values. Loads any auxiliary processor modules.

   Allocates space for shared memory, auxiliary processor work areas (512 bytes per auxiliary processor), and the workspace. Gives back FREESIZE amount to TSO.

   Determines if VS APL microcode assist is to be used.

   Calls the shared storage manager to initialize any auxiliary processors, including the session manager task and the GDDM task.

   Initializes pointers and keys in the incore workspace.

   Determines if the CONTINUE workspace is to be auto-loaded.

   Determines if terminal is display (with or without session manager) or typewriter and initializes accordingly.

   Sets SPIE, STAE, STAX (attention) exits.

   Places service request YYON in PTHYYCOD and passes control to SCAPL. From there, control is passed to the interpreter at its entry point APLIINIT.

APLYUERR

2. Asynchronous handling applies to program checks, attention exits, and abends.

   Program checks (SPIE exit): [SCSPIE]

   These are handled in module APLYUERR, routine SCSPIE (except during VS APL startup, when it is handled by routine SPIEXIT in module APLYUINI).

   Routine SCSPIE does the following:

   Saves the program check registers and PSW in WSMSURGS and WSMSUPSW (in the workspace).

   If the program check occurred in supervisor code, prints messages APL102I, APL104I, APL105I, APL106I, and abnormally terminates VS APL with the user code 1xx, where xx is the program check code in decimal.

   If the program check occurred in the shared storage manager or an auxiliary processor, prints message APL114I and handles the check as an interpreter program check.

   If the program check occurred in the interpreter, checks to see if the interpreter is in a program check loop and prints message APL101I and ABEND if in a loop. If not in a loop, moves the registers from WSMSURGS to WSMREGSV and the PSW from WSMSUPSW to WSMPCPSW and WSMNSI.

   If a program check loop occurs, the registers and the PSW for the next-to-last program check will be in WSMREGSV and WSMPCPSW/NSI, and the registers and PSW for the last fatal program check will be in WSMSURGS and WSMSUPSW.

   Attention exits (STAX exit): [SCATTN]

   These are handled by routine SCATTN in module APLYUERR, which does the following:

   Sets attention bit(s) in the PERTERM.

   If the attention is pressed while an auxiliary processor is executing, the auxiliary processor is terminated. (The purpose is to break endless or uncontrolled loops.)

   If the attention is pressed while auxiliary processors and VS APL are in deadlock, the deadlock is broken. (An auxiliary processor has issued a wait without first posting any other auxiliary processor or VS APL for work.)

   Returns to point of interrupt.

   Abends (STAE exit): [SCSTAE]

   These are handled in module APLYUERR by routine SCSTAE, which does the following:

- Prints messages APL115I and APL103D.

- Attempts to save a CONTINUE workspace.

- Terminates VS APL.

APLYUFXI

3. Service request handling: [routine APLFXIIM]

   Service requests allow the interpreter to interact with its environment (for example, type a line, load a workspace). Any module in the interpreter may issue a service request. The linkage is:

```
L         R1,=V(APLFXIIM)
BALR      R0,R1
DC        AL2(YYCODE)
```

   Routine APLFXIIM is in executor module APLYUFXI. It does the following for every service request:

   - Saves the general registers in WSMREGSV (except for YYDUMP, which preserves the contents of WSMREGSV), the floating registers in WSMREGF0, F2, F4, F6, and the address of the caller's resume point (R0+2) in WSMNSI. (All WSM fields are in the workspace.)

   - Adds the processor time used by the interpreter to an accumulated-time field (CMSCPUAC) for the quad-AI system variable.

   - Stores the YYCODE (which determines the type of request) in the PERTERM (field PTHYYCOD).

   - Looks up the request type in YYTABL (module APLYUFXI) and gets the address of the execution routine.

   - Calls the execution routine to execute the service request. The execution routine will return with the service request return code set in PTHSRCOD.

   - Updates the WSMASYNC bits in the workspace to reflect the latest status of asynchronous events (for example, attention).

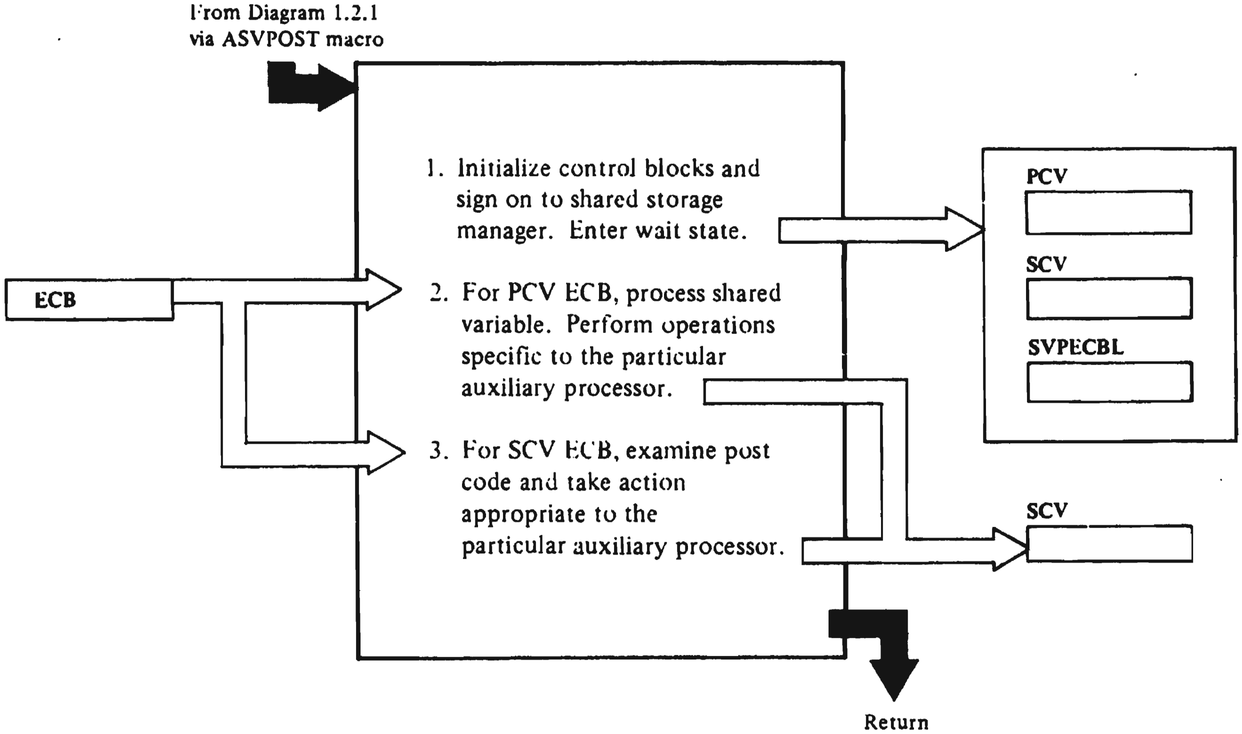
   - Sets the current time in CMSHOLDT so that processor time for the interpreter can be accumulated for quad-AI.

   - Goes back to interpreter in PSW key X'D' at its entry point, APLIINIT. There, the interpreter's environment is restored and control is returned to the instruction following the service request.

For a description of the service request codes and the names of the TSO executor routines that handle them, see "Values, Parameters, and Return Codes for Service Requests" under "Service Request Calls" in "Section 6. Diagnostic Aids."

For shared variable processing, control is passed to APLYURVC and then to ASVPSERV to route the request to the appropriate routine in the shared storage manager. See Diagram 1.2.1: "Shared Storage Manager (CMS and TSO)." After control returns to ASVPSERV, each auxiliary processor whose wait has been satisfied receives control. Control is then returned to the interpreter.

DIAGRAM 1.4.1: AUXILIARY PROCESSORS (TSO)



Notes for Diagram 1.4.1

1. Initialize

Initialization is performed as follows: The process control vector is completely filled in. The SCVID and SCVECB fields of the shared control vectors are filled in. The addresses of the ECBs are placed in SVPECBL. The PCV ECBSW switch is set in SVPECBL to identify the PCV ECB. The auxiliary processor signs on to the shared storage manager.

The auxiliary processor waits for an ECB to be posted. [WAIT]

APLYU100: TSO COMMAND

2. For PCV ECB

When control is returned from the wait state and the PCV ECB is posted, the following occurs:

An SCV is assigned to be used by the shared variable. [SCVLOOP]

A query is issued to find the variable's name. [DOQUERY]

A counter-offer is issued to complete the sharing of the variable. [QUERYSUB]

The variable is referenced. [GETNXVAR]

The type of command to be executed (TSO) is determined. [APLYUCMD]

After verifying the command, a call is made to CMDAP0 (entry point in APLYUUSR) to confirm authority for user to execute command.

The TSO command is ATTACHed.

The auxiliary processor waits for an ECB to be posted. [WAIT]

3. For SCV ECB

When control is returned from the wait state and an SCV ECB is posted, the post code is examined. [CHKPSTCD]

Processing then occurs as follows:

If the partner referenced the variable, the return code is specified. [RCODE]

If the partner set the access control vector this event is ignored. [CLRPSTCD]

If the partner retracted the variable, the variable is retracted and the SCV is made available for another variable. [RETRACT]

If the partner specified the variable, the variable is referenced and the command is executed. [GETNXVAR, TRANZCOD]

The return code is specified. [RCODE]

The auxiliary processor waits for an ECB to be posted. [WAIT]

APLYU101: ALTERNATE INPUT

2. For PCV ECB

When control is returned from the wait and the PCV ECB is posted, processing occurs as follows:

An SCV is assigned to be used by the shared variable. [SCVLOOP]

A query is issued to find the variable's name. [DOQUERY]

A counter-offer to complete the sharing of the variable is issued. [QUERYSUB]

The variable is referenced. [GETNXVAR]

The stacking and conversion options are determined. [REFOK]

The auxiliary processor waits for an ECB to be posted. [WAIT]

When control is returned from the wait state and an SCV ECB is posted, the post code is examined. [CHKPSTCD]

Processing occurs as follows:

If the partner referenced the variable, the return code is specified. [RCODE]

If the partner set the access control vector, this event is ignored. [CLRPSTCD]

If the partner retracted the variable, the variable is retracted and the SCV is made available for another variable. [RETRACT]

If the partner specified the variable, it is referenced and converted. The line is stacked according to the options determined in step 2 above. [GETNXVAR]

The return code is specified. [RCODE]

The auxiliary processor waits for an ECB to be posted. [WAIT]

If remaining items are in the stack, a GETMAIN is issued followed by invocation of the STACK macro for TSO execution after APL has completed sign-off processing.

2. For PCV ECB

When control is returned from the wait state and the PCV ECB is posted, the following occurs:

If PCVESOFF was posted, APL102 signs off. [SIGN-OFF]

If SCVEOFFR was posted, then an offer is processed as follows: [OFFER]

If there is no free SCV, then APL102 returns to the wait state.

If the offered name is not of the form DAT... or CTL..., the offer is ignored. [REFUSE]

If the offered name is of the form CTL..., the access control is set. If the offered name is of the form DAT..., processing continues with the next step.

If a match to the offered name exists, the pair of variables (DAT... and CTL...) are cross-connected. [CHKPAIR]

If this is a CTL... variable, its initial value is referenced. If the reference is successful, the variable is processed at SPEC01 as if a partner had been specified. [OFFEROK]

Following this, APL102 returns to the wait state.

3. For SCV ECB

When control is returned from the wait state and an SCV ECB is posted, the following occurs:

If the partner is retracted, the current variable is retracted. [RETRACT]

If the partner is specified, processing takes place as follows: [SPECIFY]

If the variable is of the form DAT..., it is ignored and APL102 returns to the wait state.

If the variable is of the form CTL..., it is checked to see if it is paired, its value is referenced, and the main storage display is processed as requested. Storage display data is returned in DAT by the routine RETDATA.

Finally, a return code is set in CTL and SSM is called to specify CTL and DAT.

If the partner is referenced, processing occurs as follows: [REFER]

If CTL is referenced, the last return code is given. [REFER1]

If DAT is referenced, the return code is set to 5 (DAT referenced out of sequence). [REFER1]

The return code is specified and APL102 returns to the wait state.

APLYU111: QSAM

2. For PCV SCB

When control is returned from the wait and the PCV ECB is posted, processing occurs as follows:

An SCV is assigned to be used by the shared variable. [SCVLOOP]

A query is issued to find the variable's name. [DOQUERY]

A counter-offer to complete the sharing of the variable is issued. [QUERYSUB]

The variable is referenced. [GETNXVAR]

The conversion option is determined and the file name is placed in the DCB. [CHKPARM]

The auxiliary processor waits for an ECB to be posted. [WAIT]

3. For SCV ECB

When control is returned from the wait and an SCV ECB is posted, the post code is examined. [CHKPSTCD]

Processing continues as follows:

If the partner referenced the data variable, the file is read and the data converted according to the options determined in step 2 above. [PARTREF]

The converted data is then specified. [SPEC1]

If the partner referenced the control variable, the return code from the last operation involving the data variable is specified. [RETNCODE]

If the partner set the access control vector, this event is ignored. [CLRPSTCD]

If the partner retracted the variable, the file is closed, the variable is retracted, and the SCV is made available for another variable. [RETRACT]

If the partner specified the data variable, it is referenced and converted according to the options determined in step 2 above. [GETNXVAR,CONVERT]

The converted data is then written to the OS file. [WRITE]

If the partner specified the control variable, it is referenced and ignored. [CLRPSTCD]

The auxiliary processor waits for an ECB to be posted. [WAIT]

APLYU210: BDAM FILES

2. For PCV ECB

When control is returned from the wait and the PCV ECB is posted, processing occurs as follows:

An SCV is assigned to be used by the shared variable. [SCVLOOP]

A query is issued to find the variable's name. [DOQUERY]

A counter-offer to complete the sharing of the variable is issued. [QUERYSUB]

The variable is referenced. [GETNX1]

The conversion option is determined, and the file name is placed in a DCB. [GETDCB]

The auxiliary processor waits for an ECB to be posted. [WAIT]

3. For SCV ECB

When control is returned from the wait and an SCV ECB is posted, the post code is examined. [CHKPSTCD]

Processing continues as follows:

If the partner referenced the data variable, the file is read and the data converted according to the options determined in step 2 above. [PARTREF]

The converted data is then specified. [SPEC1]

If the partner referenced the control variable, the return code from the last operation involving the data variable, the read-pointer, the write-pointer, and the number of records to be processed are specified as a 4-element integer vector. [RETNCODE]

If the partner set the access control vector, this event is ignored. [CLRPSTCD]

If the partner retracted the variable, the file is closed, the variable is retracted, and the SCV is made available for another variable. [RETRACT]

If the partner specified the data variable, it is referenced and converted according to the options determined in step 2 above. [GETNXVAR,CONVERT]

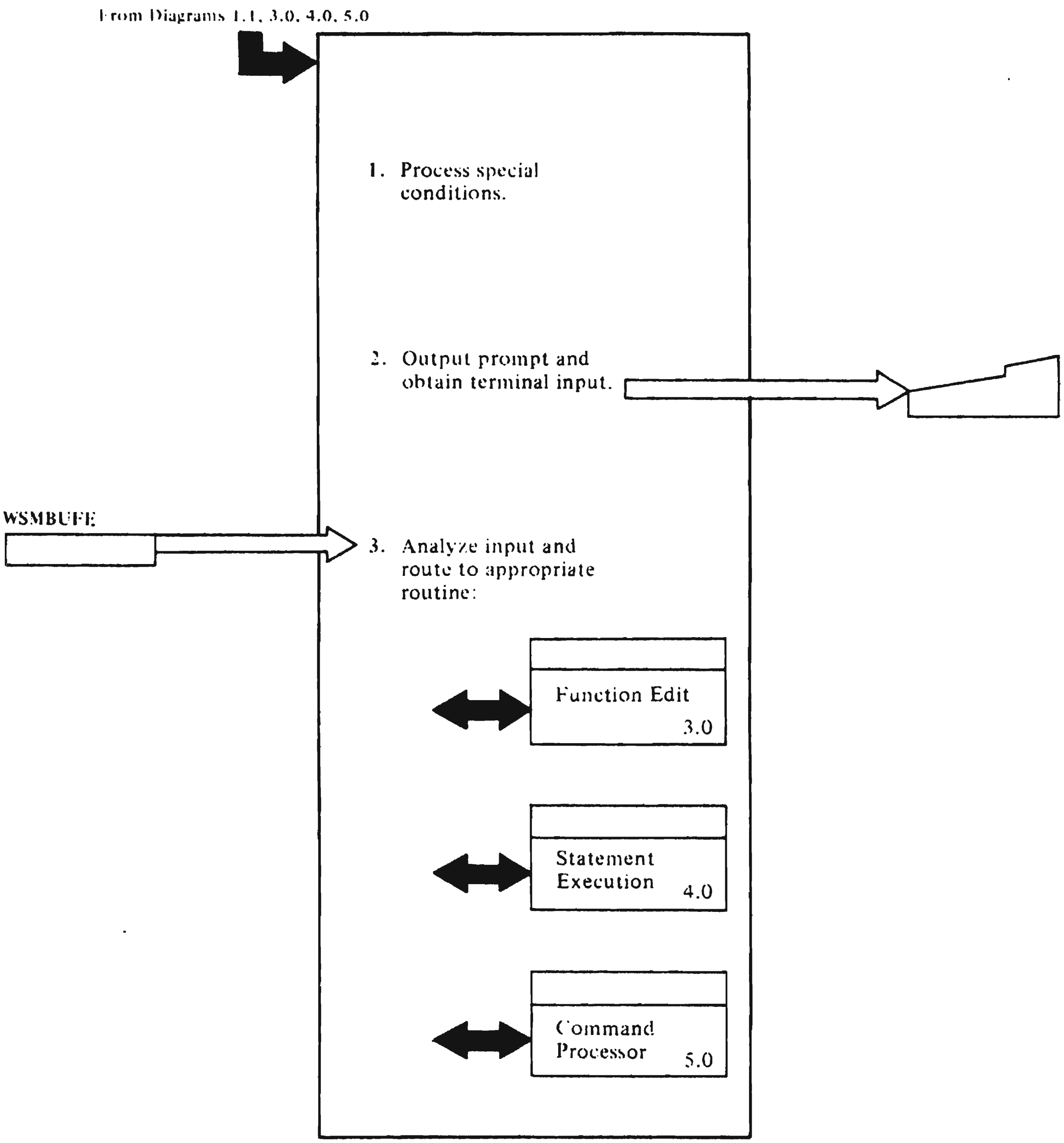
The converted data is then written to the BDAM file. [WRITE]

The auxiliary processor waits for an ECB to be posted. [WAIT]

APL123: VSAM

See Diagram 1.2.2. The same code is used for CMS and TSO.

DIAGRAM 2.0: INPUT RECOGNITION, TRANSLATION, AND ROUTING



Notes for Diagram 2.0

APLITINP

1. Before terminal input is requested, special conditions are checked for and processed as follows: [ITINPUT]

   If workspace is newly loaded and quad-LX is not null, the statement "execute quad-LX" is placed in WSMBUFF. Control is passed to the statement execution routine. [DOQLX]

   If force-off (PTHFOFF=1), a continue command is placed in WSMBUFF and control is passed to the command processor. [ITFORCOF]

   If the user's keyboard is normally locked, the YYRWAIT service request is issued. [DOWAITD]

   If attention or cancel-output is pending (PTHATTN=1, PTHDATTN=1, or PTHNOOUT=1), the YYATOFF service request is issued before terminal input is obtained. [DOATTN]

APLITINP

2. The user prompt is output, and input is obtained as follows: [GETINP]

   If workspace is in function definition mode (FDOPEN=1), a bracketed line number is built in WSMBUFF. The YYTYOI service request is issued to output the prompt and obtain input.

   If workspace is in quad-prime input mode (STQPBIT=1), WSMBUFF is filled with blanks up to the position indicated by PTHCURSR. The YYTYI service request is issued to obtain input.

   If workspace is in quad-input mode (STQBIT=1), a quad, colon, and new line character are placed in WSMBUFF. The YYTYO service request is issued to output the prompt.

   In all other cases, and following the output of the quad-input prompt, six blanks are placed in WSMBUFF. The YYTYOI service request is issued to output the prompt and obtain input.

APLITINP

3. The result of the YYTYI or YYTYOI is analyzed and processed as follows: [CHKINPUT]

   If input exceeded size of WSMBUFF, SPACE NOT AVAILABLE message is output, and processing is resumed at step 1. [ITTYIZ]

   If entry error, ENTRY ERROR message is output. Then YYTYOI is issued to output the line up to the point of error and obtain input. Processing is resumed at step 3. [ITTYIZ]

   If input is O-U-T, an interrupt exit is taken. [ITTYIZ]

   If any other error return from service request, a system error exit is taken. [ITTYIZ]

   If quad-prime input, control is returned to caller with input length in register 7.

   If input is null or all blanks, processing is resumed at step 1.

   If in function definition mode or if first non-blank is a del, the function edit and definition routine is called. (See Diagram 3.0: "Function Definition and Edit.")

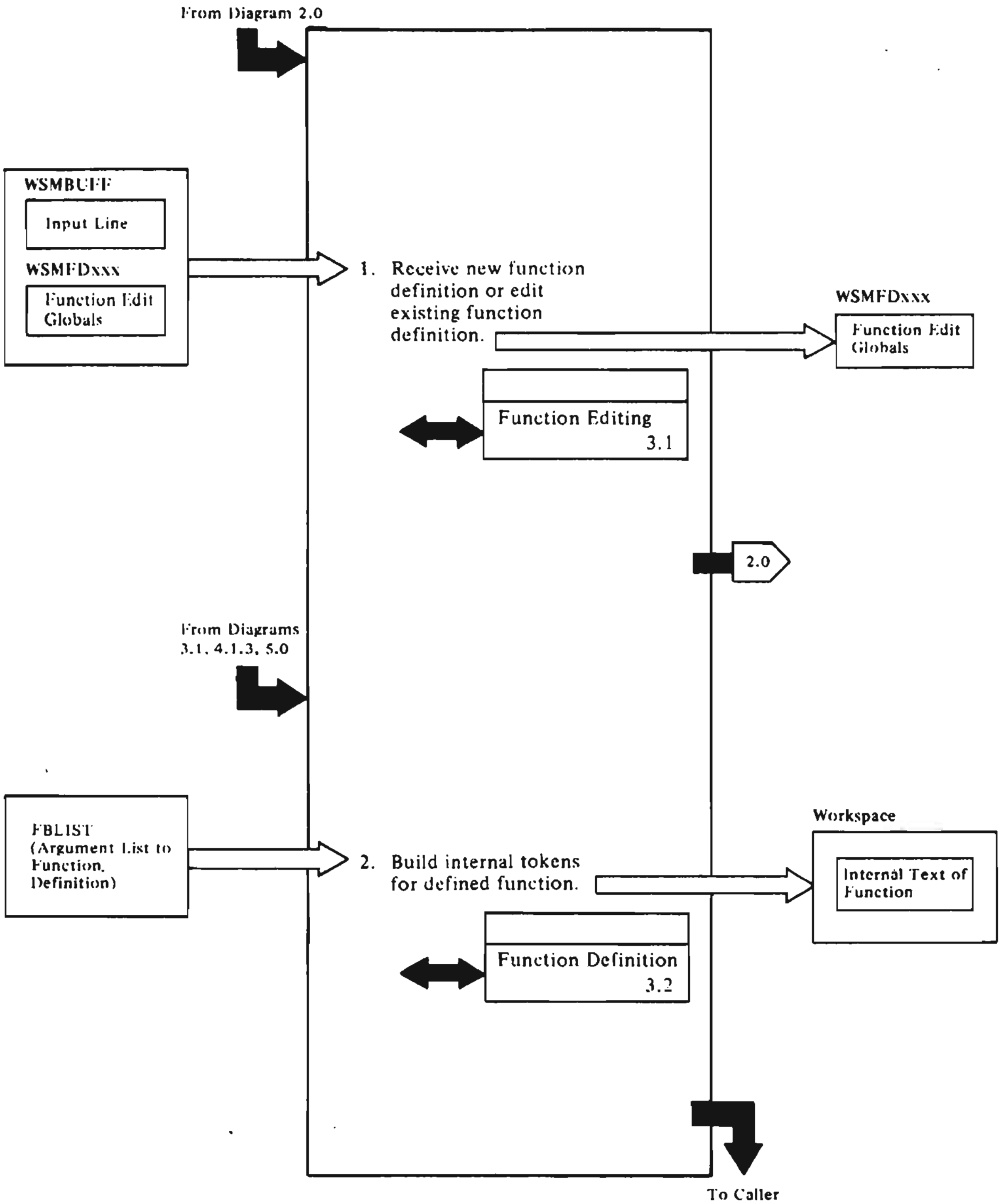
   If the first non-blank is a right parenthesis, command processor is called. (See Diagram 5.0: "System Command Execution.")

   If input is a comment, processing is resumed at step 1.

   For all other cases, ITEMPFUN is called to build an immediate execution temporary function whose single statement is the tokenized input line. See Diagram 3.2, step 2, for a description of tokenizing. The internal name of the function is returned in register 4. If quad-input, control is returned to the caller; otherwise, the statement execution routine is called. (See Diagram 4.0: Statement Execution.)

   Note that function definition and edit, statement execution routine, and command processor (Diagrams 3.0, 4.0, and 5.0) are called as subroutines. When they return control, processing is resumed at step 1.

DIAGRAM 3.0: FUNCTION DEFINITION AND EDIT



Notes for Diagram 3.0

APLITFDO

1. When a request to edit a function is received, the function-open routine receives a character-string beginning with a del or pdel character. [ITFDOPEN, ITLINE0]

   The routine validates the request; and if it is valid, puts the user's workspace in edit mode by setting a flag in WSMFDTOG, and a prompt-line number value in WSMFDLIN. If the function is a new one, APLITHDR is called to check its syntax. The header line is saved in character form.

   APLITINP

   While in edit mode, the user is prompted with a bracketed line-number. [ITINPUT]

   APLITFDE

   Once in definition mode, subsequent input strings are passed to the function edit routine. It performs the requested action and, assuming the definition is still open, sets a new value in WSMFDLIN. [ITFDEDIT]

   APLITFDC

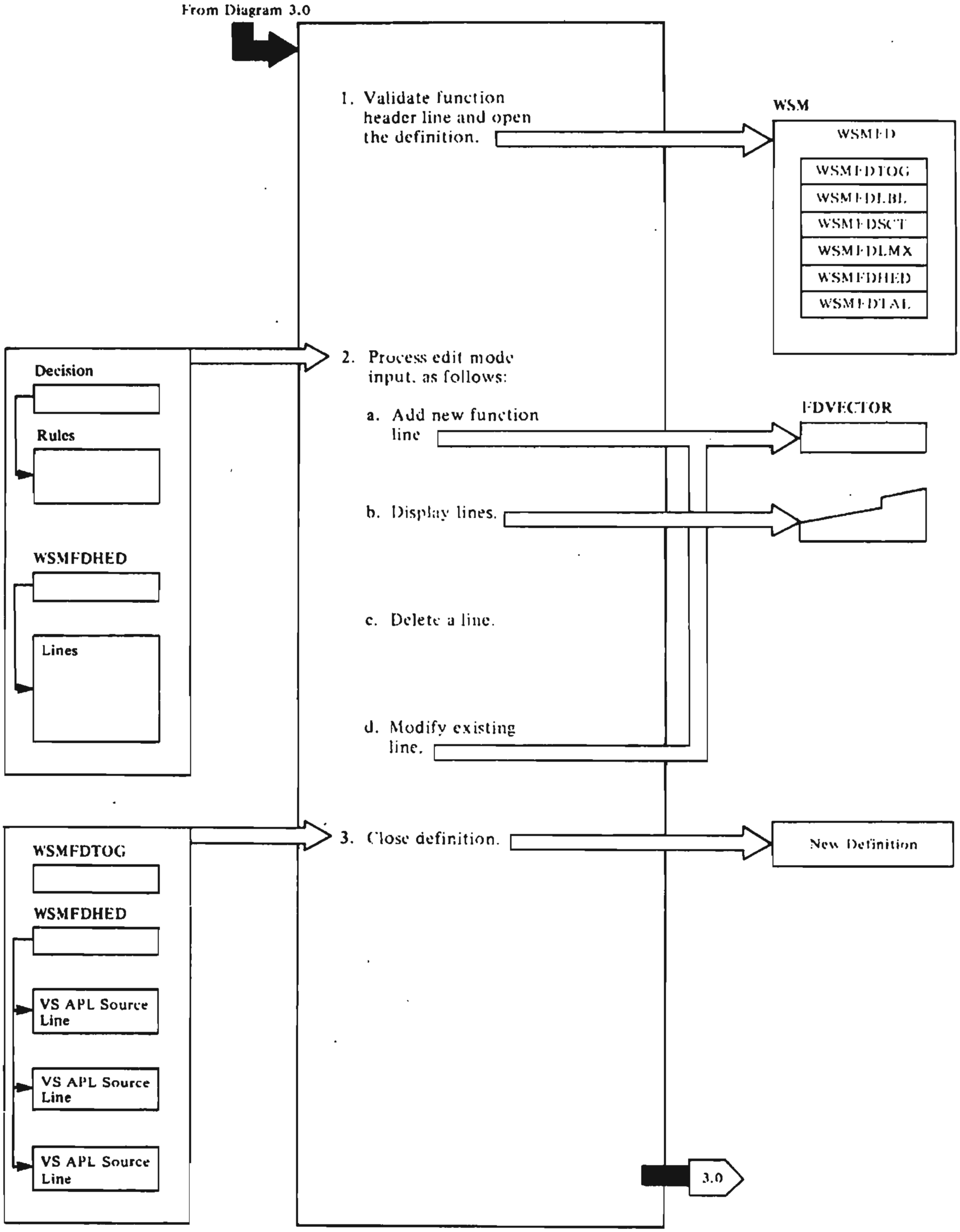
   New or replaced statements are saved in character form. If the edit request calls for closing the definition, the function close routine builds the internal text of the function and takes the user workspace out of edit mode. [ITFDCLOS]

APLITHDR

2. The function definition process is generalized so it may be called from function edit, from the COPY system command, or from the quad-FX appendage routine.

   Line 0 of the function is converted to internal form by APLITHDR. Each body line is tokenized by module APLITLXS. Module APLITFDC gets space for the function object and builds a tail entry for each statement in it.

DIAGRAM 3.1: FUNCTION EDITING



Notes for Diagram 3.1

1. The content of the header line is examined:

   APLITFDO

   The syntax is checked; the function name is isolated and converted to the internal name. [ITFDOPEN, ITLINE0]

   If the name is not globally defined, it is processed as a new definition: the line-zero syntax is checked (and rejected with DEFN ERROR if erroneous); the text of the line is saved in an FDVECTOR object; FDNEWFUN and FDOPEN are set in WSMFDTOG; and the edit globals are set to their initial values. For an existing, unlocked function, the definition is opened by setting FDOPEN in the WSMFDTOG flag to 1, and the edit globals to the values of the existing definition.

2. The function being edited consists of a set of specially formatted character vectors (FDVECTORS) in a chain whose head is named in the WSMFDHED field and whose tail is named in the WSMFDTAL field. Each input line is passed to APLITFDE in WSMBUFF, exactly as it appears at the terminal; that is, the prompt line-number forms part of the input.

   APLITFDE

   The input is scanned and each component of the edit syntax (bracketed line numbers, quad or delta symbols, closing del) noted in the EDSCAN01 byte of DECISION, a field in the R13 stack used by APLITFDE. [ITFDEDIT] If a new statement is encountered, it is collected and stored in an FDVECTOR. [ITFDNWLN, APLITFDN] The presence of a label is noted and all names used are entered in the symbol table. [ITSTSRCH]

   DECISION now contains a value between 0 and 63, indicating the action to be taken.

APLITFDE

   a. A new line is added, as follows: If the line number to be processed is higher than the line number in WSMFDLMX, the new line is entered at the end of the chain. [ITFDEDIT]

   APLITHDR

   If the line number is zero (header line), the header syntax is validated, previous header line is deleted, and the new line inserted. [ITLINE0]

   APLITPRL

   b. Function lines indicated by the user are displayed. If the function being edited exists only as an internal function object and not in display format, the ITPRLINE routine is called to put the lines in the display buffer. In this case, the line numbers exist only as integers. [RULE09]

      If the function is in display format, the text vectors are moved to the buffer from the beginning number indicated until a line number exceeding the end number is found. [R9A]

   c. The indicated line is deleted. [RULE10]

   d. The line is modified, as follows: The specified line is found and displayed. Blanks are displayed to position the cursor as requested; the edit mask is saved; and the edited line built in the buffer. Backspaces are built to position the cursor to the first inserted blank (if any). Buffer contents are then displayed and the input, overstruck on the display, is obtained. The new line is then processed as in step b above.

APLITFDC

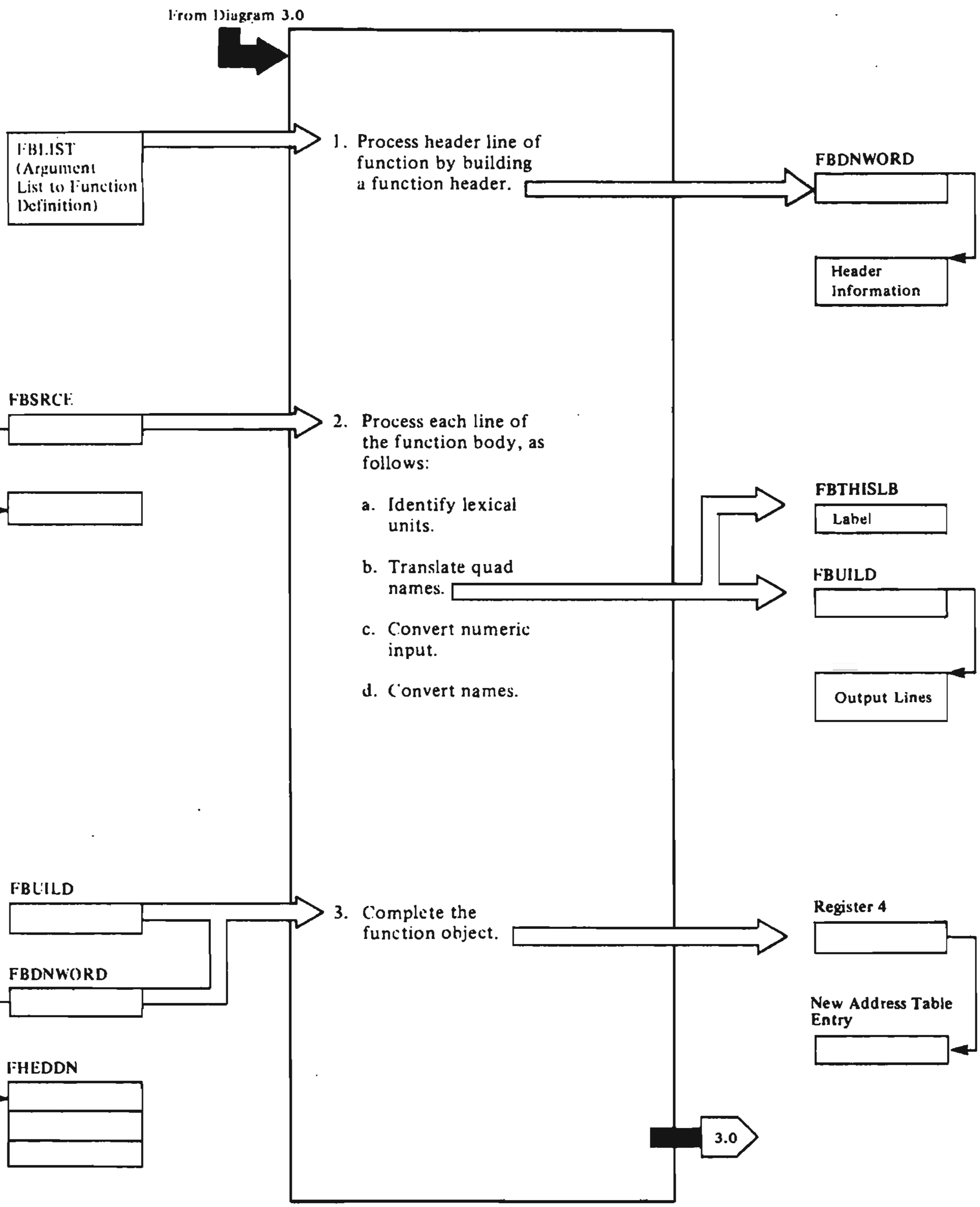
3. If the edit request calls for closing the definition, the function close routine builds the internal text of the function and removes the workspace from edit mode. [ITFDCLOS, CL2]

   Internal text is built, as follows: The FBLIST parameter block is prepared and each line is passed to the function definition routines (see Diagram 3.2: "Function Definition"). The WSMFDHED field contains the name of the function header text vector (FDVECTOR DSECT), and each line has the name of the next line. The last step of function definition returns a temporary internal name of the new function object. Note that text vectors are not deleted until the function has been completely created. [CL4A]

Changing line 0 of a suspended function or any part of a pendant function causes damage to the operation stack. In either case, a message is issued. Any existing function definition corresponding to the edited one, is freed. The new function object is assigned a permanent name. [CL4B]

The temporary address table entry is copied to the permanent one named by WSMFDOLD; the object DN-word is updated; and the temporary entry is freed. Edit mode is ended; all text vectors are freed and the WSMFDTOG flag is set to zero.

DIAGRAM 3.2: FUNCTION DEFINITION



Notes for Diagram 3.2

APLITHDR

1. Input is in or pointed to by the FBLIST DSECT prepared by the caller and addressed by register 2. The value in FBUILD is used to set the DN-word address of the function. The internal names for "function-name," "result," and "arguments" are entered in the function header. [ITLINE0]

   The number of internal names found determines the function syntax. The FBSYNT field is set to the values of the SBITFUN0, SBITFUN1, and SBITFUN2, respectively. [HDOVER]

   APLITHDR

   Local variable names are converted [ITSTRCH, APLITIDS] and are appended to the function header. [LOCALOOP]

   Operation stack space required to call the function is computed and entered in the function header. The offset from the beginning of the function to the first label position is set in the FBLBLOFF field. [LABELS]

   Four bytes are reserved for each label that will be encountered later, as given in FLABELS. [DONE]

   The end-of-locals mark is X'0002' or any halfword whose low-order bits are set to 10. If errors are found, register 0 is set to an abnormal termination code, and the ERROR1 exit to the caller is taken. [DEFNERR]

   The FBUILD and FBUILDL fields are set to reflect the space used by the header. [DONE]

APLITFDC, APLITLXS

2. As input, register 2 contains the address of the FBLIST DSECT prepared by the caller and updated as in step 1 above. [ITFDCLOS, ITOKENIZ]

   The string addressed by the FBSRCE field is examined and identified as either: identifier, numeric scalar, numeric vector, character scalar, character vector, primitive operator, or label. [SCAN]

   These are processed as follows:

   Identifier: an initial alphabetic signals a name. The symbol table is searched and an internal name is returned as follows: [IDENT]

   Internal names are found by the symbol table search: the ITBLDID routine isolates the name string, calculates its length, and enters these in the WSMNEWID field. [ITBLDID]

   APLITIDS

   Initial hashing to the symbol table index [WSMSYMX] combines the first 8 bytes of the name, its length value, and some prime numbers to get an index between zero and the value in the WSMSYMBL field. [ITSTSRCH]

   Each symbol table entry is a pair of adjacent address table entries, one for the name of an object, the other for its current value. The symbol table is searched for a match to the name in the WSMNEWID field. If a match is found, the internal name of this entry is returned via register 4. If a match is not found and the caller wants the name entered in the table (WSMISC.STCREATE=1), the namestring is put in the free space as a character vector. If WSMISC.STCREATE=0, a code of 0 is returned. If entry of the namestring causes the symbol table to become full, or, if there is not enough space in the workspace for the character vector containing the namestring, an error code is returned (that is, register 0 is set to ABSTFU or ABWSFU).

   Initial T or S causes a test for the diagnostic trace and stop vectors. [SDLETA, TDELTA]

   APLITIDS

   If an initial quad starts a distinguished name (shared variable or primitive system function, APLITLXS), the character part of the name is entered in the WSMNEWID field and the APLITQVB table is searched for a match. [ITBLDQD]

   Internal names corresponding to system variables and operation tokens corresponding to primitive system functions (as defined in the APLIOPERC macro), are returned in register 4.

APLITLXS

A quad symbol not beginning a distinguished name is treated as a primitive. [QUAD]

APLITNCV

Numeric scalar: An initial numeric signals the start of a numeric literal. The literal is scanned and converted to internal form by the numeric input conversion routine. The routine has three entry points: ITININT for conversion from typed integer constant; ITFDCVT for conversion of a typed line number from a function definition; ITNUMCVT for conversion of numeric constant character strings. Each entry point sets the WSMNCVSW switch to indicate the kind of output needed, that is, integer or floating point. [ITNUMCVT]

APLITLXS

If the absolute value of the literal exceeds 65K bytes or is real, a general scalar is built consisting of a halfword header followed by the value. [SCALAR, SSCALI, SSCALF]

Small integer values are encoded as immediate scalars in the format of an address-table immediate value. [S16BIT]

Numeric Vector: When a numeric is followed by another numeric, a vector numeric literal is built.

The first integer is examined for size: for 1 or 0, a boolean vector is begun. For greater than 1, an integer vector is begun. [VECTOR3,VECTOR]

For floating-point, a floating point vector is begun. Successive integers are converted to internal format as with a numeric scalar provided that they are of the same type as the initial one. [STORE]

If a value appears that requires more space than the previous ones, all values are converted to the larger size. [VTEST, CVT1]

Note: An invalid numeric literal (ITNUMCVT returns WSMNCVSW.NCVFAIL) causes the statement to be encoded as an ill-formed line. Numeric literals that are larger than 7E75 (ITNUMCVT returns WSMNCVSW.NCVOFLOW or WSMNCVSW.NCVUFLOW) are specially encoded to cause a VALUE ERROR at execution. [NERROR]

Character scalar and character vector: An initial quotation mark denotes a character literal. [CHARLIT]

The null character is replaced by the internal name for a constant null string. [CEND]

For a 1-byte character scalar, an immediate character literal is built. [CEND2]

Primitive operators: Primitive operators are replaced by their internal codes found by indexing into the OPTAB table, using the graphic byte value. [PRIMITIVE]

Tests are made for correct use of the branch arrow and for balancing of parentheses and brackets. [GOTO, LBRACKET, RBRACKET, LPAREN, RPAREN]

Labels: When a colon is found, the name preceding it is put in the FBTHISLB field for later movement to the header line. [COLON]

The line is then inverted so that it can be scanned right to left for execution. [ENDLINE]

The FBUILD and FBUILDL fields are updated to reflect the space used. [EXITR2]

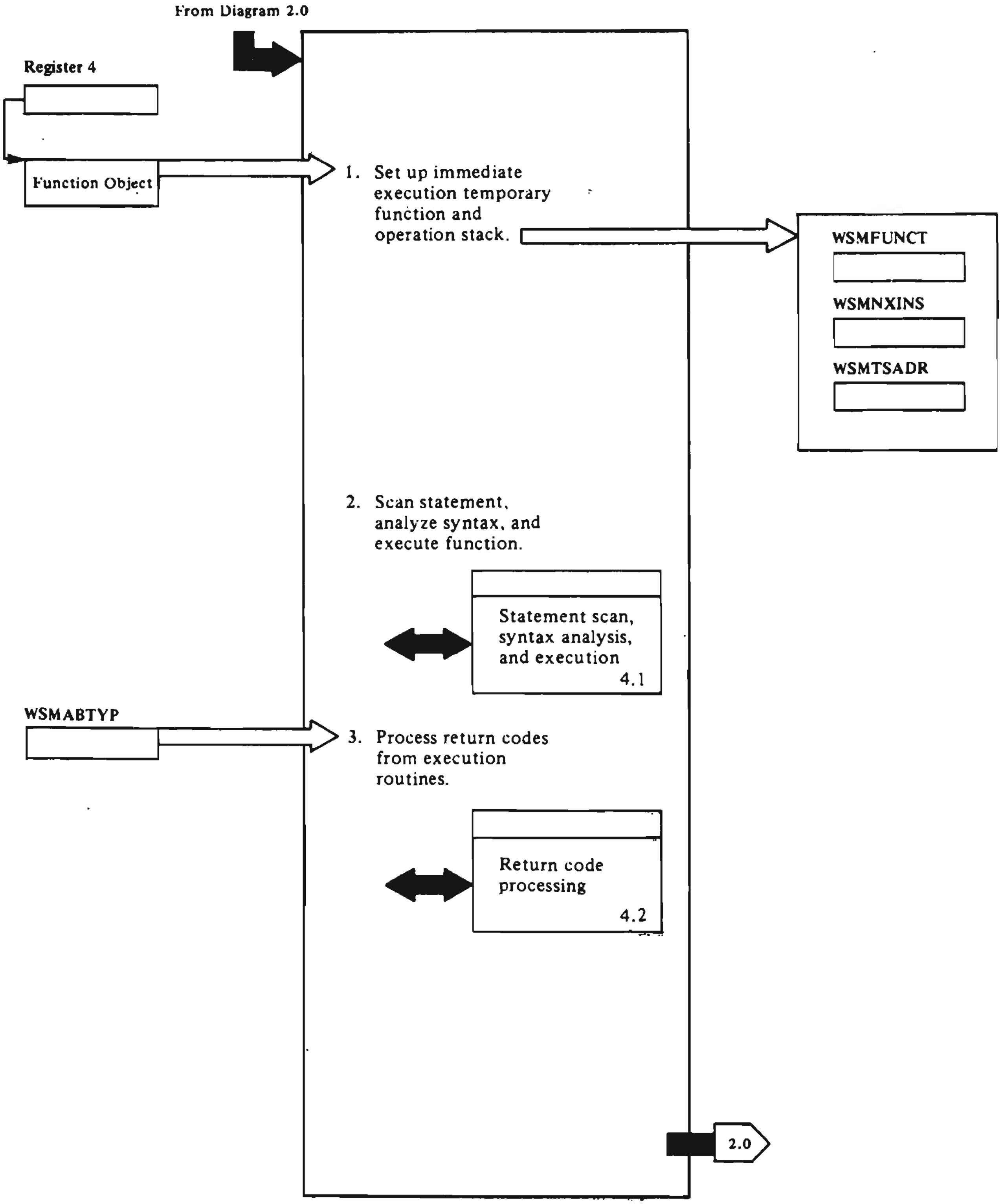
APLITFDC

3. Finally, the FHEDT field of FHEDDN is set to the displacement from the beginning of the function of each end-of-statement token of each line of the function. [ITCLOSET]

APLIESPA

Duplicate local variable names are marked. Space is obtained for the FBUILD and FBDNWORD fields and a temporary name for the function is created. [IESFIND]

DIAGRAM 4.0: STATEMENT EXECUTION



Notes for Diagram 4.0

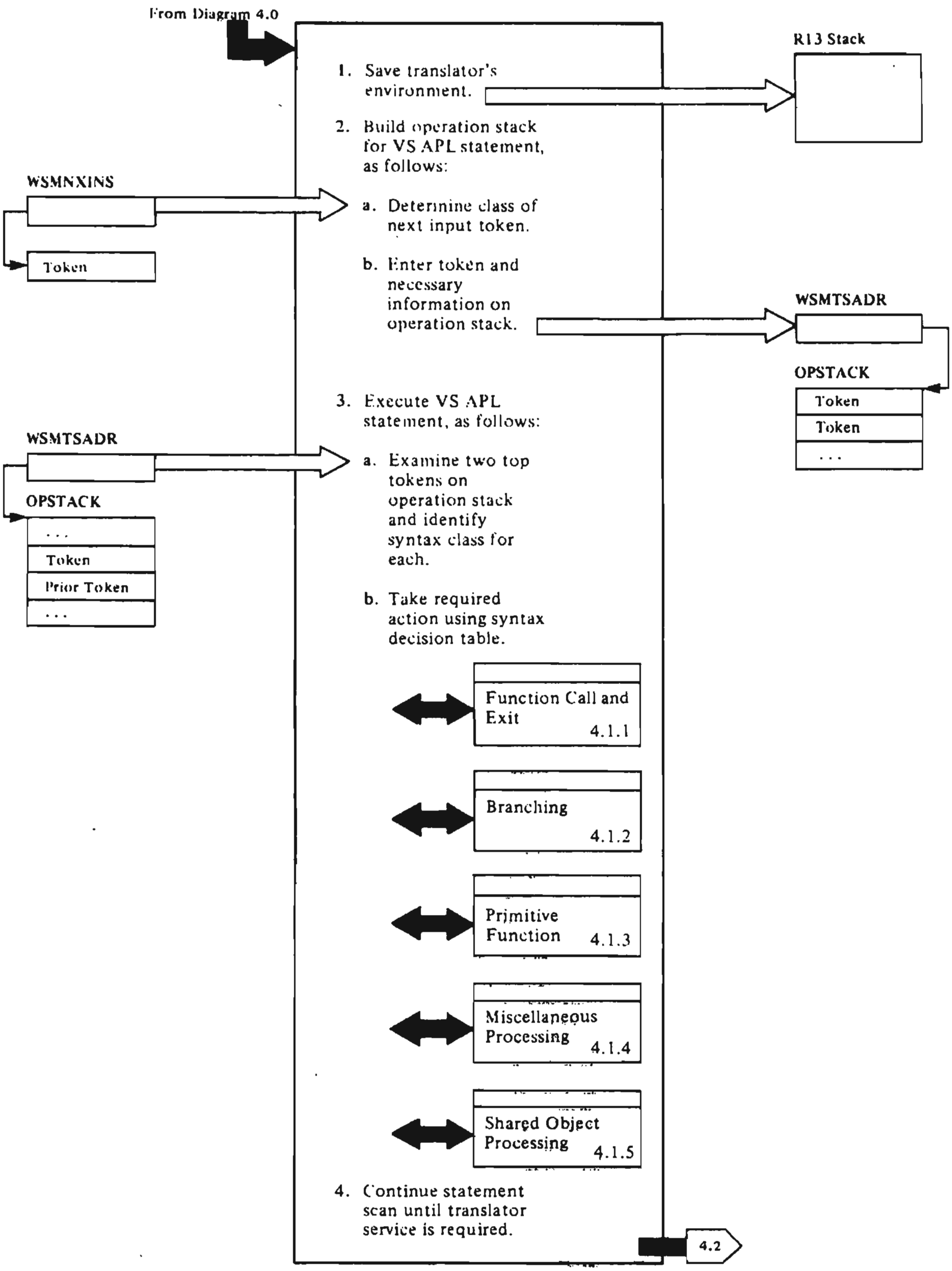
1. The name of the immediate execution temporary function is placed in field WSMFUNCT. The address of the first token is placed in WSMNXINS. A null token is placed on the operation stack, and WSMTSADR is set so that the null is the top token. [ITEXECUT]

2. Control is passed to the interpreter for statement scan syntax analysis and execution.

   The interpreter processes the function and any invoked functions, statement by statement. Control remains in the interpreter until a translator service is required. (See Diagram 4.1.)

3. The service indicated by the return code in WSMABTYP is provided (see Diagram 4.2). Control then is either returned to ITINPUT to obtain terminal input or again passed to the interpreter to resume statement execution. (See Diagram 4.1.)

DIAGRAM 4.1: STATEMENT SCAN, SYNTAX ANALYSIS, AND EXECUTION



Notes for Diagram 4.1

APLIEXAR

1. Save translator's environment: The translator's on-vector and registers 12 through 15 are saved in the R13 stack. (See "R13 Stack" in "Section 5. Data Areas".) The syntax of the top token on the operation stack determines the processing that is to occur. If the token is null, statement scan occurs (step 2). For other cases, syntax analysis occurs using the top two tokens on the operation stack (step 3). [IEXARCH]

   Reentry conditions to the interpreter are as follows:

   a. An escape exit for an ill-formed line, an error exit, or a "nothing to do" exit to the translator was taken and terminal input was obtained. The top token on the operation stack is a null token. The WSMNXINS field contains the address of the first token of an immediate-execution temporary function whose body is the tokenized terminal input.

   b. A stop, trace, print, or attention exit to the translator was taken at the end of the prior statement; or an escape exit for assignment to a trace or stop vector was taken and the next token was EOS. The top token on the operation stack is a null token. The WSMNXINS field contains the address of the first token of the next statement in the current function.

   c. The end of the only statement of a quad-input or execute temporary function caused exit to the translator, because the trace bit is always set to 1 in the EOS token of these functions. At exit, the operation stack was: EOS, result of quad-input or execute, null, function call block (FCB) for temporary function, prior token. The operation stack is now: result, prior token. The WSMNXINS field contains the address of the token following the quad or execute in the calling (now current) function.

   d. A branch in a quad-input or execute temporary function caused exit to the translator. For quad-input: The translator took an error exit and reentry is as in a above. For execute, the branch is to be evaluated in the context of the pendant function. At exit, the operation stack was: EOS, fast or normal branch operator (the argument of a normal branch), null, FCB for execute temporary function, prior token(s), FCB for pendant function. The operation stack is now: normal branch, argument of branch, prior token (null), FCB for pendant (now current) function. The WSMNXINS field contains the address of the token following the execute token in the calling function. [EOS]

   e. Assignment to a trace or stop vector caused exit and the next token is not EOS. The operation stack was: escape token, left arrow, right argument, prior token. The operation stack is now: right argument, prior token. The WSMNXINS field contains the address of the token following the escape token in the current function.

   f. Initial entry from the translator is as in step a above.

APLIESCA

2. Build operation stack for VS APL statement:

   a. The token whose address is in the WSMNXINS field is identified as one of the following: internal name, operator or separator, literal, fast branch, escape special operator, indirect special operator, comment, or system function. [IESCANG, ACTION0]

   b. The token is entered on the operation stack, that is, it is placed in the word whose address is in WSMTSADR. (See "Operation Stack" in "Section 5. Data Areas".) Entering takes place as follows:

Internal name: The name is placed in the right half of the stack word. The syntax and primary descriptor from the address table are placed in the left half. [ACT0]

Operator or separator: The token is put on the operation stack duplicated in the left and right halves of the stack word. If the operator is overstruck with a hyphen, the operator index value of 0 is placed in the fourth byte of the stack word. [ACT01]

Literal: For 16-bit literal, the token is put on the stack as a stack immediate variable. For other literals, a temporary internal name and a block of free space are obtained; the descriptor and value are put in the block; and the internal name is put on the stack with the syntax of a temporary remote variable. [ACT0LIT]

Fast branch: The token and the following token (end-of-statement) are put on the stack and the branch processing routine is called. (See Diagram 4.1.2: "Branch Processing.") [ACT0SP]

Escape special operator: This token indicates an ill-formed line or assignment to a stop or trace vector. The token is put on the operation stack as a stack immediate variable. An escape exit to the translator is taken. [ACT0SP2]

Indirect special operator: This operator is used in embedded VS APL functions. The next token containing the internal name of a primitive operator is obtained. The operator is then obtained from the address table, and put on the stack duplicated in the left and right halves of the stack word. [ACT0SP3]

Comment: The WSMNXINS field is set to the address of the token following the comment. Statement scan is resumed at step 2a. [ACT0SP5]

System function: The token is put on the stack in the right half of the word. The quad-q operator is put in the left half of the word. [ACT0SP6]

APLIESCA

3. Execute VS APL statement:

   a. The WSMNXINS field is set to the address of the token following the one processed. The WSMTSADR field is decremented by four. The token just entered on the operation stack now becomes the top token or the current token. [DECIDE]

   b. The action to be done is selected according to the syntax class of the two top tokens on the operation stack. [DECIDE2]

   Syntax class codes are as follows:

Code	Meaning
0	Null
1	Operator
2	Variable
3	Dyadic function
4	Right parenthesis or bracket
5	Left parenthesis or bracket
6	Semicolon
7	Assignment (left arrow)
8	Right operator index bracket
9	Niladic function
A	End of statement (EOS)
B	Monadic function
C	Shared object (quad, quote-quad, system variable, shared variable)
D	Not used
E	Not used
F	System object (group, printname)

   The syntax decision table which follows is used to determine the appropriate action.

Syntax Class Codes (Prior Token \ Current Token)				3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	1	0	1	0	1	1	1	1	5	10	1	11	1	1	1
1	1	3	2	4	17	4	4	4	4	18	4	4	19	1	1	1
2	1	0	1	0	1	8	16	0	9	1	10	5	1	1	1	1
3	1	1	5	1	17	1	1	1	1	18	1	1	19	1	1	1
4	1	1	0	1	0	14	14	1	1	5	1	1	11	1	1	1
5	1	12	6	1	17	1	1	1	1	18	1	1	19	1	1	1
6	1	1	0	1	0	14	14	1	1	5	1	1	11	1	1	1
7	1	1	7	1	13	1	1	1	1	1	1	1	15	1	1	1

Explanation: The actions symbolized by the action codes are as follows:

Code	Action
0	Continue statement scan. [IESCANG, ACTION0]
1	Syntax error. Exit to translator. [IESCANG, ACTION1]
2	Do dyadic operation (see Diagram 4.1.3: "Primitive Function Processing"). [IESCANG, IEDYAD]
3	If the prior token is a slash or backslash, do reduction or scan operation (see Diagram 4.1.3: "Primitive Function Processing"). For other cases, do Action 4. [IESCANG, ACTION3]
4	If the current token is a period, do inner or outer product operation; for other cases do monadic operation (see Diagram 4.1.3: "Primitive Function Processing"). [IESCANG, ACTION4]
5	Do function call (see Diagram 4.1.1: "Function Call and Function Exit Processing"). [IEFUNN]
6	Do subscripting operation (see Diagram 4.1.4: "Miscellaneous Processing"). [IEINDD]
7	Do assignment (see Diagram 4.1.4: "Miscellaneous Processing"). [IESCANG, ACTION7]
8	If current token is left bracket, continue statement scan. If current token is left parenthesis, operation stack is: left parenthesis, variable (result of parenthesized expression), right parenthesis, prior token. Modify operation stack so that it is: variable, prior token. Select next action (see step 3b above). [IESCANG, ACTION8]
9	Change syntax class of current token from 8 (operator index bracket) to 4 (right bracket). Then do Action 16. [IESCANG, ACTION9]
10	Process end of statement (see Diagram 4.1.4: "Miscellaneous Processing"). [IESCANG, ACTION10]
11	Do shared object reference (see Diagram 4.1.5: "Shared Object Processing"). [IESCANG, ACTION11]
12	Operation stack is: operator, left bracket, variable (operator index), right bracket, prior token. Get value of operator index and put it in fourth byte of stack word containing operator; set explicit indexed operator bits (OPHASIND and OPEXIND). Modify operation stack so that it is: operator, prior token. Continue statement scan. [IESCANG, ACTION12]
13	Operation stack is: right separator, left arrow. Set SSASGN bit to 1 in right separator to indicate subscripted assignment. Then do Action 17. [IESCANG, ACTION13]
14	Operation stack is: semicolon or left bracket, semicolon or right bracket. Modify operation stack so that it is: semicolon or left bracket, empty subscript marker, semicolon or right bracket. Continue statement scan. [IESCANG, ACTION14]

15 Do shared object specification (see Diagram 4.1.5: "Shared Object Processing"). [IESCANG, ACTION15]

Note: Actions 16 through 19 are done when the current and prior tokens are such that there may be a named permanent variable on the operation stack that has not yet been evaluated. Before it is evaluated, a new value may be assigned to the name. To provide consistent right-to-left execution, the value of a named variable when it is encountered in the statement scan must be preserved. If the variable in question is temporary or stack immediate, nothing is done. In any other case, a copy or synonym of the value with a temporary internal name is made; the permanent name on the stack is replaced with the temporary name.

16 Copy prior token (see note above). Then continue statement scan. [IESCANG, ACTION16]

17 Copy third token (see note above). Then continue statement scan. [IESCANG, ACTION17]

18 Copy third token (see note above). Then do function call. [IESCANG, ACTION18]

19 If current token is other than quad, do Action 11. For other cases, copy third token (see note above). Then do Action 11. [IESCANG, ACTION19]

4. Continue statement scan until translator service is required.

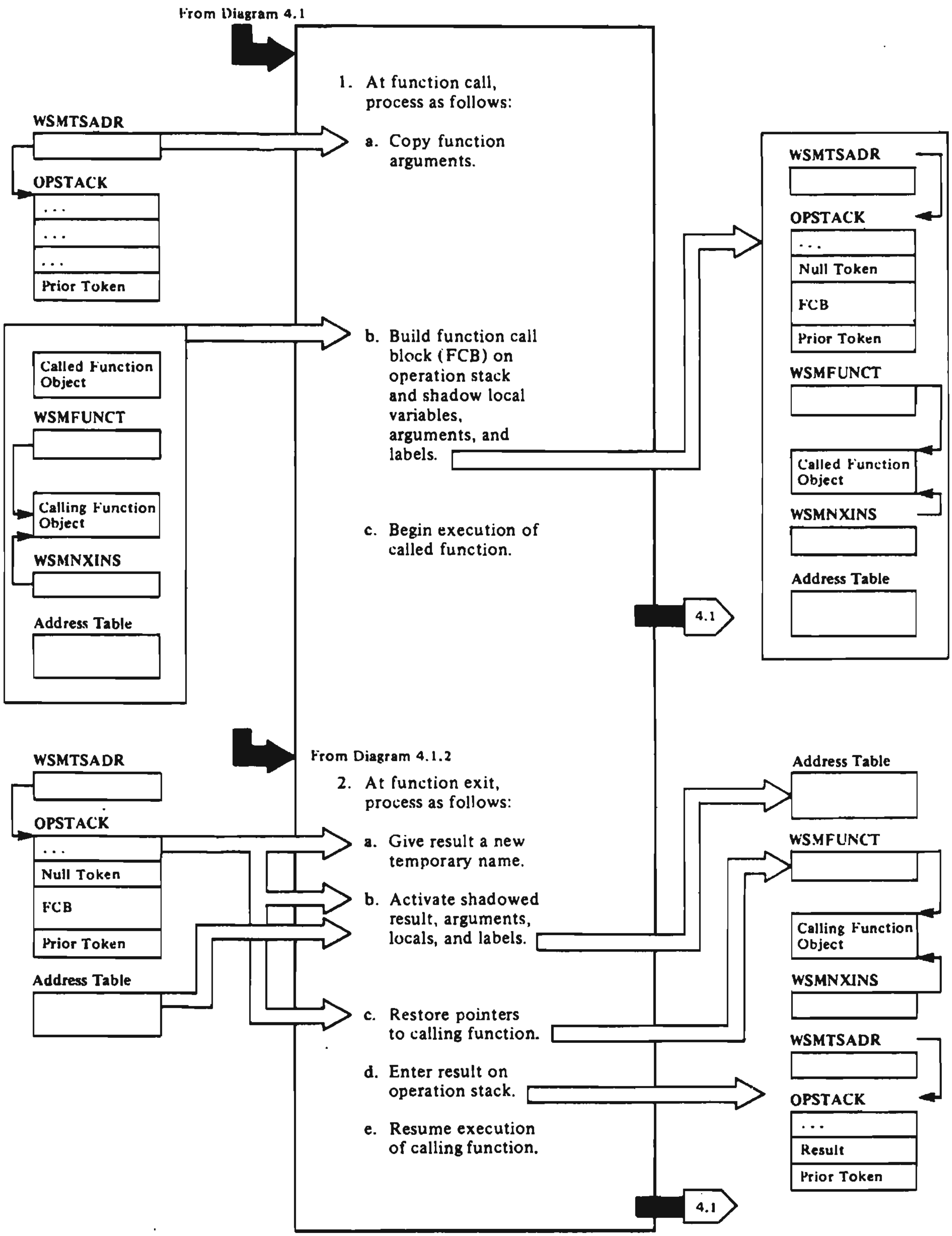
All actions described above eventually terminate in one of three ways:

With a return to the translator for one of the following reasons: an error is discovered; stop, trace, print, or attention service is required; or the operation stack is exhausted. The reason code is passed in field WSMABTYP (see Diagram 4.2).

With control passed to IESCANG-ACTION0 to continue statement scan (step 2).

With control passed to IESCANG-DECIDE2 to do syntax analysis (step 3).

DIAGRAM 4.1.1: FUNCTION CALL AND FUNCTION EXIT PROCESSING



Notes for Diagram 4.1.1

1. At function call, the operation stack is in one of the following conditions:

   Left argument, dyadic function, right argument, prior token.

   Monadic function, right argument, prior token.

   Niladic function, prior token.

APLIEFNM

a. A copy is made of the arguments, giving them temporary internal names. This is done so that references to the arguments within the function are to their local values and not to their global values. [IEFUNN, FUNN1]

b. A function call block (FCB) is built on the operation stack overlaying the input tokens. The space required for the FCB is obtained from the FHEDK field in the called function header. The FCB is built as follows:

   The internal name of the calling function is obtained from the WSMFUNCT field and entered in the FCB; the internal name of the called function is entered in the WSMFUNCT field. [FUNN3]

   The offset to the next token in the calling function is computed and entered in the FCB. [FUNN3]

   The active referent of each variable named in the function header (that is, FHEDZ through FHEDLOCLn fields) is shadowed (that is, the global value is saved in the FCB, and an initial local value is assigned). Shadowing occurs as follows [FCLOOP]:

   The internal name and address table entry are entered in the FCB.

   The internal name in the value block and any associated synonym blocks are changed to that of the address table entry saved in the FCB.

   For system variables, the no-value and implicit-error bits (ATIMNOVL and ATIMERR) in the address table entry are set to 1. For quad-IO and quad-CT system variable, the implicit-error bits (SWQIOIMP and SWQCTIMP) in WSMASYNC are set to 1. For quad-HT system variables, null tab settings are sent to the executor.

   For labels, the statement number is entered in the address table entry with a syntax descriptor of X'2F11' (indicating a read-only variable with immediate integer value).

   For all other local names, X'2700 0000' (indicating a variable with no value) is entered in the address table entry.

   Translator flags, obtained from the FHEDBITS field, and the length of the FCB, are entered in the FCB. [FUNN4]

   Function arguments are activated by changing their address table entries from "no value" to the specified values using the copies described in step a above. The temporary names of the copies are discarded. [FUNN5]

   A null token is entered on the operation stack, and the WSMTSADR field is set to make the null the top token on the operation stack. [FUNN6]

   The WSMNXINS field is set to the address of the first token of statement 1 of the called function. [FUNN6]

c. Execution of the function is begun as follows:

   If the stop bit (EOSTPBIT) is not set for statement 1, control is passed to IESCANG, ACTION0 to resume statement scan. [FUNNXIT]

   APLIESCA

   If the stop bit is set, the WSMNXINS field is set to the address of the EOS token of statement 0. The EOS token is placed on the stack, and the WSMTSADR field is set to make the EOS the top token on the operation stack. A "stop" exit to the translator is taken. [IESCANG, ENTRY12]

2. Function exit processing is as follows:

   a. If the function has a result, the result is given a new temporary internal name, and its real address table entry is set to "no value." This is done so that the shadowed referent can be activated without destroying the result. [IEUNFN, UNFN]

   b. The shadowed referent of each local variable named in the function call block (FCB) is activated. [UNLOOP]

      Processing occurs as follows:

      For a system variable: the IAUNSHAD routine is called to unshadow the system variable. [Called by IASHRPST]

      For a shared variable: the IARTRACT routine is called to retract the shared variable. [Called by IASHRPST]

      For a remote value: the value block is freed.

      The address table entry saved in the FCB is reentered in the address table. The internal name is reentered in the value block and in any associated synonym blocks.

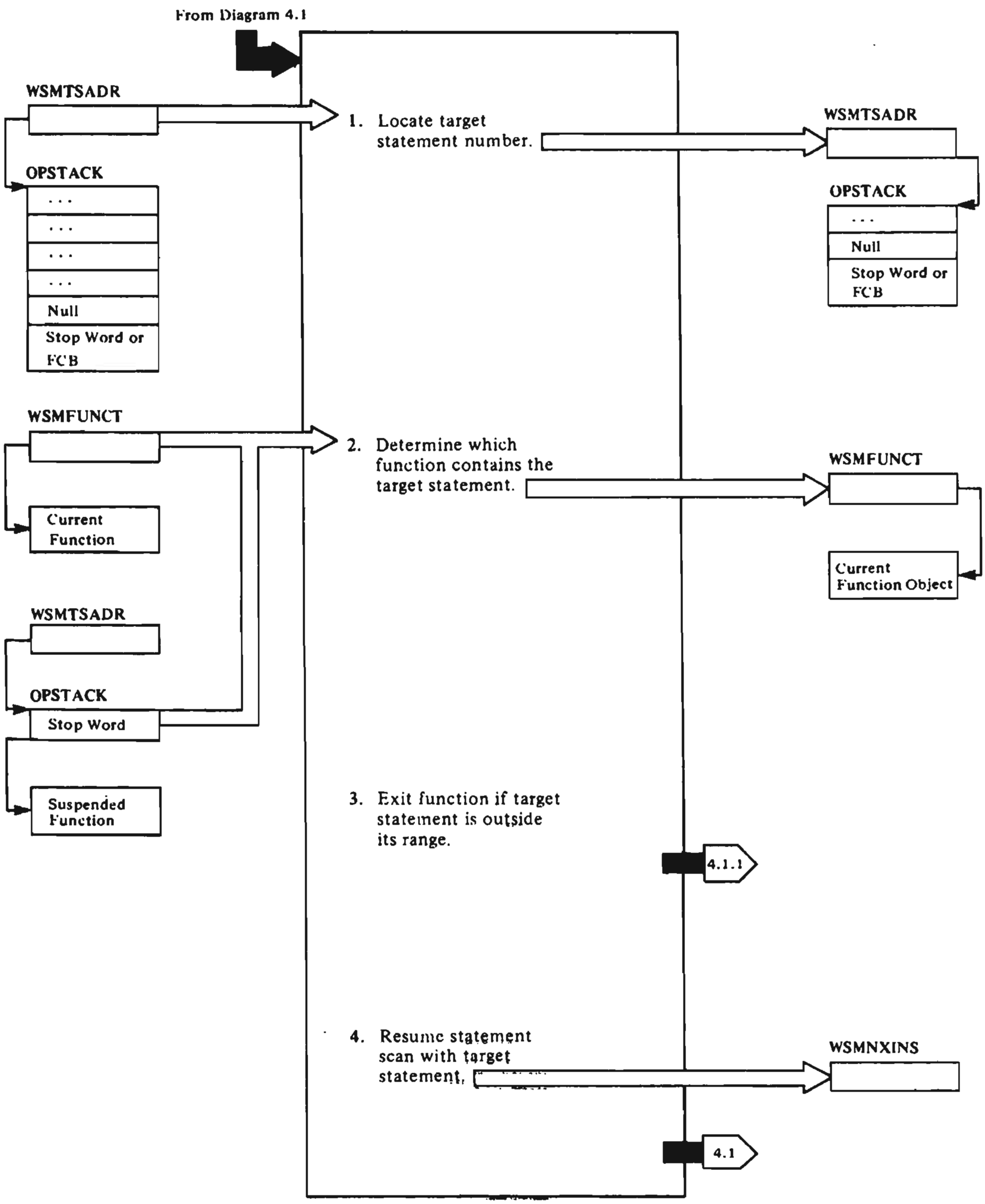
   c. The calling function is set as the current function by moving its internal name from the FCB to the WSMFUNCT field. [UNFN3]

      If the WSMFUNCT field indicates that damage has been done to the calling function, an SI DAMAGE error exit to the translator is taken. For other cases, the input pointer is set to the address of the token following the function call by obtaining the offset to the next token from the FCB, computing the address of the token, and entering the address in the WSMNXINS field. [UNFN4]

   d. The function result is placed on the operation stack following the token that preceded the FCB. The WSMTSADR field is set to make the function result the top token on the operation stack. If the function has no result, the constant WSMNOVAL (variable with no value) is used as the result. [UNFN5]

   e. Execution of the calling function is resumed with a syntax analysis of the result and prior token. [IESCANG, ENTRY11]

DIAGRAM 4.1.2: BRANCH PROCESSING



Notes for Diagram 4.1.2

APLIEFNM

1. For permanent functions with trace requested (OPTEMPGO=0 and EOSTRBIT=1) or quad-input or execute temporary functions (OPTEMPGO=1 and EOSTRBIT=1), control is passed to the translator, where all processing occurs. [IEGOGOMN, PRINT]

   Target statement is determined, as follows:

   For fast branch operator: Operation stack consists of these tokens: EOS, operator, null, stop word, or beginning of function call block (FCB). Target statement number is bits 1 through 11 of the branch operator. The WSMTSADR field is set to make the null the top token on the operation stack. [IEGOGOSC]

   For normal branch operator: Operation stack consists of EOS, operator, right argument, null, stop word or beginning of FCB. Processing is as follows:

   For permanent function with null argument: End-of-statement processing occurs (see Diagram 4.1.4).

   For immediate-execution temporary function with user-coded branch (OPTEMPGO=1 and EOSTRBIT=0) with null argument: The suspended function statement number in the stop word is the target statement number. [GOGOMN3]

   For all other cases: The target statement number is the first or only element of the argument. [GOGOMN4]

   If the argument is temporary, its internal name and block of free space are freed. [GOGOMN5]

   The WSMTSADR field is set to make the null token the top token on the operation stack. [GOGOMN6]

2. The internal name of the current function is obtained from the WSMFUNCT field, and the current function is located. If it is a permanent function, it contains the target statement. If it is an immediate execution temporary function, the function that contains the target statement is located as follows:

   The temporary internal name of the immediate-execution temporary function and its block of free space are freed. [GOGO]

   If there is no suspended function (that is, the stop word indicates the end of stack condition), control is returned to the translator. [NORMEX]

   If there is a suspended function but it is damaged (indicated as such by the stop word), the SI DAMAGE error exit is taken. [GOTOEX]

   If there is a suspended function (not damaged), the internal name of the suspended function is obtained from the stop word and entered in the WSMFUNCT field; the suspended function is now the current function. The stop word is replaced with a null token, and the WSMTSADR field is set to make the null the top token on the operation stack. [GOGO]

3. If the target statement is not within the range of statements in the current function, processing is as follows:

   If there is no pendant function (operation stack item preceding the null token is a stop word), control is returned to the translator. [NORMEX]

   If there is a pendant function (operation stack item preceding the null token is the beginning of a function call block), control is passed to the function exit routine (see Diagram 4.1.1: "Function Call and Function Exit Processing"). [GOGO4]

4. If the target statement number is greater than 0 and no greater than the number of statements in the current function, processing is as follows:

APLIESCA

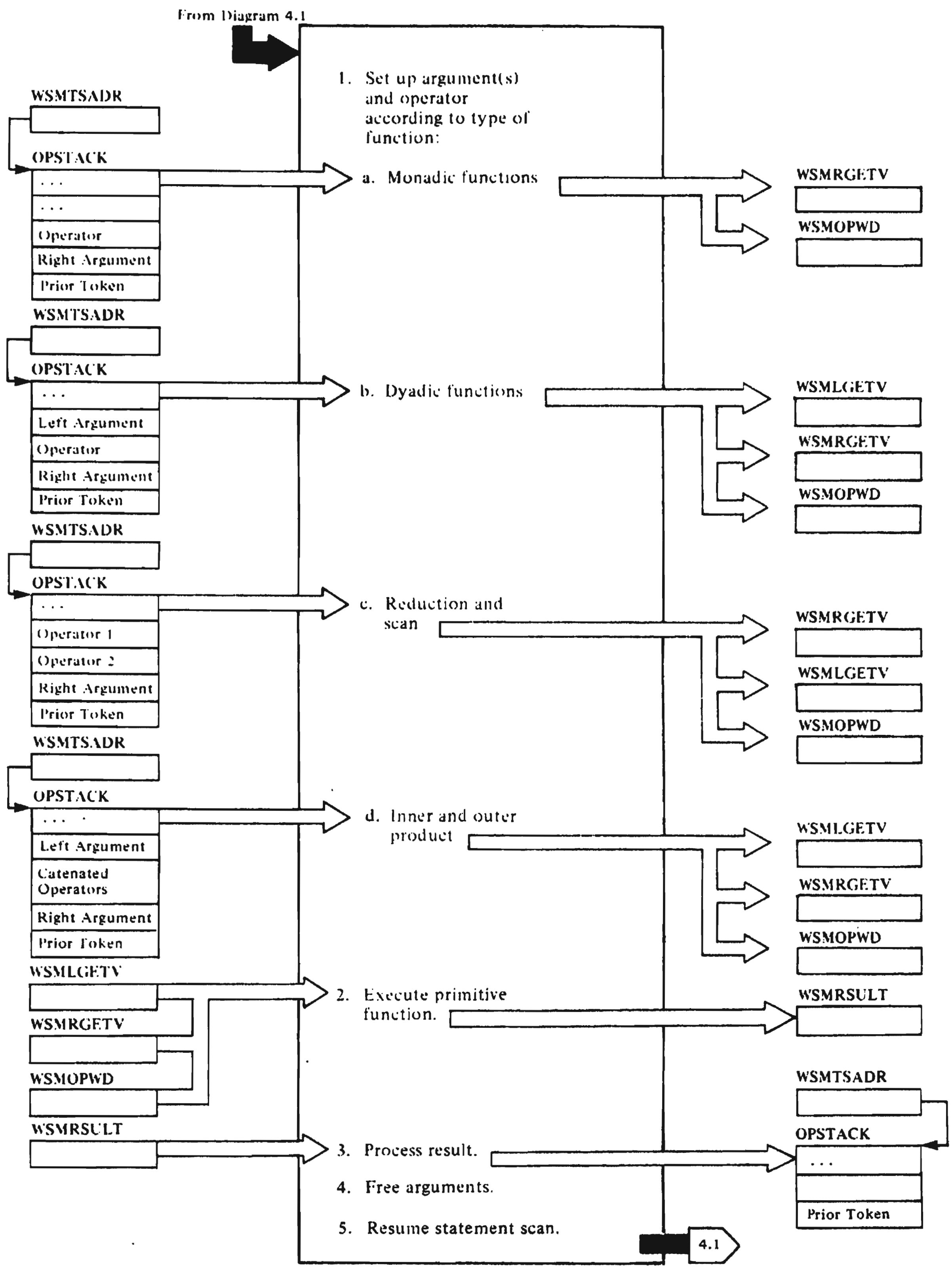
   If stop has been requested (EOSTPBIT=1), the WSMNXINS field is set to the address of the end-of-statement (EOS) token of the statement preceding the target statement. The EOS token is put on the operation stack, and the WSMTSADR field is set to make the EOS the top token on the operation stack. Control is passed to the translator via the stop exit. [IESCANG, ENTRY12]

APLIEFNM

If attention has been signalled, the WSMNXINS field is set to the address of the EOS token of the statement preceding the target statement. Control is passed to the translator via the attention exit. [ATTNEX]

For other cases, the WSMNXINS field is set to the address of the first token of the target statement. Control is passed to IESCANG, ACTION0 to resume statement scan. [GOGOXIT]

DIAGRAM 4.1.3: PRIMITIVE FUNCTION PROCESSING



Notes for Diagram 4.1.3

APLIESCA, APLIEMND

1. The argument(s) and operator are processed as follows according to the type of primitive function:

   Monadic functions: The right argument entry is obtained from the operation stack, and the IEGETV routine is called to set up the right argument block (WSMRGETV) for data fetching. The primitive function (operator) is obtained from the operation stack and placed in field WSMOPWD. [IEMONAD]

   Dyadic functions: The left argument entry is obtained from the operation stack, and the IEGETV routine is called to set up the left argument block (WSMLGETV) for data fetching. The right argument and operator are processed as described above. [IEDYAD]

   Reduction and scan functions: The right argument is processed as described above. The reduction or scan operator (OP2) is obtained from the operation stack and placed in field WSMOPWD. The primitive function (OP1) is obtained from the operation stack and placed in the left argument block. [IESCANG, ACTION3]

   Inner and outer product operations: The operators are catenated into one word on the operation stack. For inner product, the stack word contains (in bytes): dot (period), dot, OP1, OP2. For outer product the stack word contains: dot, jot (small circle), OP2, OP2. Then statement scan and execution continue until the operation stack contains: left argument, catenated operators, right argument. These are then processed as described above for routine IEDYAD. [IESCANG, ACTION4]

2. Control is passed to some routine (see below) to perform the function. In general, an operator routine computes the shape and size of the result, obtains a temporary internal name and a block of free space, builds the result, enters the syntax and internal name of the result in the WSMRSULT field, and passes control to a result-processing routine (see step 3). Exceptions to this are:

   Normal branch operator: exit to routine IEGOGOMN; see Diagram 4.1.2.

   Operations that are completed by subscripting: ⌽B, ⍉B, and A⍉B, when B is an array; and A⌽B, when A is scalar are performed as B[R] where R is a subscript list built by the operator routine. In these cases, the routine builds a subscript list in free space, and enters its descriptor and internal name in the right argument block.

   Operations that return a function as the result: the result of the execute operation is a temporary niladic function whose body is the tokenized right argument. The result of certain cases of encode and decode is a dyadic embedded VS APL function. (See "Operation Stack" in "Section 5. Data Areas".) The result of certain cases of scan is a monadic embedded VS APL function.

   The processing of all monadic functions begins in routine IEMONAD. The monadic functions and the routines that perform them are:

Function	Routine
+B	IEMONAD, PLUS
-B	IEMONAD, NEG; performed as 0-B
×B	IEMONAD, SIG
÷B	IEMONAD, RECIP; performed as 1÷B
⌊B	IEMONAD, FLCL
⌈B	IEMONAD, FLCL
*B	IEMONAD, EXP; performed as e*B
⍟B	IEMONAD, EXP; performed as e⍟B
∣B	IEMONAD, MAG
!B	IEMONAD, FACT
?B	IEMONAD, ROLL
○B	IEMONAD, PI; performed as pi×B
~B	IEMONAD, NOT; performed as 0=B
⍴B	IEMONAD, SIZE

Function	Routine
,B	IEMONAD, RAVEL
⍳B	IEMONAD, IOTA
⍋B	IAGRADE
⍒B	IAGRADE
⌽B	IEMONAD, REV;
⊖⌽B	IAREVARY if argument is an array
⍉B	IAMTRAN
⌹B	IAMDOM
⍎B	IAEXECTE
⍕B	IAMFORM
⎕××B	(System functions): IADSHARE

The processing of all dyadic functions begins in routine IEDYAD. The dyadic functions and the routines that perform them are:

All dyadic scalar operations: (+ - × ÷ ⌊ ⌈ * ⍟ | ! ○ ∧ ∨ ⍲ ⍱ < ≤ > ≥ = ≠): IEDYB

Function	Routine
A?B	IADEAL
A⍴B	IERSHP
A,B	IECOMMA
A⍳B	IEEPSIOT
A∊B	IEEPSIOT
A↑B	IETKDP; IATKDP if A is nonscalar
A↓B	IETKDP; IATKDP if A is nonscalar
A/B	IECMEX
A\B	IECMEX
A⌽B	IAROTA
A⍉B	IADTRAN
A⊥B	IADECODE
A⊤B	IAENCODE
A⌹B	IADDOM
A⍕B	IADFORM
A⎕××B	(System functions): IADSHARE

The composite functions and the routines that perform them are:

Function	Routine
op/B	IAREDU
op\B	IASCAN
A∘.op B	IEDYB

The outer product function is executed in one of two ways according to the shape of the arguments. If either argument is scalar, the function is done as an ordinary dyadic scalar function. Otherwise, the function is done as a series of dyadic scalar functions using the right argument and successive elements of the left argument for each iteration. The latter case is identified by bit OPISMIX = 1 in WSMOPWD.

A op1.op2 B IAIPROD

The inner product function is executed in one of two ways according to the shape of the arguments. If one argument is a vector and the other is an array, or if both arguments are arrays, the function is done by routine IAIPROD. Otherwise the function is done in two steps. First the dyadic scalar function A op2 B is performed by routine IEDYB. Then the op1 reduction of the result is performed by routine IAREDU.

3. Processing of the result varies according to its type. Exarch operator routines pass control directly to the appropriate result-processing routine. Appendage operator routines set WSMAFLG2 to indicate exceptional result types, and then return control to their calling routine (IEMONAD; IEDYAD; or IESCANG, ACTION3). Control is then passed to the appropriate result-processing routine.

Various types of results are processed as follows:

Operation is to be completed by subscripting (WSMAFLG2=AFLG2MOR+AFLG2TSP): exit to subscripting routine IEINDB (see Diagram 4.1.4).

Result is a niladic function (syntax class is 9): The syntax and internal name are transferred from WSMRSULT to the operation stack, replacing the right argument entry, and the WSMTSADR field is set to make it the top token on the operation stack. Exit to function call routine (see Diagram 4.1.1). [IESCANG, ENTRY5A]

Result is a monadic function (syntax class is B): The syntax and internal name are transferred from WSMRSULT to the operation stack, replacing the operator entry, and the WSMTSADR field is set to make it the top token on the operation stack. Exit to function call routine (see Diagram 4.1.1). [IESCANG, ENTRY5A]

Result is a dyadic function (syntax class is 3): The syntax and internal name are transferred from WSMRSULT to the operation stack replacing the operator entry. Exit to function call routine (see Diagram 4.1.1). [IESCANG, ENTRY2B]

Result equals right argument (WSMAFLG2=AFLG2MOR+AFLG2RT): If the right argument is an address table immediate value, it is placed on the operation stack as a stack immediate value. If the right argument is a remote value, it is marked as permanent, and the operation stack is left as is. [IESCANG, ENTRY9]

Result is a logical or integer scalar returned in register 2 by exarch operator routines: If the result is logical or a small integer, it is placed on the operation stack as a stack immediate value, replacing the right argument entry. If the result is a large integer, a temporary internal name and a block of free space are obtained; the value block is filled in; the syntax and internal name are placed on the operation stack replacing the right argument entry. [IESCANG, ENTRY3 or ENTRY6]

Result is a real scalar returned in floating-point register 4 by exarch operator routines: a temporary internal name and a block of free space are obtained; the value block is filled in; the syntax and internal name are placed on the operation stack replacing the right argument entry. [IESCANG, ENTRY4 or ENTRY7]

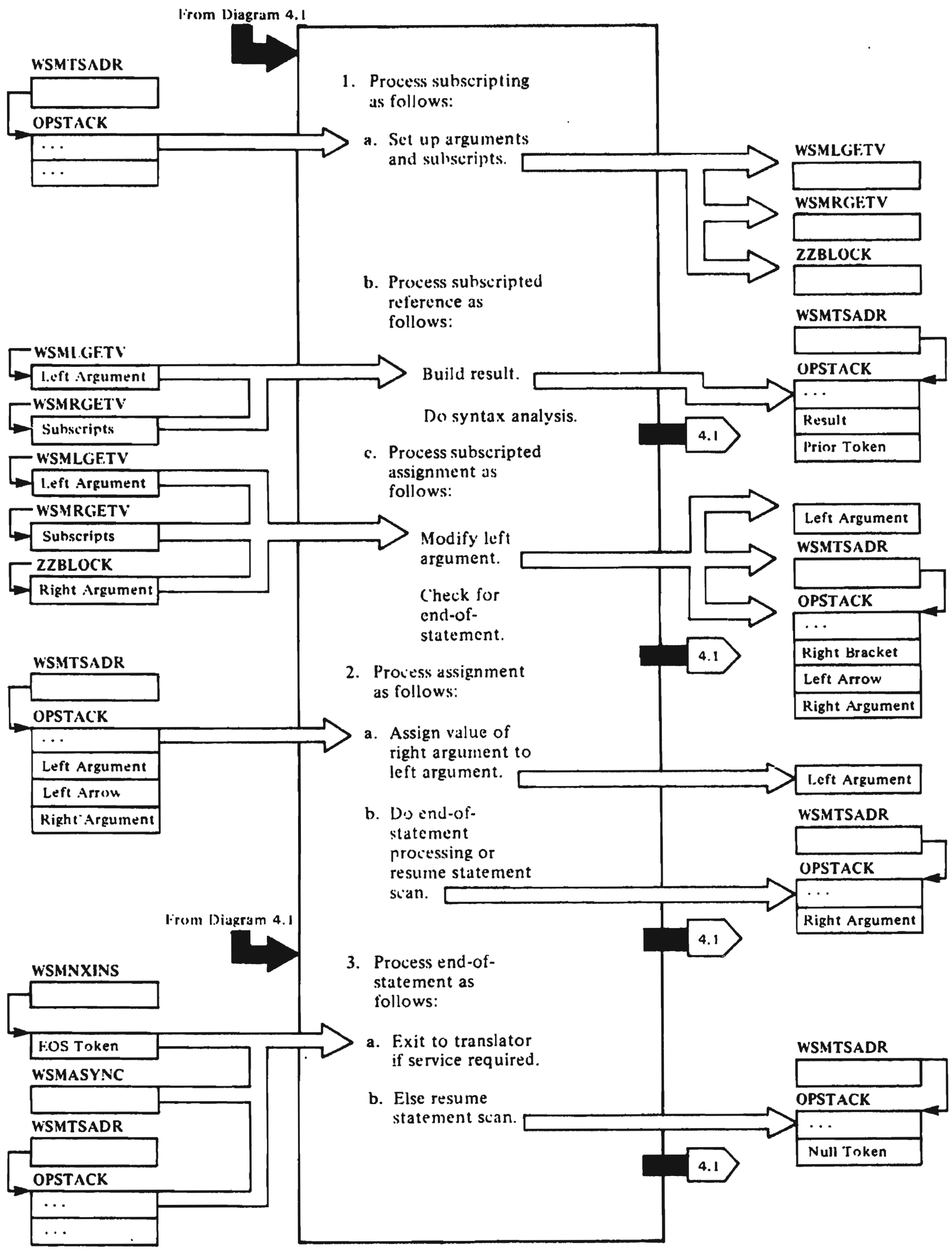
Result is a variable whose name or value is returned in field WSMRSULT: the syntax and internal name or immediate value are placed on the operation stack replacing the right argument entry. [IESCANG, ENTRY2 or ENTRY5]

For all cases in which the result is a variable, the WSMTSADR field is set to make the result entry the top token on the operation stack. [IESCANG, PUSHDOWN]

4. If the operation was dyadic and the left argument is temporary, its internal name and value block are freed. If the right argument is temporary, its internal name and value block are freed. [IESCANG, LFREE]

5. Control is passed to routine IESCANG, ACTION0 to resume statement scan.

DIAGRAM 4.1.4: MISCELLANEOUS PROCESSING



Notes for Diagram 4.1.4

APLIEIDX

1. The subscripting routine is called either to do ordinary subscripting or to complete a transpose or rotate operation.

   a. The arguments are set up as follows for each case:

      For ordinary subscripting:

      The operation stack contains the left argument followed by one or more subscripts. Each subscript is either a stack immediate value, the syntax and internal name of a remote value, or an empty subscript marker (indicating that all elements of the corresponding left argument dimension are selected).

      The left argument entry is obtained from the operation stack, and the IEGETV routine is called to set up the left argument block (WSMLGETV) for fetching of data. [IEINDD, SECTION1]

      If the left argument is a vector, there can be only one subscript. The subscript entry is obtained from the operation stack, and the IEGETV routine is called to set up the right argument block (WSMRGETV) for fetching of subscript data. If the subscript is an empty subscript marker, a temporary internal name and a block of free space are obtained, and a subscript in the form of an arithmetic progression (AP) vector is built. [IEINDD, SECTION4]

      If the left argument is an array, there is one subscript for each of its dimensions. A temporary internal name and a block of free space for a subscript list are obtained; the right argument block (WSMRGETV) is set up to address it. Each subscript entry is obtained from the operation stack, and the appropriate data is placed in the subscript list. The format of the subscript list is described in the listing of routine IEINDD. [IEINDD, SECTION2 and SECTION3]

      If the operation is a subscripted assignment, the operation stack also contains a right argument. Its entry is obtained, and the IEGETV routine is called to set up an argument block (ZZBLOCK) for fetching of data. [IEINDD, SECTION5]

      For subscripted assignment, the left and right arguments must be the same data type; the left argument must not be a synonym or arithmetic progression (AP) vector. If necessary, a copy of the left or right argument is made with the elements converted to the required data type. [IEINDD, SECTION6]

      For completion of transpose or rotate:

      The transpose or rotate operator routine has built a subscript list and placed its internal name in the right argument block (see Diagram 4.1.3 step 2). The IEGETV routine is called to set up WSMRGETV for fetching of subscript data. The left argument entry is obtained from the operation stack, and the IEGETV routine is called to set up the left argument block (WSMLGETV) for fetching of data. [IEINDB]

   b. Subscripted reference is processed as follows:

      A temporary internal name and a block of free space for the result are obtained. The name is placed in WSMRSULT. [IEINDD, SECTION8]

      The result is built by fetching elements of the left argument as indicated by the subscript(s) and placing them in the result block. [IEINDD, SECTION9 and SECT10]

      APLIESCA

      The syntax and name of the result are transferred from WSMRSULT to the operation stack, replacing the right bracket entry. The WSMT$ADR field is set to make the resulting entry the top token on the operation stack. [IESCANG, ENTRY10]

      Control is passed to IE$CANG, DECIDE2 to do syntax analysis using the result and the prior token. [IESCANG, ENTRY10B]

APLIEIDX

c. Subscripted assignment is processed as follows:

   Element,s of the left argument as indicated by the subscript(s), are replaced by successive elements of the right argument. [IEINDD, SECTION9 and SECT10]

   If the left argument is a shared or system variable, control is passed to the IASHRPST routine. The remainder of the processing is described in Diagram 4.1.5, step 3. [IEINDD, EXIT]

   APLIESCA

   The WSMTSADR field is set so that the right bracket is the top token on the operation stack. The result of the subscripted assignment is the right argument, not the modified left argument. A check is made for end of statement as described in step 2b below. [IESCANG, ENTRY10B]

2. Assignment is processed as follows:

   a. For assignment, left and right arguments are examined: If the left argument is temporary, a SYNTAX ERROR exit is taken. If it is read-only (a label), a DOMAIN ERROR exit is taken. If the left argument has a remote value, the space for its value block is freed. [IESCANG, ACTION7]

      The value of the right argument is assigned to the left argument, as follows:

      If the left and right value blocks are the same size and neither argument is a synonym, the right block is copied into the left block. [ACTION7]

      If the right argument has an immediate value, an address table immediate value is built for the left argument. [ACT7C]

      For other cases, a copy or synonym of the right argument is made and is given the internal name of the left argument. [ACT7E]

   b. If the next input token is EOS, control is passed to the end-of-statement processing routine.

      For other cases the WSMTSADR field is set so that the right argument is the top token on the operation stack. Control is passed to IESCANG, ACTION0 to resume statement scan. [ACT7X]

3. End-of-statement processing occurs as follows:

   a. An exit to the translator is taken if any of the following conditions are true:

      Trace is requested; in EOS token, EOSTRBIT=1. [IESCANG, ENTRY8 or ACTION10]

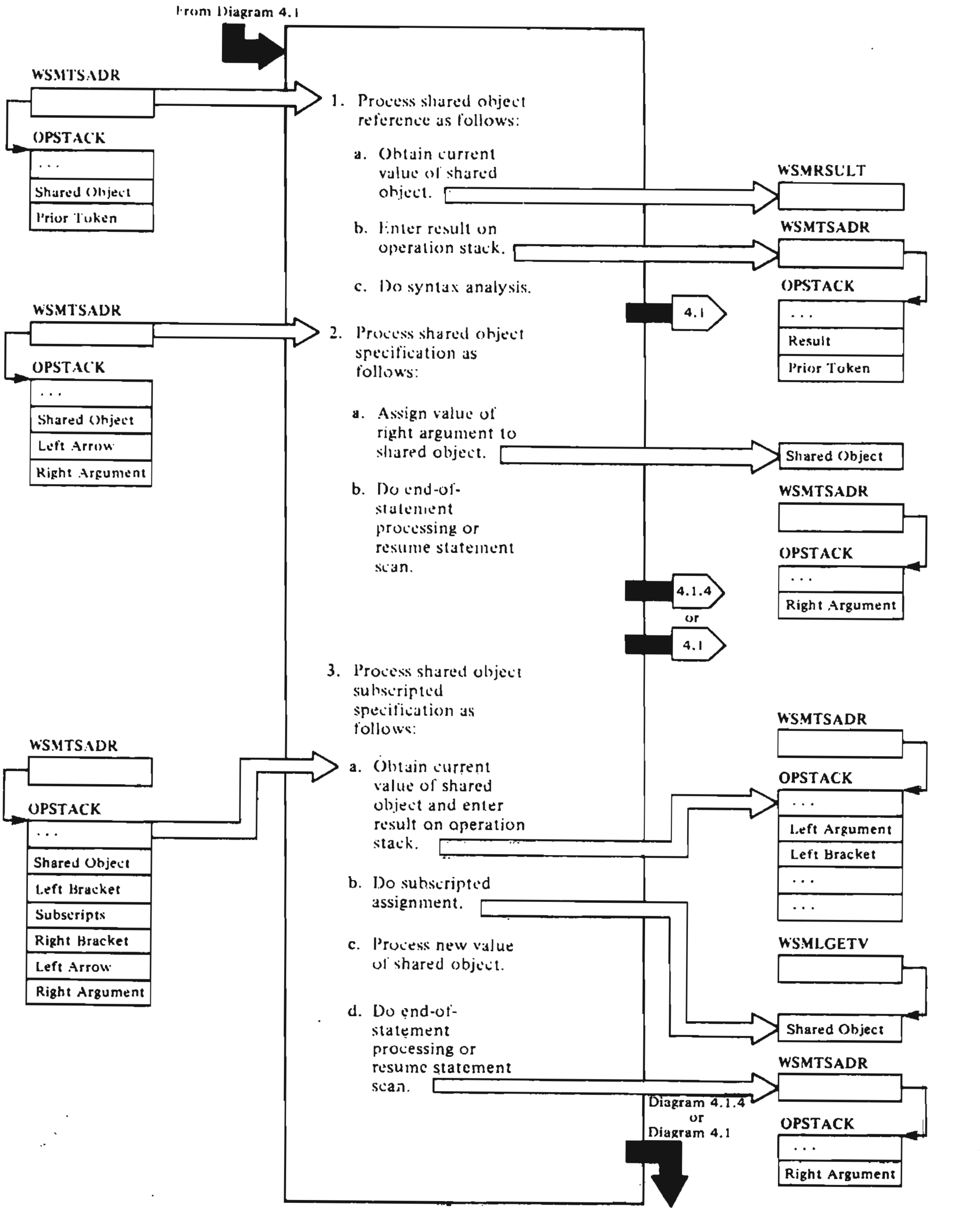
      Stop is requested; in EOS token, EOSTPBIT=1. [IESCANG, ENTRY8 or ACTION10]

      There is something to be printed; on the operation stack, the token preceding the EOS is a variable that is not the result of the assignment. [IESCANG, ACTION10]

      The user has signalled attention; in WSMASYNC, SWATTN or SWDATTN=1. [IESCANG, ACT10A]

   b. For other cases, the WSMTSADR field is set so that the null token that precedes the EOS token or the result of assignment is the top token on the operation stack. Control is passed to IESCANG, ACTION0 to resume statement scan. [IESCANG, ACT10C]

DIAGRAM 4.1.5: SHARED OBJECT PROCESSING



Notes for Diagram 4.1.5

APLIESCA

1. Processing for the shared object reference is as follows:

   a. The token for the shared object is obtained from the operation stack, and entered in the left argument block (WSMLGETV). [IESCANG, ACTION11]

   APLIATRN

   The type of shared object is determined by the IAQUADS routine, and processed as follows:

   For quad: The ITINPUT routine is called to obtain input from the terminal. The ITEMPFUN routine is then called to build a temporary niladic function in free space; the body of the function is the tokenized terminal input. The syntax and internal name of the function are then entered in the WSMRSULT field. [IAQUADS, CALLIN]

   For quote-quad: the ITINPUT routine is called to obtain terminal input. If the input is null, the syntax and internal name of the null character vector (WSMNULCH) are entered in the WSMRSULT field. If the input is scalar, it is entered in the WSMRSULT field as a stack immediate value. For other input, a temporary internal name and a block of free space are obtained, the input is entered in the block as a character vector, and its syntax and internal name are entered in the WSMRSULT field. [IAQUADS,QUADP]

   APLIASYV

   For system variable: The value of the variable is either computed (quad-WA and quad-LC), obtained from the executor (quad-AI and quad-TS), or obtained from the variable's address table entry or value block (all other system variables). If the value is logical or a small integer scalar, it is entered in the WSMRSULT field as a stack immediate value. For other cases, a temporary internal name and block of free space are obtained. The current value of the system variable is entered in the block, and its syntax and internal name are entered in the WSMRSULT field. [IASYSREF]

   APLIASHV

   For shared variable: The YYSREF service request is issued to transmit the current value of the shared variable from shared memory to the unallocated block. A temporary internal name is obtained and given to the new value block. The IACHK routine is called to validate the data. The internal name of the new value block is entered in the variable's share-ID block. The old value and its internal name are freed. The syntax and internal name of the new value block are entered in the WSMRSULT field. [IASCOPY]

   APLIESCA

   b. The syntax and internal name or immediate value of the result is obtained from the WSMRSULT field and entered in the operation stack in place of the shared object entry. [IESCANG, ACT11C]

   c. Control is passed to IESCANG, DECIDE2 for syntax analysis using the result and prior token.

APLIESCA

2. Processing for shared object specification is as follows:

   a. The entries for the shared object and right argument are obtained from the operation stack and entered in the left and right argument blocks (WSMLGETV and WSMRGETV). [IESCANG, ACTION15]

   The type of shared object is determined by the IAQDSPEC routine, and processed as follows:

   APLIATRN

   Quad or quote-quad: the IAGOUT routine is called to transmit the value of the right argument to the terminal. [IAQDSPEC, CALLGOUT]

APLIASYV

System variable: For a read-only system variable, the specification is ignored. For other cases, the right argument value is entered in the system variable's address table entry or value block. If the value is invalid, the implicit error bit (ATIMERR) is set to 1 in the system variable's address table entry. If the system variable is quad-PW or quad-HT, the new value (if it is valid) is transmitted to the executor. [IASYSPEC]

APLIASHV

Shared variable: A temporary internal name and block of free space are obtained and the right argument is copied. The internal name is entered in the WSMRSULT field. The YYSSPEC service request is issued to transmit the new value to shared memory. The internal name of the new value block is entered in the shared variable's share-ID block. The old value block and its internal name are freed. [IASHSPEC]

APLIESCA

b. If the next input token is EOS, control is passed to the end-of-statement processing routine (see Diagram 4.1.4). For any other case, the WSMTSADR field is set so that the right argument is the top token on the operation stack. Control is passed to IESCANG, ACTION0 to resume statement scan. [IESCANG. ACT7X]

3. Processing for shared object subscripted specification is as follows:

   a. The token for the shared object is obtained from the operation stack and entered in the left argument block (WSMLGETV). In the right bracket token, the SHRASGN bit is set to 1 to indicate subscripted assignment to a system or shared variable. [IESCANG, ACTION11]

   APLIATRN

   The type of shared object is determined by the IAQUADSA routine, and the current value is obtained as follows:

   APLIASYV

   System variable: For system variables with an immediate value, a RANK error exit is taken. For read-only system variables, a copy of the variable's current value is made; the syntax and internal name of the copy is placed in the WSMRSULT field. For other cases, the syntax and internal name of the variable's value block are entered in the WSMRSULT field. [IASYSREF]

   APLIASHV

   Shared variable: The current value of the shared variable is obtained as described in 1a above. [IASCOPY]

   APLIESCA

   The result is processed as described in step 1b above. Then control is passed to IESCANG, DECIDE2 for syntax analysis. [IESCANG, ACT11C]

   b. Since the current token on the operation stack is now an ordinary variable, and the prior token is a left bracket, the subscripting routine (IEINDD) is called. (The subscripted assignment is done as described in Diagram 4.1.4, step 1). At completion, WSMLGETV contains the internal name of the shared object's new value; that is, its current value as modified by subscripting.

   c. The type of shared object is determined by the IASHRPST routine, and the new value is processed as follows:

   APLIASYV

   System variable: For read-only system variables, the new value and its internal name are freed. For quad-HT, the new value (if it is valid) is transmitted to the executor. For other cases, no processing of the new value is needed. [IASYSPST]

   APLIASHV

   Shared variable: The YYSPEC service request is issued to transmit the new value to shared memory. [IASHSPEC]

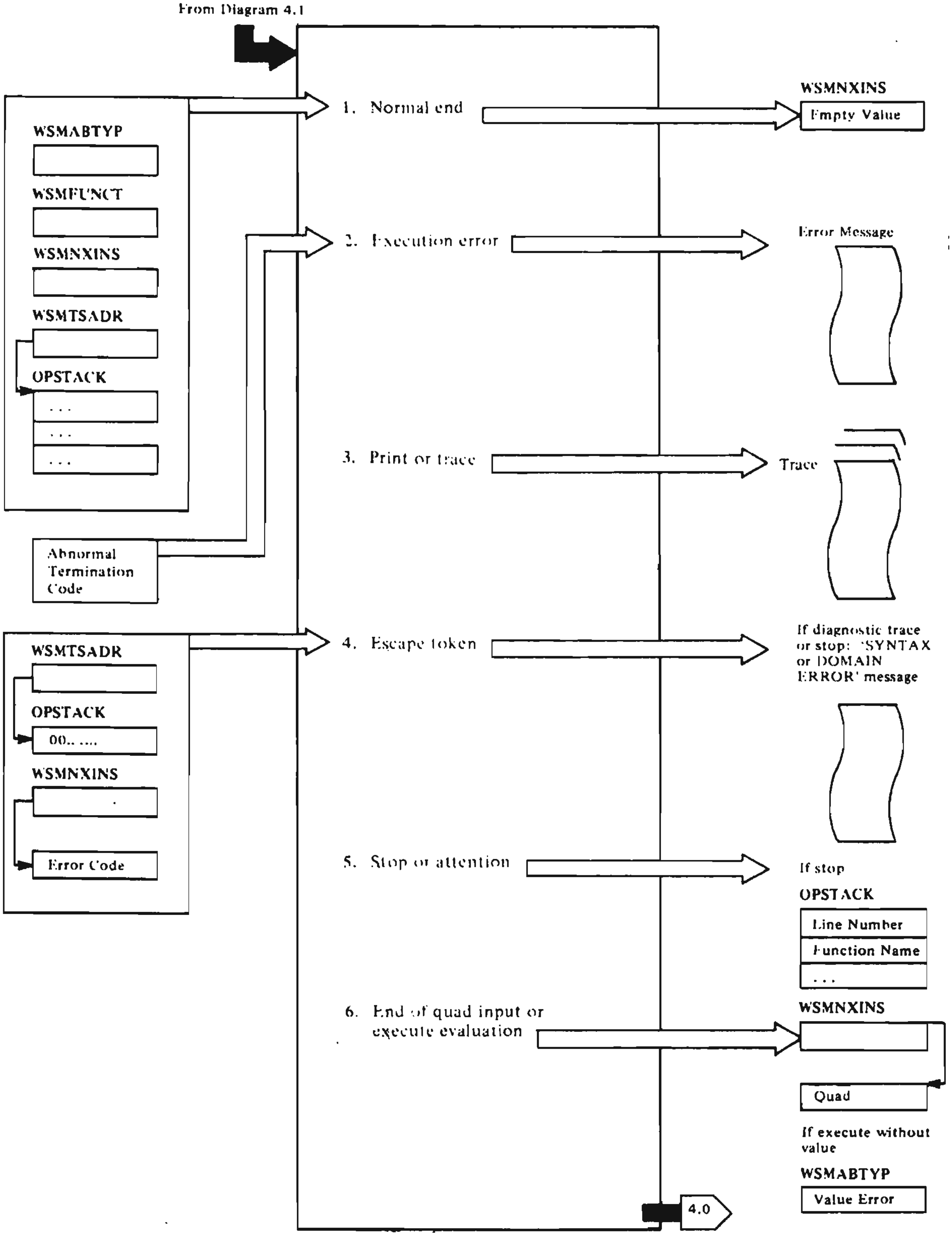
APLIESCA

The WSMTSADR field is set so that the right bracket is the top token on the operation stack. [IESCANG, ENTRY10B]

d. If the next input token is EOS, control is passed to the end of the statement processing routine (see Diagram 4.1.4).

For other cases, the WSMTSADR field is set so that the right argument is the top token on the operation stack. Control is passed to IESCANG, ACTION0 to resume statement scan. [IESCANG, ACT7X]

DIAGRAM 4.2: RETURN CODE PROCESSING



Notes for Diagram 4.2

APLITEX

1. At normal end, the null token is deleted, and the temporary function is freed. [ITEXECUT, EXNORM]

   If the statement that was executed was a branch to line zero and the operation stack was empty, the temporary function is erased before control returns to this routine. On exit from this module, the WSMNXINS and WSMFUNCT fields are set to empty values.

APLITERR

2. Execution error is signalled by an abnormal termination code greater than the value of ABESCA. Error processing occurs in a loop. Each time that the calling function is made the active function, processing returns to step a below. Termination conditions are: An unlocked function is found and suspended; an end-of-stack condition is met; a quad-temporary is found and execution is restarted. Ordinarily, control passes to the ITERRORS routine, where processing is as follows:

   a. The operation stack contains entries representing the state of the expression being evaluated when the error occurred. These include: primitive functions, temporary results, separators, subscript lists, etc. Each operation stack entry is deleted and remote temporary variables are freed, until a suspended function or end-of-stack condition, or a function call block (FCB) is met. The FCB is for the function named in the WSMFUNCT field; suspended functions and end-of-stack conditions indicate that the function named in the WSMFUNCT field is an immediate execution function. [ITERRORS, ERLOOP1]

   b. Subsequent processing depends on the type of the active function: If the active function is locked, the error indicated as RANK, VALUE, etc. in the WSMABTYP field is changed to a DOMAIN ERROR. [ERLOCK]

      The caller of this function is made active; the FCB for the function is deleted from the operation stack by the IESUNFUN routine.

      Note: The calling function may be damaged (that is, erased, or modified by an edit command). Damage to a function is discovered during type determination. The function is then treated as locked and the error is changed to SI DAMAGE.

   c. If the function is an execute temporary function, the temporary function is freed, and the trouble report (prefixed by the execute symbol) is displayed. [EREXEC]

      The IESUNFUN routine is called to delete the function call block of this function and the caller of the deleted function is made the active function.

   d. If the function is a quad temporary function, the error message is displayed, the temporary function is freed, and its caller is made active (see step b above). The WSMNXINS field is decremented by two so that it addresses the quad token for reexecution and the interpreter is recalled.

   e. If the active function is defined and not locked, the error message is displayed and the function is suspended. [ERNOLOCK]

   f. If the function is an immediate execution function, the error message is displayed and the temporary function is freed. [ERIMEX]

   g. If the caller of the ITERRORS routine is a system command processor (see Diagram 1.1) requesting an interrupt after a save command during quad-input, control is returned to the system command processer. [EXIT]

   h. If the caller of the ITERRORS routine is the ITEXECUT routine, and the error is not such as to reinvoke a quad function, registers 13 and 14 are set to the bottom of the R13 stack. [EXIT] Control is passed to the input routine. [ITINPUT]

If the error occurred in quad-input, control is returned to the caller (ITEXECUT) to reinvoke the interpreter. [EXIT]

APLITERR, APLITSUB

3. If the line is to be traced, the ITPRFNLN routine is called to enter the function name and line number in the buffer. If the line is a branch statement, a right-arrow graphic is entered in the buffer. [ITPRFNLN]

   APLITEX

   For end-of-line printing, or for tracing, the value on top of the operation stack is passed to IAGOUT to be formatted and placed in the buffer (to follow the trace output, if any).

   If an attention signal is received, part of the display may be built but printing does not occur.

4. The escape token signals either an ill-formed line or the assignment to a trace or stop vector. The escape token is in the right half of the word at the top of the operation stack. If its high-order byte contains zero, this signals the head of an ill-formed line. The WSMNXINS field contains the address of the error code [ABSYNT or ABDOMA], followed by the text of the line. The ITERRORS routine is called to display a SYNTAX or DOMAIN ERROR message. [ITEXECUT, ESCAPE]

   If the escape token signals a diagnostic trace or stop vector, processing is done by the TSTEST routine as follows:

   a. The ITFETCH routine is called to validate the value given and to procure its elements. [TSBOTH]

   b. For each integer value, a trace or stop bit is set in the named function.

APLITEX

5. For attention signal only, processing is as follows:

   The buffer is cleared. If the current function is locked, execution continues. (A locked function is never suspended.) [ATTN]

   If the function is an immediate-execution function, the function is freed and the ITEXECUT routine returns to its caller.

   If the current function is a temporary function built from a quad-input statement or the execute primitive, processing occurs as in step 6.

   For the defined, unlocked function, processing occurs as for stop, described below. [ATTPERM]

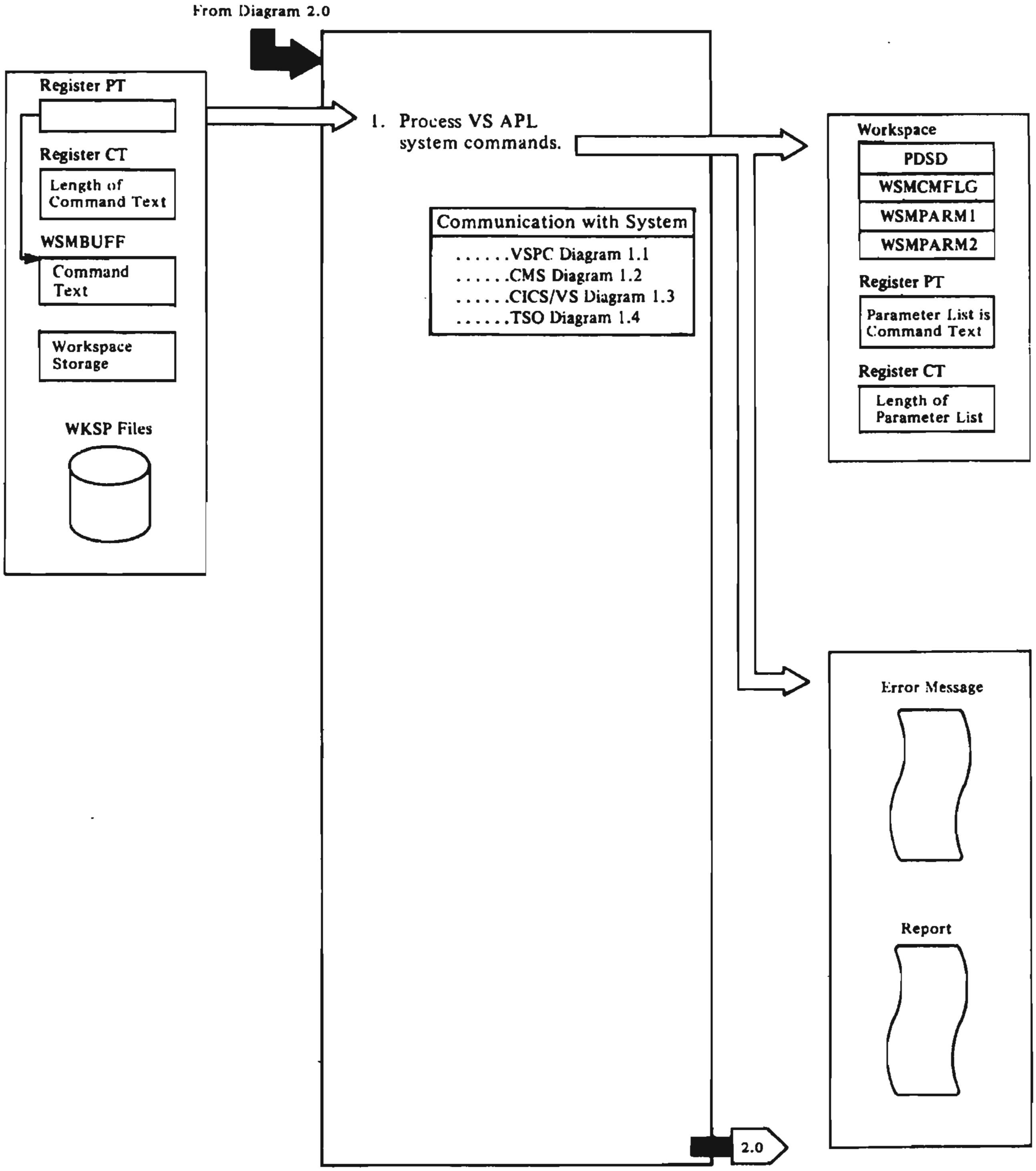
   For stop or attention, the function is suspended. That is, the number of the next line to be executed, the name of the function, and a bit to indicate that the function is not damaged, are placed on the operation stack (in place of the initial null and adjacent to the FCB for this function). Control is then returned to the caller. [STOPP]

6. The temporary function created from quad-input or the argument of execute is deleted from the operation stack. The value resulting from its evaluation is placed on top of the operation stack and the calling function is made the active one. [UNQUEX]

   If there was no value, the position in the calling function is checked: If at end of line, execution continues; if not, ITERRORS is called to cause a VALUE error. In any case, if the calling function is at end-of-statement, a test is made for the presence of conditions 1 through 5.

   Note: Any combination of conditions 1 through 6 can occur together, or recursively. When all conditions have been cleared, one of three cases obtains: Execution is over; control is returned to ITINPUT to prompt the user; an error exists, ITERRORS is called to handle it; execution continues, and IEXARCH is called to resume execution.

DIAGRAM 5.0: SYSTEM COMMAND EXECUTION



Notes for Diagram 5.0

APLITCMD

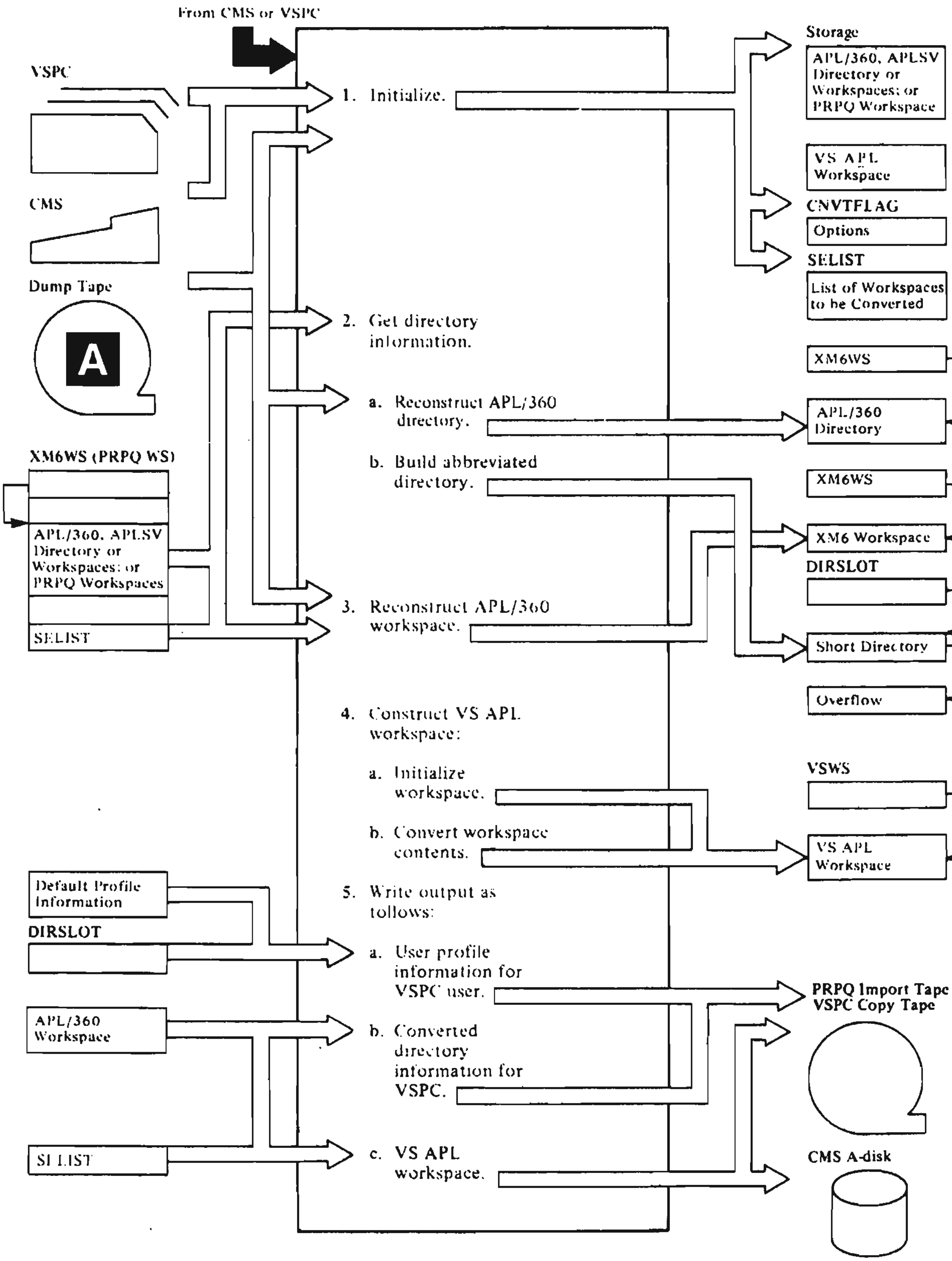
1. The type of command is determined and the corresponding verb is located in the VERBTABL table. The command syntax is analyzed and execution parameters are built. Control is passed to a routine (see below) to execute the command. [ITSYSCMD]

   The commands and the routines that execute them are listed below. For commands that affect the system outside the active workspace, service request calls to the executor are issued by the routines.

Command	Routine
CLEAR	ITCMCLEA
CONTINUE	ITCMCONT
COPY	ITCMCOPY
DROP	ITCMDROP
ERASE	ITCMERAS
FNS	ITCMFNS
GROUP	ITCMGROU
GRP	ITCMGRP
GRPS	ITCMGRPS
LIB	ITCMLIB
LOAD	ITCMLOAD
MSG	ITCMMSG
OFF	ITCMOFF
OPR	ITCMOPR
PCOPY	ITCMPCOP
QUOTA	ITCMQUOT
SAVE	ITCMSAVE
SI	ITCMSI
SINL	ITCMSINL
STACK	ITCMSTAC
SYMBOLS	ITCMSYMB
VARS	ITCMVARS
WSID	ITCMWSID
WSSIZE	ITCMWSSI

   Any other syntactically-valid command will be passed to ITCMCMD.

DIAGRAM 6.0: WORKSPACE CONVERSION



Notes for Diagram 6.0

APLCINIT (CMS), APLOINIT (VSPC)

The three versions of the VS APL conversion program described are:

- CMS under CMS, conversion from APL/360 or APLSV to VS APL.
- VSPC under OS/VS or DOS/VS, conversion from APL/360 or APLSV to VS APL.
- PRPQ under CMS, conversion from APL/CMS (PRPQ) to VS APL.

1. CNVTFLAG consists of bits that are set to indicate options specified in the convert command (CMS) or specified by convert command cards (VSPC). The SELIST is built from select parameters (APLCPARM, APLOPARM). Then storage is obtained (by GETMAINs) for:

    a. The APL/360 or APLSV workspace or directory

    b. The VS APL workspace

    c. The display buffer

    The tape label and first data record on the tape are read to compute buffer size. The tape is then repositioned to the first data record. Also, the printer data set is opened. The XM6WS pointer is set to point to the start of the APL/360 workspace. The VSWS pointer is set to point to the start of the VS APL workspace. The BUFFSTRT pointer is set to point to the display buffer.

2. The input tape is now read, workspace by workspace. However, there are two types of workspaces that are very similar in structure: directories and workspaces proper. If there are directories on the tape, they all precede the workspaces proper. There may be any number from 0 to n of directories. Therefore, the directory (if any) is read and reconstructed from its condensed tape form into the APL/360 slots (APLCINIT and APLOINIT).

    If APLCINIT or APLOINIT identifies this workspace as a directory, it calls APLCDIRE or APLODIRE to process it. In CMS, APLCDIRE is a dummy routine which prints the message "DIRECTORY" at the terminal. In VSPC, APLODIRE extracts data from each PERLIB of interest, and saves the extracts in DIRSLOT. It is saved until the workspace and account to which it pertains is finally found, later on the tape. If full conversion, extracts from all PERLIBs are saved. If select conversion, only those PERLIBs pertaining to workspace and accounts in SELIST are saved. If resume conversion, only those of PERLIBs pertaining to the workspace at which conversion is to resume, and all following workspaces are saved.

    DIRSLOT holds extracts for up to 400 accounts (there is one PERLIB per account). If there are more accounts, DIRSLOT overflows; it is written as a block to a temporary data set (APLDIRE) to make the slot available for 400 more accounts. The first word in DIRSLOT is a high water mark pointer which points to the next available position for an extracted PERLIB. The data extracted is:

    APL/360 library (account number)

    PASSWORD

    WORKSPACE QUOTA

    SHARED VARIABLE QUOTA (if any)

    MAX TIME BETWEEN INTERACTIONS

    If the account is empty (no workspace for this library), the PERLIB is ignored. Later, when a workspace proper is converted, these saved extracts will be used to create the VSPC user profile record and directory entry record.

3. Eventually, APLCINIT or APLOINIT reconstructs the first of the workspaces. When this happens, there are no more directories because a directory cannot follow a workspace on a VS APL dump tape. Upon identifying the workspace as a workspace, APLCINIT or APLOINIT calls APLCCULL (CMS) or APLOCULL (VSPC) to determine if the workspace should be converted. APLCCULL or APLOCULL checks (if select conversion) if the workspace is in SELIST. If not and if select conversion, the workspace is ignored and APLCINIT or APLOINIT gets the next workspace. APLCCULL and APLOCULL also validate the library number and workspace name. If VSPC, and either is invalid and not renamed in SELIST, the workspace is rejected. If CMS and either is invalid, APLCCULL or APLOCULL requests a new number and/or a new workspace name from the

terminal. If resume conversion, workspaces are ignored until the one specified in the resume command is encountered. Thereafter, conversion reverts to full conversion logic. If the workspace passes culling, control is returned to APLCINIT or APLOINIT, which calls APLCWKSP (CMS) or APLOWKSP (VSPC) to manage workspace conversion.

4. Construct VS APL workspace:

   a. The VS APL slot is initialized. This is a clear workspace (APLCLEAR, APLOLEAR, or APLQLEAR) with the workspace environment converted by CLEAR. APLCWKSP, APLOWKSP, or APLQWKSP then calls APLCIBNM, APLOIBNM, or APLQIBNM to provide a unique name for the IBEAM simulator function which may have to be added to the workspace as a result of idiom conversions. Then APLCWKSP, APLOWKSP, or APLQWKSP unshadows global names so that each active symbol table or address table entry points to its most global value (if any).

   b. At this point, conversion of workspace objects begins. For the rest of this workspace, APLCWKSP, APLOWKSP, or APLQWKSP is driven by the symbol table or address table through which it loops looking for variables, groups, and functions which have values. APLCVARB (CMS), APLOVARB (VSPC), or APLQVARB (PRPQ) is called to validate and convert variables. APLCGRUP (CMS), APLOGRUP (VSPC), or APLQGRUP (PRPQ) is called to convert groups. The converted objects (variables, groups) are entered into the VS APL workspace symbol table by APLITIDS. Space for the objects in the sink workspace free space is obtained by calling APLIESPA. These are VS APL interpreter routines borrowed by conversion and require VS APL linkage (APLCALL, APLEXIT macros).

      Upon encountering a function in the symbol table, WKSP calls APLCFUNC (CMS), APLOFUNC (VSPC), or APLQFUNC (PRPQ) to manage the conversion of the function. It is here that idiom (context) conversion takes place.

      Functions are converted line by line from internal tokens to display format by APLCDISP, APLODISP, or APLQDISP. First, FUNC calls DISP to display the header line. Syntax errors are not tolerated here; if any are found, the function is ignored. If no errors are found in the header, FUNC calls VS APL interpreter routine APLITHDR to tokenize the function header into the VS APL workspace. Also, APLITHDR enters the name of the function into the VS APL symbol table along with any declared locals, results, and arguments. Then FUNC calls DISP to display and make idiom conversions for each line. DISP returns with a summary of idioms found which FUNC places in the summary table with the function line number to which it pertains. FUNC enters each displayed (converted) line into VS APL by calling APLITLXS. Finally, all function lines are processed; FUNC formalizes the converted function by calling APLITFDC. Then FUNC analyzes the summary table, calling APLCRPRT (CMS), APLORPRT (VSPC), or APLQRPRT (PRPQ) to print a summary of idioms found and the lines in which the idioms occurred. There is no printing if no idioms occurred. The summary table is reset for the next function, and control is returned to WKSP for the next object.

      FUNC does not go through this process, however, for identifiable workspace functions: ORIGIN, SETLINK, SETFUZZ, WIDTH, DELAY, and DIGITS (from distributed library 1 in XM6). If FUNC detects a locked, two-line function, it calls APLCWSFN (CMS) or APLOWSFN (VSPC) only. This routine checks the function bit by bit for a match with one of the WSFNS functions listed above. If it does not match, control is returned to FUNC, which processes the function in the normal way. If it does match, WSFN returns to FUNC with a "hit" return code and a pointer to a function that is the VS APL equivalent. FUNC then calls APLCSHIP (CMS and PRPQ) or APLOSHIP (VSPC) to process the substitute. SHIP enters the substitute by calling in turn APLITHDR, APLITLXS, and APLITFDC. FUNC then prints the message "REPLACED" on the conversion report via RPRT. Eventually, WKSP exhausts the XM6 symbol table. At this point, WKSP adds the IBEAM simulator function to the VS APL workspace if appropriate. It does this by calling APLCSHIP or APLOSHIP with a pointer to the VS APL definition of the simulator function. APLCSHIP, or APLOSHIP enters the simulator in the same way it entered the workspace functions. Conversion of the workspace is now completed. WKSP then calls APLCSAVE (CMS), APLOSAVE (VSPC), or APLQSAVE (PRPQ) to write out the converted workspace.

5. In CMS, APLCSAVE writes the VS APL workspace to the user's A-disk and calls RPRT to print the CMS file identification of workspace. In VSPC, saving is more complex. If the workspace is the first encountered in an account (THIS LIBNO ≠ LASTLIBNO), SAVE creates a user profile record which it writes to tape (APLOUT). To do this, it retrieves the extracted PERLIB from DIRSLOT by calling GETDIRE in APLODIRE. If there were no directories, SAVE uses default values to create the user profile. This logic occurs only for the first workspace encountered in each library. For all workspaces, SAVE creates and writes to tape a VSPC directory entry record describing the workspace. Finally, SAVE writes the workspace on APLOUT as 16K byte control intervals as if the workspace were a member of a VSAM data set. Control then returns to WKSP, which returns control to INIT to get the next workspace. PRPQ APLQSAVE writes the workspace either to VSPC input tape or to the user's CMS A-disk.

A. CMS and VSPC

Tape structure. An example of tape structure is shown below:

a. Tape label, one or two 80-byte records.

First record is optional VOL1 record

Second (or first if no VOL1) is HDR1 record; contains record size in bytes 57, 58.

	APL LIBRARY DUMP APL SVS LIBRARIES	RECORD SIZE
1....4	5...............21	57 58

b. Data: directories and workspaces, variable length records. Each directory or workspace:

1st record	144 bytes from workspace (or directory) origin through SV1.
n records	of variable length from PARREL through m-entries to beginning of free space. Last record may be padded with a few bytes of free space if too short for a tape record.
n records	of variable length from top of execution stack (low core) through bottom (hi core) of R13 stack.

Sequence is: directory 0 through directory n followed by workspaces in directory and PERLIB order (entry sequence, not collating sequence). In APL/360 tapes, workspaces are in PERSAVEW order; that is, entry sequenced.

c. Trailer label

EOF1	if end of file
EOV1	if end of volume
m	columns 1 through 4

APLQINIT (PRPQ)

1. APLFLAGS consists of bits describing conversion options. These bits are set by APLQPARM from execution parameters and terminal input.

   APLQINIT establishes the first values for most other modules. APLQINIT takes all of virtual storage with the CMS macro

DMSFREE. Conversion cancels if there is not at least 64K bytes available. APLQINIT then returns to CMS, 16K bytes at the low end and 16K bytes at the high end of the area taken. This is to provide CMS with free space for implicit GETMAINs and DMSFREEs. The remaining storage is then allocated for the VS APL workspace and the APL/CMS (PRPQ) workspace. The display buffer comprises the PRPQ R13 stack and the VS APL WSMBUFF.

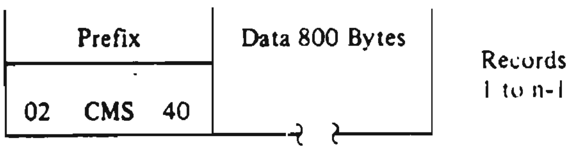
2. No directory for PRPQ.

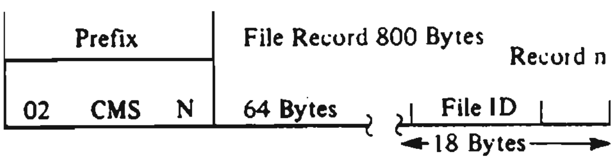
3. APLQINIT builds APL/CMS (PRPQ) workspaces in the PRPQ slot from a CMS dump tape input. On the tape, the workspaces are compacted, thus they have to be properly constructed in storage. Also, internal workspace pointers are relocated. If the option is resume, APLQINIT checks the fileid for a match with the resume point fileid and bypasses further processing of this workspace if there is no match. When the match is found, APLQINIT processes that workspace and all subsequent workspaces on the input tape.

4. Same as numbers 4 and 5 for CMS and VSPC.

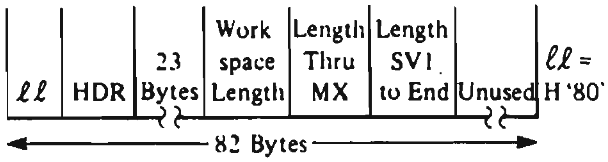
A

CMS dump tape structure for APL/CMS (PRPQ) workspaces 805-byte physical records as shown below.



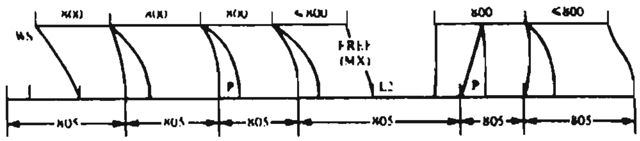


However, the data portion represents logical disk records as shown below.



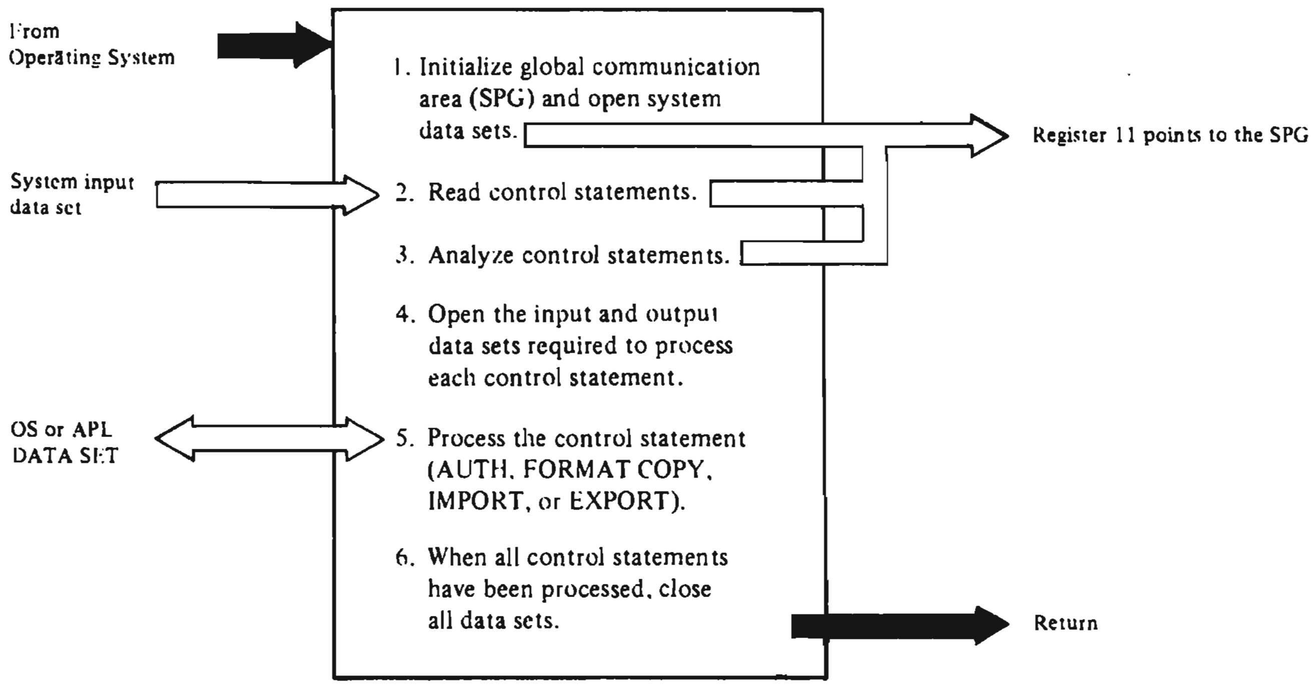
1. One or more logical records containing workspace from origin to beginning of free space.

2. One or more logical records containing end of free space to end of R13 stack as shown in the example that follows.



P = CMS Prefix
L1 = Halfword length of logical record in bytes
L2 = Halfword length of logical record in bytes

DIAGRAM 7.0: CICS/VS LIBRARY SERVICE PROGRAM



Notes for Diagram 7.0

The APL library service program runs as a batch job, separate from the VS APL online subsystem. The library management commands are control statements for the service program. These commands are COPY, EXPORT, IMPORT, FORMAT, AUTH, and ENVIRONMENT. The commands are contained in the SYSIN data set.

The service program executor module, APLKVEXC, is the first-level module. It controls the execution of the second-level subroutines: APLKVINT and APLKVTRM, which initialize and terminate each service program request, and APLKVCMD, which analyzes each control statement request. Another set of second-level routines actually process the control statements. Input and output are done by a set of third-level modules called by the second-level routines.

The message processor module, APLKVMSG, writes output to SYSPRINT in response to calls from all three module levels. Communication among the service program modules is made using a global work area, the SPG. It is addressed using register 11.

The APL data sets used by the service program are either the APL directory data set, a key-sequenced VSAM data set, or the APL library (entry-sequenced VSAM data set that contains the library data).

APLKVEXC

1. Initializes the SPG and calls ALKVPIN to read the JCL input parameters and open the required data sets.

2. Reads and scans the next control statement and moves it to the buffer in the SPG. Calls APLKVMSG to print the control statement (passwords are converted to blanks). If the control statement is continued, the remaining data is read, a card image at a time, and printed. Continuation marks are removed, and a complete statement is prepared in the buffer. [READCOMM]

APLKVCMD

3. This module contains the syntax tables defining the valid control statements. When called by APLKVEXC, it calls APLKVSCN. APLKVSCN processes the control statement against the tables in APLKVCMD and returns the encoded control statement in SPGPARMA. A code representing the control statement type is placed in SPGCOMM.

APLKVINT

4. APLKVEXC passes input to this routine in the SPGPARMA and SPGOPENA fields. This routine checks for invalid data set names in a TO or FROM operand.

   This routine then completes DCBs with default values for parameters not specified by the user's JCL, and an end-of-data exit address. Initialization procedures, by control statement, follow:

   - AUTH - none
   - COPY - Open the data sets named TO and FROM operands. If TO and FROM aren't both named, open APLLIB and APLDIR. If the COPY statement is to the APL library, open the APL library for output.
   - FORMAT - Ensure that the APL directory and library data sets are open.
   - EXPORT - Open the output data set and ensure that the APL library is present.
   - IMPORT - Open the input data set, and open the APL library for output.

5. APLKVEXC calls the second-level modules that follow to process the control statements. Note that IMPORT accesses an OS data set as input; EXPORT produces one as output. COPY can accept COPY-produced sequential data sets in lieu of an APL input library; COPY can produce an OS output data set.

   APLKVAUT - AUTH Control Statement

   If user level authorization is requested, reads the user profile from the APL library. Compares the password passed with the AUTH control statement with the user log-on password. The user's identification from the AUTH control statement is saved in the SPGUSID field. If system level authorization is requested, checks the password against that in APLKPASS (APL directory update password). The privilege level of APL library access is saved in SPG-PRIVA.

APLKVFMT - FORMAT Control Statement

Requires complete library level authority over the APL library and an unformatted library. Formats the APL library data set into 4K blocks. Builds a free space profile and writes it to the APL directory. If USERS is requested on the FORMAT control statement, writes the user profiles for libraries 1, 2, and 314159.

APLKVCPY - COPY Control Statement

Requires a system level authority over libraries being accessed when a range of libraries are to be copied. Searches the input library or FROM data set over a range of one or more user identifications, calling APLKVLBI for I/O. For each user profile read, either ignores (for the REMOVE option) or writes profile to output library or TO data set calling APLKVLBO or APLKVTPO for I/O. If copying to the APL library, calls APLKVALD to allocate space for the files. For each user written library, inspects directory records for all files owned and writes files matching the TYPE attribute.

APKLVEXP - EXPORT Control Statement

Calls APLKVLBI to read directory entry from input data set. Calls APLKVLBI to read each control interval of the member from the APL library and calls APLKVDSO to deblock and write the contents of the control intervals to the operating system data set.

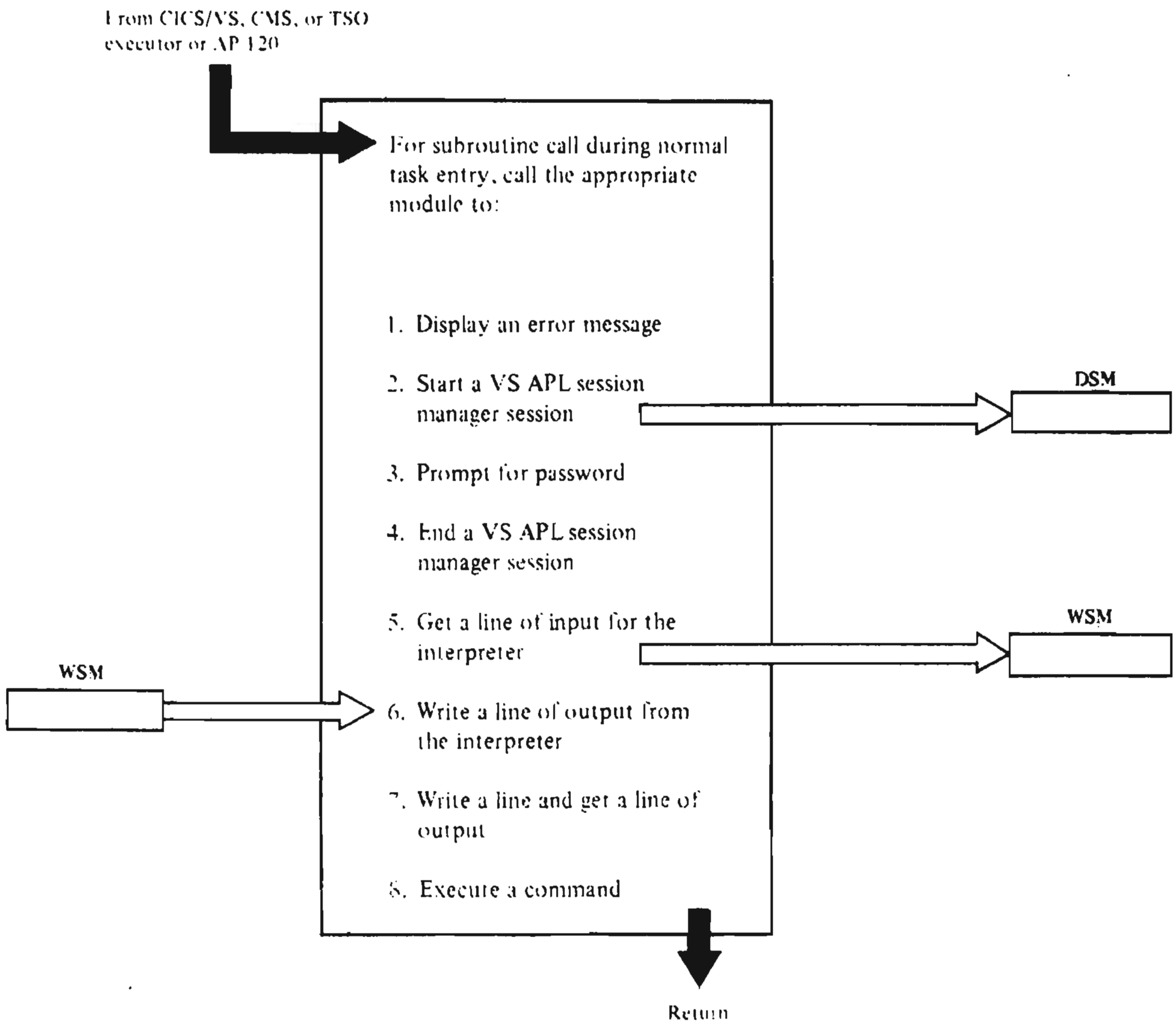
APLKVIMP - IMPORT Control Statement

Checks input parameters for consistency. Calls APLKVLBI to read the library profile of the library being imported to. Calls APLKVALD to allocate space for the file. Creates a new directory entry and calls APLKVLBO to write the entry to the output library. Calls APLKVDSI to read the input data set being imported and block its records into a control interval. Calls APLKVLBO to write each control interval to the APL library.

APLKVTRM

6. Checks the SPGOPEND field to determine whether there are open OS data sets. If so, issues a CLOSE macro instruction to close the data sets whose DCBs are identified in the SPGRDDCB and SPGWQDCB.

DIAGRAM 8.2: VS APL SESSION MANAGER EXECUTOR SCHEDULER



Notes for Diagram 8.2

APLASCHD

1. The purpose is to request the VS APL session manager to display an error message for an abending executor, an auxiliary processor, or for any other reason. It also waits for the message to be displayed before returning. [APLAERRM]

2. Tells the VS APL session manager to start the session, then waits for the session to start before returning. [APLAINIT]

3. Requests that the VS APL session manager prompt the user for a password. [APLAPASS]

4. Shuts down the session, waits, then returns. [APLATERM]

5. Tells the VS APL session manager to put a line of text in the WSM. [APLATYI]

6. Tells the VS APL session manager to take a line of text from the WSM. [APLATYO]

7. Tells the VS APL session manager to write a line of output from the WSM, get a line of input from the terminal, and put it in the WSM. [APLATYOI]

8. A text string is passed as an argument, and the entry point tells the VS APL manager session manager to process this string as a VS APL session manager command. [APLAXCMD]

DIAGRAM 8.2.1: VS APL SESSION MANAGER EXECUTOR PROCESSOR

From CICS/VS, CMS or TSO initialization

Process VS APL Session Manager request.

1. Take the request off the DSM request chain.

2. Call the appropriate module to:

   A. prompt for input.

   B. queue input lines.

   C. execute a command.

   D. redefine the display screen.

   E. redefine the line column or user specified field.

   F. refresh the display input area.

   G. add a line to the display input area.

   H. display a message.

   I. update the session log.

Return

Notes for Diagram 8.2.1

APLACRCP

1. Takes a single request off the DSM chain and calls APLACPRO to process it.

APLACPRO

2. Determines the type of request and calls one of the modules listed below to process it.

   APLACPRM

   a. Prompts user for input.

   APLACQUE

   b. Queues and dequeues a series of commands or input lines.

   APLACXCM

   c. Verifies the syntax of a command passed by APLACPRO, and, if valid, tries to execute it. If the command is invalid, it returns a message.

   APLACNDP

   d. Defines the position and size of the VS APL session manager display in the user's screen.

   APLACRSA

   e. Defines or redefines the line column and user-specifiable area when the display size or position has changed, or there has been an error message for some session manager error.

   APLACRDA

   f. Redefines condition of display input area after the user has changed the display column or line setting, or the setting of the display command.

   APLACDSL

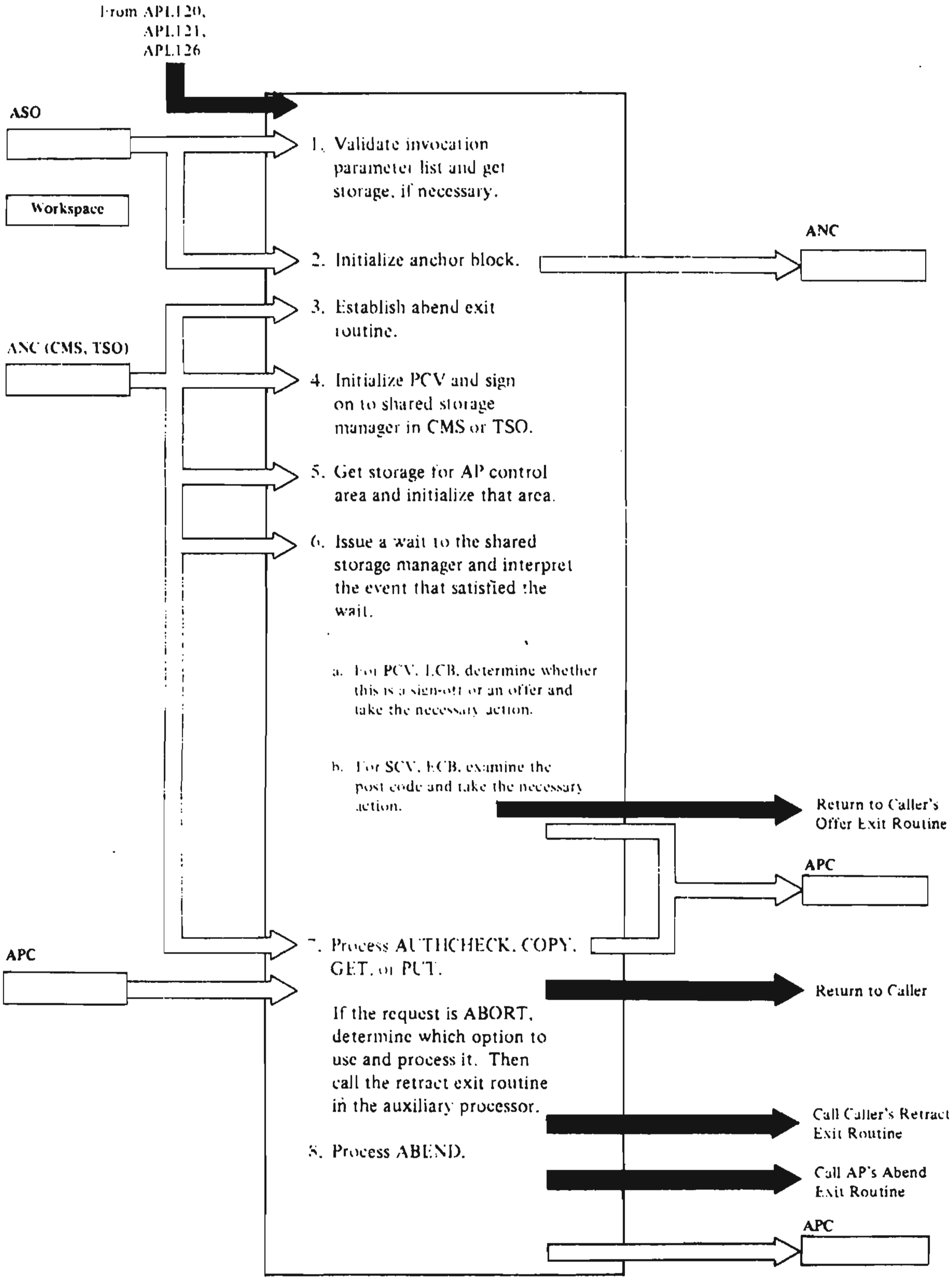
   g. Either is called repeatedly by APLACRDA to define each line of the display area, or is called by APLACPRO to add a single line to the display area.

   APLADMSG

   h. Displays an informational or error message.

   APLACSF

   i. Maintains the session log.


From APL120,
APL121,
APL126
ASO
Workspace
ANC (CMS, TSO)
APC
1. Validate invocation parameter list and get storage, if necessary.
2. Initialize anchor block.
3. Establish abend exit routine.
4. Initialize PCV and sign on to shared storage manager in CMS or TSO.
5. Get storage for AP control area and initialize that area.
6. Issue a wait to the shared storage manager and interpret the event that satisfied the wait.
a. For PCV, ECB, determine whether this is a sign-off or an offer and take the necessary action.
b. For SCV, ECB, examine the post code and take the necessary action.
7. Process AUTHCHECK, COPY, GET, or PUT.
If the request is ABORT, determine which option to use and process it. Then call the retract exit routine in the auxiliary processor.
8. Process ABEND.
ANC
Return to Caller's Offer Exit Routine
APC
Return to Caller
Call Caller's Retract Exit Routine
Call AP's Abend Exit Routine
APC

Notes for Diagram 8.3.1

APLXASD(CMS), APLXAYD(TSO)

1. In CMS, APLXASD scans the invocation parameter list until the end-of-list marker is reached. The DMSFREE macro is invoked to obtain storage. The parameter is copied to the new storage, and a pointer to the storage is returned to APLXAC. The back size of storage obtained is also passed to APLXAC. [APLXAINP]

   APLXAMSG routes messages from APLXAC to the terminal.

   In TSO, APLXAYD picks up the parameter count from the parameter list, invokes macro GETMAIN, and moves these parameters to the new storage. A pointer to the storage and the back size of storage obtained are passed to APLXAC. [APLXAINP]

   APLXAMSG routes messages from APLXAC to the terminal.

APLXAC(CMS/TSO)

2. Every auxiliary processor has an anchor block known as ANC. This contains information passed by the auxiliary processor at sign-on, as well as additional data needed by APLXAC.

APLXAC(CMS/TSO)

3. Calls APLXBEND to establish an abend exit for these modules.

APLXAC(CMS/TSO)

4. PCV is a process control vector. APLXAC sets fields, as requested, in the auxiliary processor sign-on request block for CMS and TSO. It then issues a sign-on to the shared storage manager (SSM). If the sign-on fails. the auxiliary processor is terminated.

APLXAC(CMS/TSO)

5. GETMAIN is invoked for the auxiliary processor control area. This area will contain the ECB list, the SCV ECBs, and the SCVs. The SCV is the shared control vector; there is one SCV per shared variable.

   The addresses of ECBs are now put in the address list. In each SCV is placed the address of the corresponding ECB. In each ECB is placed the index to the SCV list of the corresponding ECB.

APLXAC(CMS/TSO)

6. Passes the address of the ECB list. When the wait is satisfied, one of two events can take place:

   a. PCV ECB has been posted. Determine if sign-off request or offer is received. If sign-off request, free the storage, retract any shared variables (calling the auxiliary processor retract exit for each set of variables), and issue a sign-off to the shared storage manager.

      If offer is received, and it was the primary variable, counter-offer that variable, and initiate an offer for each member of the shared variable set. Get storage for an APC and set the appropriate fields. Transfer control to the auxiliary processor's offer exit routine.

   b. If SCV ECB has been posted, one of two events can take place:

      The user has retracted a variable. If this is a primary variable, retract the set of variables, and call the auxiliary processor retract exit routine. If it is not the primary variable, ignore it.

      Or if the interlock is broken, shared storage is now available, or the user has specified a value, take appropriate action; otherwise, ignore and continue to wait.

APLXAC(CMS/TSO)

7. There are five available service requests:

   AUTHCHECK: Set zero return and reason codes.

   COPY: Issue a COPY request for the variable to the shared storage manager. If there is a temporary reject condition. enter a wait state and try the COPY again if the AP has so requested. If the translate option is set, translate character data.

GET: Issue a reference for the variable to the shared storage manager, and proceed as for COPY.

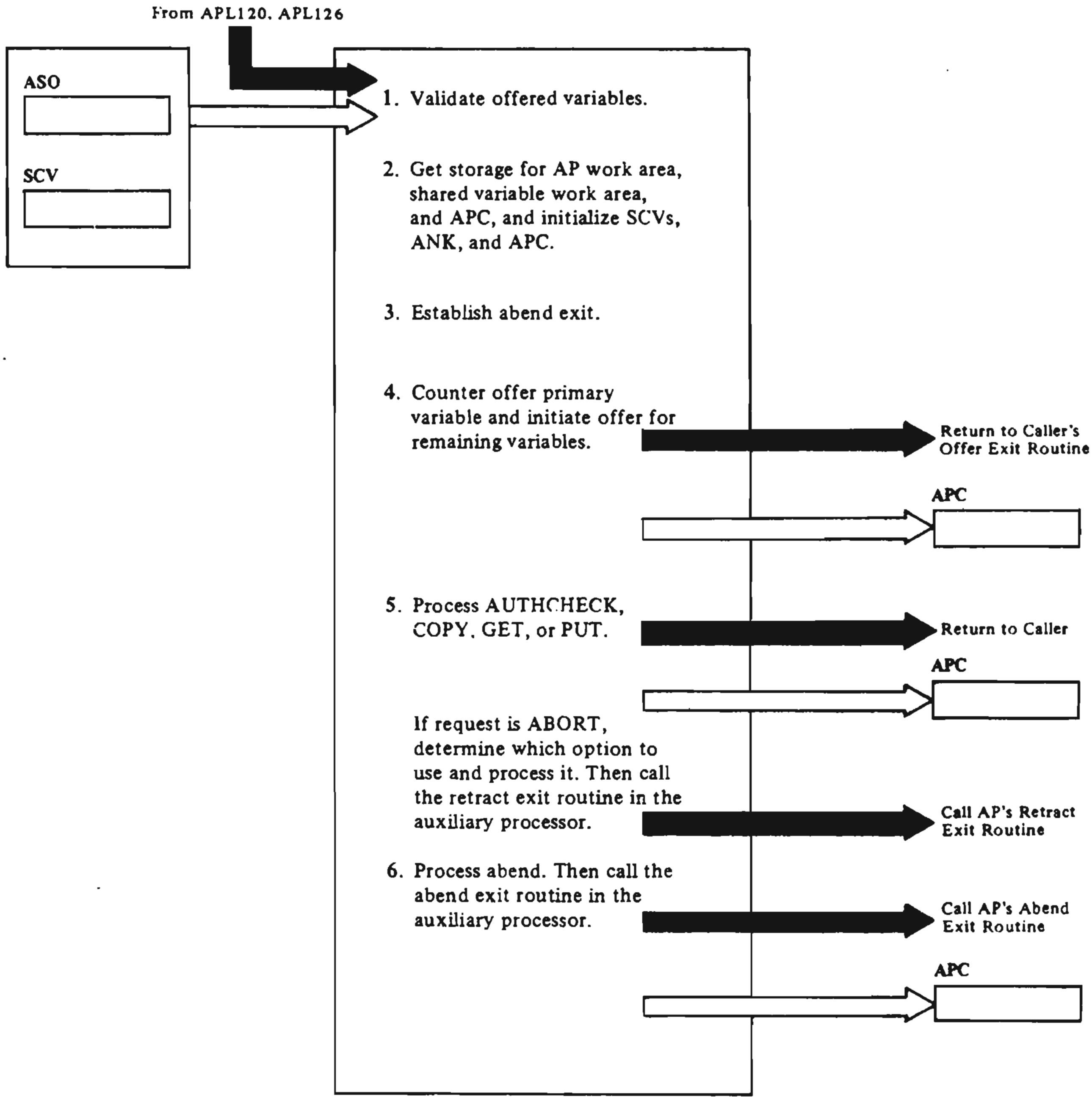
PUT: If the translate option is set, translate character data. Issue a specification to the shared storage manager. If there is a temporary reject condition, enter a wait state and try the reference again if the AP has so requested.

ABORT: There are two options—abort and abort all. For abort, retract the set of variables, pass control to the invoking auxiliary processor's retract exit routine, and free the storage for this set of variables.

For abort all, pass control to the invoking auxiliary processor's retract exit routine, once for each set of variables, to retract all sets; then issue a sign-off to the shared storage manager (CMS or TSO).

If an abend occurs in the auxiliary processor, restore the processor's registers and call the processor's abend exit, if there is one. Then retract the variables in the set and enter a wait state.

DIAGRAM 8.3.2: COMMON AUXILIARY PROCESSOR SERVICES UNDER CICS/VS



Notes for Diagram 8.3.2

APLXAK(CICS/VS)

1. Validate that the shared variable offered is a valid primary variable as defined by the AP in the AP sign-on block (ASO). If not, do not counter-offer, but return.

APLXAK(CICS/VS)

2. Every auxiliary processor has an anchor block known as an ANC. This contains the address of the auxiliary processor sign-on block, as well as additional data needed by APLXAK. Every set of shared variables has an associated APC employed as a communications block between the AP and common auxiliary processor services.

APLXAK(CICS/VS)

3. Calls APLXBEND to establish an abend exit.

APLXAK(CICS/VS)

4. Counter offer, through the shared storage manager, the primary variable, and initiate an offer for each member of the shared variable set.

   Transfer control to the auxiliary processor's offer exit routine.

APLXAK(CICS/VS)

5. There are five available service requests:

   AUTHCHECK: Issue an AUTHCHECK to the shared storage manager.

   COPY: Issue a COPY request for the variable to the shared storage manager. If there is a temporary reject condition, enter a wait state and try the COPY again if the AP has so requested. If the translate option is set, translate character data.

   GET: Issue a reference for the variable to the shared storage manager, and proceed as for COPY.

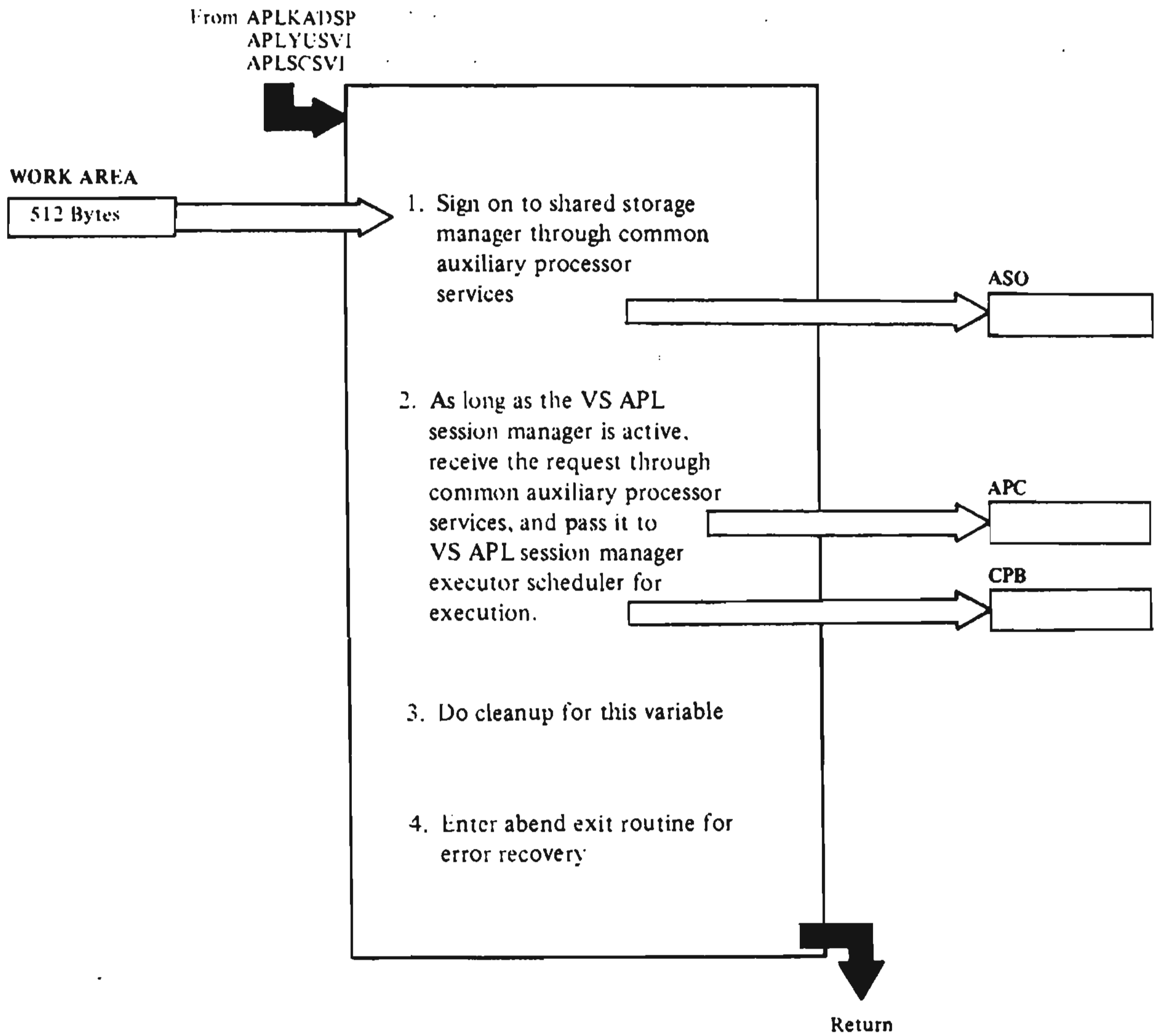
   PUT: If the translate option is set, translate character data. Issue a specification to the shared storage manager. If there is a temporary reject condition, enter a wait state and try the PUT again if the AP has so requested.

   ABORT: There are two options: abort and abort all. For abort, retract the set of variables, pass control to the invoking auxiliary processor's retract exit routine, and free the storage for this set of variables.

   For abort all, pass control to the invoking auxiliary processor's retract exit routine, once for each set of variables, to retract all sets.

6. If an abend occurs in the auxiliary processor, restore the processor's registers and call the processor's abend exit, if there is one. Then retract the variables in the set and enter a wait state.

DIAGRAM 8.4.1: VS APL SESSION MANAGER AUXILIARY PROCESSOR FOR CICS/VS, CMS, AND TSO



Notes for Diagram 8.4.1

APL120

1. Invokes common auxiliary processor services (CAPS) to issue a sign-on to shared storage manager. Control is returned to one of three entry points: OFF120, RET120, ABE120. [APL120]

2. A variable has been successfully offered and counter-offered. Local initialization (via main storage services) is done, and the CTL variable is referenced through common auxiliary processor services. If the partner specified CTL with a VS APL session manager request, the request is passed to the session manager for execution, using the CPB request block.

   The CTL variable is specified with a 2-element return code. Any text produced as a result of the execution of the request is returned as a character matrix in the DAT variable. [OFF121]

3. Control returns here from common auxiliary processor services if the partner has retracted the variable. Necessary cleanup is performed for this variable instance. [RET120]

4. This is the entry point for the occurrence of an abend. Dump services is called to dump the local work areas and registers. All variables shared with this auxiliary processor are retracted and control returns to common auxiliary processor services. [ABE120]

DIAGRAM 8.4.2: GDDM AUXILIARY PROCESSOR FOR CICS/VS, CMS, AND TSO

From APLYUSVI
APLSCSVI
APLKADSP

APC

1. Issue a sign on to common auxiliary processor services. → ASO

2. Establish addressability to storage and code.

   a) If first offer, initialize the AP communication area.

   b) Issue a GET to common auxiliary processor services for the control variable. → APC

3. If first reference, call APLXGDDM to initialize a path for the set of shared variables. → GDM

4. Interpret the request and perform appropriate action.

5. Issue a PUT to common auxiliary processor services for the DAT variable to return all character data to the user. → APC

6. Issue a PUT to common auxiliary processor services to return the return code vector and any numeric data in the control variable. → APC

7. Repeat steps 2b-6 until the set of variables is retracted.

8. If a retract or signoff request has occurred, free storage and terminate the path to GDDX.

Notes for Diagram 8.4.2

APL126 (CICS/VS, CMS, and TSO)

1. Creates a sign-on block and calls APLXAC (CMS/TSO) or APLXAK (CICS/VS) to establish the environment, and sign-on to the shared storage manager (SSM).

APL126 [OFF126]

2. This entry point is entered when a control variable has been offered by a user. The address of the auxiliary processor work area is in the APC, and is used to establish the addressability for APL126.

   a. The GDM request block is for requesting services to GDDM. This block is built by analyzing an entry in the GDDM call table. The auxiliary processor communication area is initialized to contain the number of active paths to GDDX.

   b. Issues a GET to common auxiliary processor services (CAPS) to wait for the first user request.

3. There is a single path for each pair of shared variables. This call to APLXGDDX establishes a path, and returns a unique path ID to APL126.

4. There are three categories of requests:

   a. AP Control Request: Request to the auxiliary processor to either establish an environment, or perform functions exclusive of GDDM.

   b. Zero Request: No-ops result in no action.

   c. Normal GDDM Request: Passed to GDDM, in some cases with special processing first.

5. Any character returned by GDDM is put into vector form in the DAT variable, and translated, as determined by the user.

6. The CTL variable returned is in vector form, with the first element representing the highest severity of any error incurred in the processing string.

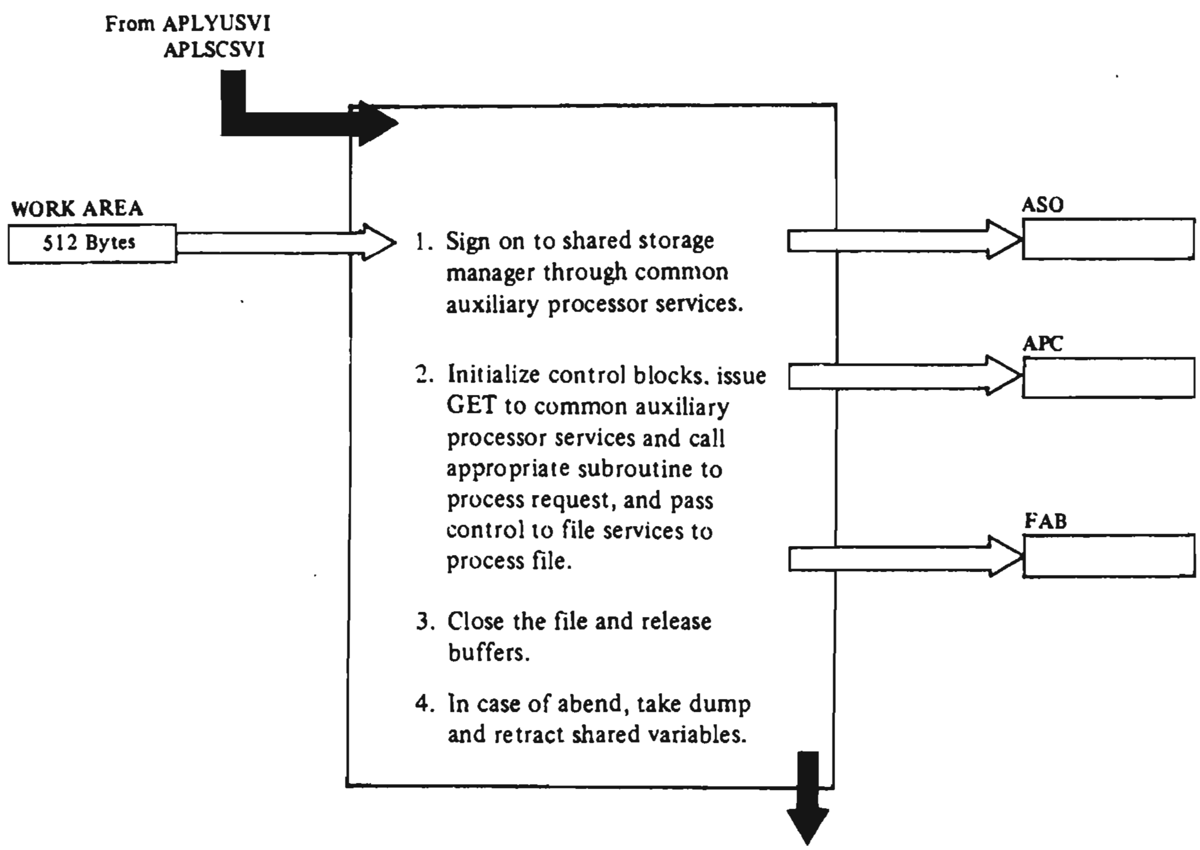
   This is followed by a four-element return code and any numeric data for each request in the string.

7. Continue to perform tasks 2b through 6 until the user retracts or signs off, or an abnormal termination occurs.

APL126 [RET126]

8. Control is passed to this label when a shared variable set is retracted by common auxiliary processor services (CAPS). Frees storage associated with this shared variable pair, and calls APLXGDDX to terminate this path, and returns to common auxiliary processor services (CAPS).

DIAGRAM 8.4.3: VS APL DATA FILE AUXILIARY PROCESSOR FOR CMS/TSO



Notes for Diagram 8.4.3

APL121

1. Invokes common auxiliary processor services (CAPS) to issue a sign-on to the shared storage manager. Control will return to one of three entry points: OFF121X, RET121X, ABE121X. [APL121X]

2. Control comes here from common auxiliary processor services when a shared variable has been successfully offered and counter-offered. Local initialization is done, and the CTL variable is referenced through common auxiliary processor services. If the partner specified CTL with a request, the appropriate action is executed. The FAB control block is used in communicating with file services. [OFF121X]

   For the sequential read request, the file is opened, each record is read sequentially and specified in the CTL variable. [SRFILE]

   For the sequential write request, the file is opened, the CTL variable is continuously referenced and written into the file. [SWFILE]

   For the create request, a new file is created if it doesn't already exist. [CFILE]

   For a drop request, the specified file is deleted. [DFILE]

   For the file size change request, the size is changed according to the specified value. [FSFILE]

   For the direct update request, the file is opened for direct processing, the DAT variable is referenced, and the corresponding record is updated. [DIRUPRD]

   For the direct read request, the file is opened for direct processing, the record is read, and specified in the DAT variable. [DIRUPRD]

   For the change share status request, an error is returned in CMS/TSO. [SHFILE]

   For the password change request, an error is returned in CMS/TSO. [PFILE]

   The control variable is specified with a 1-element return code for all operations.

3. If the partner retracted the variable, control comes from common auxiliary processor services to this entry point where the file is closed and buffers are released. [RET121X]

4. This entry point is invoked by common auxiliary processor services if an abend occurs. A dump is then taken and all shared variable instances of this auxiliary processor are retracted. [ABE121X]

Entry points are listed in alphabetic order in this section.

FOR DOS/VS: The conversion modules for DOS/VS differ from those for OS/VS. The modules are functionally the same, but the DOS/VS modules are designed to interface with DOS/VS and the OS/VS modules with OS/VS. For CMS, the OS/VS modules begin with the characters APLO; the DOS/VS modules begin with the characters APLD. For CICS/VS, the OS/VS modules begin with the characters APLKV; the DOS/VS modules begin with the characters APLKD. To avoid unnecessary repetition in this publication, only the OS/VS names are used in this publication wherever possible.

APCREATE

Module: APLPAPCD

Called By: APLPAPPR

Description: Executes service request to internal auxiliary processors AP121 and AP122 to create a VSPC file.

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

APDFN

Module: APLPAPCD

Called By: APCREATE, APDROP, APFILSIZ, APSHARE, APPASSWD, APOPEN, APVIO

Description: Converts file identification in service requests to internal auxiliary processors to VSPC standard file name.

Exit: Returns; ERSAVEAR (Error)

APDROP

Module: APLPAPCD

Called By: APLPAPPR

Description: Executes service request to internal auxiliary processors AP121 and AP122 to drop a VSPC file.

Calls: APDFN, ERMSGRTN

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

APFILSIZ

Module: APLPAPCD

Called By: APLPAPPR

Description: Executes service request to internal auxiliary processors AP121 and AP122 to change the size of a VSPC file.

Calls: APDFN, ERMSGRTN

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

APIO

Module: APLPAPCD

Called By: APLPAPPR

Description: Validates request to internal auxiliary processors AP121 and AP122 to read or update a VSPC file directly.

Exit: Returns; ERSAVEAR (Error)

APL

Module: APL

Called By: CMS

Description: Locates VS APL under CMS.

Exit: To APL DCSS or module VSAPL

APL

Module: APLYUINI

Called By: Operating system (initial entry)

Description: Initializes VS APL under TSO.

Calls: APLAINIT, APLXDUOP, APLXGYON, APLYUFXI, APLYULNE, APLYUTIO, APLYUUSR, SCCONT, SCSUPINI

Exit: Returns

APLACCBE

Module: APLACCBE

Called By: APLACPRM, APLACQRY, APLACRDA, APLACRSA

Description: Converts a binary number to EBCDIC.

Exit: Returns

APLACDSL

Module: APLACDSL

Called By: APLACPRO, APLACRDA

Description: Displays a single line on the screen.

Calls: Macro APLXG

Exit: Returns

APLACHLP

Module: APLACHLP

Called By: APLACXCM

Description: Executes the APL session manager HELP command.

Calls: APLACMSG.

Exit: Returns

APLACMDF

Module: APLACNDP

Called By: APLACPRM, APLACPRO, and APLADMSG

Description: Part of the session manager new display position routine. It updates the contents of the command field, and saves a copy of the contents in DSMCMTXT.

Calls: APLACREA, APLACRDA, APLADMSG. Macro APLXG

Exit: Returns

APLACMDX

Module: APLACMDX

Called By: APLACXCM

Description: Contains a default exit that approves all commands passed to the APL session manager.

Calls: Macro APLPATCH

Exit: Returns

APLACMER

Module: APLACQRY

Called By: APLACPRO, APLACOPY, and APLACXCM

Description: Part of the session manager command query module. For a given error number, it inserts the message, return code, and return status into the CPB, and, if appropriate displays the error message and command on the session manager screen.

Calls: APLADMSG, APLADSON

Exit: Returns

APLACNDP

Module: APLACNDP

Called By: APLACRPO, APLACXCM, APLADMSG

Description: Part of the session manager new display routine. Defines the APL session manager display at a new position on the screen.

Calls: APLXG

Exit: Returns

APLACOPL

Module: APLACOPY

Called By: APLACPRO

Description: Copies a single line to the copy destination when continuous copy of the session log is on. Called every time, a new line is added to the session log while copy is on.

Calls: APLACMER, APLADMSG. Macro APLXG

APLACOPY

Module: APLACOPY

Called By: APLACXCM

Description: Processes the 'COPY' session manager command.

Calls: APLACMER, APLACMSG, APLACSF, APLADMSG, APLAMODE, APLAUCAE, APLAUNCO. Macro APLXG

APLACPRM

Module: APLACPRM

Called By: APLACPRO when the VS APL session manager requires to get input from the terminal.

Description: This module performs a read from the terminal and enters the running mode. It then restores the screen, if necessary, and, based on user action, stacks input for APL and the VS APL session manager command processor.

Calls: APLACCBE, APLACMSG, APLACQUE, APLACSF, APLXGDDM, APLACMDF, APLACRSA, APLADSON. Macro APLXG

Exit: Returns

APLACPRO

Module: APLACPRO

Called By: APLACRCP

Description: Processes a request from the TSO, CMS, or CICS/VS executor or from any auxiliary processor, calls APLACXCM when a VS APL session manager command is to be executed, and calls APLACPRM whenever input is needed from the terminal.

Calls: APLACDSL, APLADTTM, APLACMSG, APLACNDP, APLACPRM, APLACOPL, APLADSON, APLAMODE, APLAUALT, APLAUATN, APLACQUE, APLACRDA, APLACRSA, APLACSF, APLACXCM, APLADMSG, APLAUSRX. Macro APLXG

Exit: Returns

APLACQRY

Module: APLACQRY

Called By: APLACXCM

Description: Part of session manager command query module. Formats the message returned to a query command.

Calls: APLACCBE, APLACMSG, APLACQRY, APLAUPRO, APLXGDDX, APLADMSG, APLADSON. Macro APLXG

Exit: Returns

APLACQUE

Module: APLACQUE

Called By: APLACPRO, APLACPRM, APLACXCM

Description: Maintains queues of character strings for the APL session manager, and performs the create, add, remove, purge, and delete functions on the queue.

Exit: Returns

APLACRCP

Module: APLACRCP

Called By: VS APL dispatcher

Description: This is the main entry point in the VS APL session manager request chain processor which runs as the top routine in a process separate from the TSO, CMS, or CICS/VS executor and frm the AP120. The processor functions by removing requests, one at a time, from the VS APL session manager request chain, and passing them to APLACPRO. It posts the requestor when each request is completed, and then waits until the chain is empty for a new request to be generated. It also contains an abend exit for the task.

Calls: Main storage services. Macro APLACPRO

Exit: Returns

APLACRDA

Module: APLACRDA

Called By: APLACNDP, APLACPRO, APLACXCM, APLALINE

Description: Refreshes the APL data area on the screen.

Calls: APLACCBE, APLACDSL, APLACMSG, APLACSF. Macro APLXG

Exit: Returns

APLACRSA

Module: APLACRSA

Called By: APLACNDP, APLACPRM

Description: Defines the line, status, and USA fields on the APL session manager screen.

Calls: APLACCBE, APLACMSG. Macro APLXG

Exit: Returns

APLACSF

Module: APLACSF

Called By: APLACPRO, APLACPRM, APLACRDA, APLACXCM

Description: Maintains the APL session manager's session log.

Calls: Files and maintains via VCT.

Exit: Returns. Macros APLPATCH, APLSFID, APLXDMP, APLXEND, APLXFAB, APLXMAI,

APLACXCM

Module: APLACXCM

Called By: APLACPRO

Description: Executes APL session manager commands.

Calls: APLACHLP, APLACMDX, APLACMSG, APLACNDP, APLACQRY, APLACRDA, APLACSF, APLAUPRO, APLACMER, APLACOPY, APLACQUE, APLADSMG, APLADSON, APLALINE, APLAMODE, APLAPAGE. Macro APLXG

Exit: Returns

APLAD

Module: APLAD

Called By: APLYUSVI

Description: Signs on to the shared storage manager and initiates a session manager task.

Calls: APLACRCP

Exit: The subtask terminates when a sign-off is requested.

APLADMSG

Module: APLADMSG

Called By: APLACPRO, APLACNDP, APLACQRM, APLACXCM

Description: Displays an APL session manager error or informational message.

Calls: APLACNDP, APLACMDF, APLACPRM. Macro APLXG

Exit: Returns

APLADSON

Module: APLACRDA

Called By: APLACPRO, APLACOPY, and APLACXCM

Description: Part of the session manager module to refresh the display/input area. It turns the display on, and moves the data area to the latest (new) line as part of the process.

Calls: APLACDSL, APLACSF. Macro APLXG

Exit: Returns

APLADTTM

Module: APLADTTM

Called By: APLACPRO

Description: Formats an elapsed time in APL standard format.

Exit: Returns

APLAERRM

Module: APLASCHD

Called By: TSO, CMS, or CICS/VS executor or by AP120.

Description: Requests VS APL session manager processing subcomponent to display an error message.

Exit: Returns

APLAESTK

Module: APLAESTK

Called By: APLKIFIX, APLSCFXI, APLYUFXI

Description: Sets up the executor stack for the APL session manager and makes whatever calls are necessary to stacked protocol entry points.

Calls: APLATYI, APLATYO, APLATYOI.

Exit: Returns

APLAINIT

Module: APLASCHD

Called By: TSO, CMS, or CICS/VS executor

Description: Requests VS APL session manager processing subcomponent to perform initialization processing.

Exit: Returns

APLALINE

Module: APLALINE

Called By: APLACXCM

Description: Part of the session manager line and page commands module. It executes a line command.

Calls: APLACMER, APLACRDA, APLACSF

Exit: Returns

APLAMODE

Module: APLACNDP

Called By: APLACOPY, APLACPRO, APLACXCM

Description: Part of the session manager new display position routine. It moves the cursor to the mode field, updates the mode, and forces the display of updated fields.

Calls: Macro APLXG

Exit: Returns

APLAPAGE

Module: APLALINE

Called By: APLACXCM

Description: Part of the session manager line and page commands. It executes a page command.

Calls: APLACMER, APLACRDA, APLACSF

Exit: Returns

APLAPASS

Module: APLASCHD

Called By: TSO, CMS, or CICS/VS executor or by AP120.

Description: Requests VS APL session manager processing subcomponent to prompt for a password.

Exit: Returns

APLATERM

Module: APLASCHD

Called By: TSO, CMS, or CICS/VS executor

Description: Requests VS APL session manager processing subcomponent to terminate session.

Exit: Returns

APLATYI

Module: APLASCHD

Called By: TSO, CMS, or CICS/VS executor

Description: Requests VS APL session manager processing subcomponent to perform a TYI.

Exit: Returns

APLATYO

Module: APLASCHD

Called By: TSO, CMS, or CICS/VS executor

Description: Requests VS APL session manager processing subcomponent to perform a TYO.

Exit: Returns

APLATYOI

Module: APLASCHD

Called By: TSO, CMS, or CICS/VS executor

Description: Requests VS APL session manager processing subcomponent to perform a TYOI.

Exit: Returns

APLAUALT

Module: APLASA

Called By: APLACPRO

Description: Returns a line of alternate input if it exists, or purges the alternate input stack.

Calls: NUCON (CMS nucleus). Macros APLXPROC, APLDEFN, APLPATCH, NUCON, RDTERM, APLXSTK

Exit: Returns

APLAUALT

Module: APLAYA

Called By: APLACPRO

Description: Returns a line of alternate input if it exists, or purges the alternate input stack.

Calls: Macros APLXPROC, APLDEFN, APLPATCH

Exit: Returns

APLAUALT

Module: APLAK

Called By: APLACPRO (common session manager module)

Description: Part of the session manager CICS/VS-dependent, SP-entry, routines. It is called by the session manager to determine if the subsystem has any alternate input available. Alternate input may be generated by the input invocation option or by AP139.

Exit: Returns

APLAUATN

Module: APLAKP (CICS/VS)

Called By: APLXGKT

Description: Main and only entry point to the session manager system-dependent, non-stack-processor entry, CICS/VS routines. It handles asynchronous terminal activity by analyzing the asynchronous input to determine if it is a "real" attention or an asynchronous input to the session manager.

Exit: Returns

APLAUATN

Module: APLASP (CMS)

Called By: APLXGKT

Description: Main and only entry point to the session manager system-dependent, non-stack-processor entry, CICS/VS routines. It handles asynchronous terminal activity by analyzing the asynchronous input to determine if it is a "real" attention or an asynchronous input to the session manager.

Exit: Returns

APLAUATN

Module: APLAYP (TSO)

Called By: APLXGKT

Description: Main and only entry point to the session manager system-dependent, non-stack-processor entry, CICS/VS routines. It handles asynchronous terminal activity by analyzing the asynchronous input to determine if it is a "real" attention or an asynchronous input to the session manager.

Exit: Returns

APLAUCAE

Module: APLAK (CICS/VS)

Called By: APLACOPY (session manager command module)

Description: Part of the session manager CICS/VS-dependent, stack processor-entry, routines. It is called by the session manager to determine if using a copy ID would destroy data in any copy files which had previously been created within the same ID.

Exit: Returns

APLAUCAE

Module: APLAS (CMS)

Called By: APLACOPY (session manager command module)

Description: Part of the session manager CICS/VS-dependent, stack processor-entry, routines. It is called by the session manager to determine if using a copy ID would destroy data in any copy files which had previously been created within the same ID.

Exit: Returns

APLAUCAE

Module: APLAY (TSO)

Called By: APLACOPY (session manager command module)

Description: Part of the session manager CICS/VS-dependent, stack processor-entry, routines. It is called by the session manager to determine if using a copy ID would destroy data in any copy files which had previously been created within the same ID.

Exit: Returns

APLAUNCO

Module: APLAK (CICS/VS)

Called By: APLACOPY

Description: Part of the session manager CICS/VS-dependent, SP-entry, routines. It is called by the session manger to determine if the subsystem supports opening the same ID multiple times.

Exit: Returns

APLAUNCO

Module: APLAS (CMS)

Called By: APLACOPY

Description: Part of the session manager CICS/VS-dependent, SP-entry, routines. It is called by the session manger to determine if the subsystem supports opening the same ID multiple times.

Exit: Returns

APLAUNCO

Module: APLAY (TSO)

Called By: APLACOPY

Description: Part of the session manager CICS/VS-dependent, SP-entry, routines. It is called by the session manger to determine if the subsystem supports opening the same ID multiple times.

Exit: Returns

APLAUPRO

Module: APLAS

Called By: APLACXCM, APLACQRM

Description: Opens a file and writes or reads records for an APL session manager profile (CMS only).

Calls: APLSCOPT. Macros FSOPEN, FSCLOSE, FSREAD, FSWRITE, FSSTATE

Exit: Returns

APLAUPRO

Module: APLAY

Called By: APLACXCM

Description: Opens a file and writes or reads records for an APL session manager profile (TSO only).

Calls: APLYUUSR, APLYDAIR. Macros: OPEN, PUT, CLOSE, GET

Exit: Returns

APLAUPRO

Module: APLAK (CICS/VS)

Called By: APLACQRY (common session manager module), APLACXCM (common session manager module which calls APLACQRY)

Description: Part of the session manager CICS/VS-dependent stack processor-entry, routines. It provides session manager support (open a file, read records from a file, close a file). These actions are passed via the PRB (profile request) control block.

Calls: APLXFKFL

Exit: Returns

APLAUSRX

Module: APLAUSRX

Called By: APLACPRO

Description: Contains a user authorization exit to allow optional rejection of the use of the session manager for some users.

Exit: Returns

APLAXCMD

Module: APLASCHD

Called By: AP120

Description: Requests the VS APL session manager to process a text string as a command.

Exit: Returns

APLFXIIM

Module: APLKIFIX

Called By: APLFXIIM

Description: Part of the interpreter interface provided by the CICS/VS executor. Serves as an entry point from the interpreter to the executor to handle service requests.

Calls: Entry points KRSTEX, KCQZ, APLKFDPY, KCATOFF, KCTIME, KCQAI, KCDELAY, KCTABS, KCWIDTH, KCMBL, KCTRAN, KCOPI, KCOPO, KCOPZ, KCDUMP, KFOFF, KCSYSER, KCQUOTA, KLOAD, KCOPA, KSAVE, KDROP, KLIB, KCLEAR, KWSID, KPASS, APLKISVI. Macros APLKHIST, DFHKC, APLKWAIT, DFHTR

Exit: KTOINTER, KADSP8.

APLFXIIM

Module: APLPFXIM

Called By: Many interpreter and translator routines

Description: Sole entry point from interpreter to VSPC executor; saves interpreter's environment and calls routine to handle service request.

Calls: PC(...): routines

Exit: APLIINIT

APLFXIIM

Module: APLSCFXI

Called By: Many interpreter and translator routines

Description: Sole entry point from interpreter to CMS executor; saves interpreter's environment and calls routine to handle service request.

Calls: SC(...): routines

Exit: APLIINIT

APLFXIIM

Module: APLYUFXI

Called By: VS APL interpreter, shared storage manager (SSM)

Description: Executes a service request for the VS APL interpreter or shared storage manager (SSM). Its main tasks are 1) preserve the caller's environment, 2) determine the type of request by table lookup, 3) call supervisor routine that handles service request, and 4) always return control to APLIINIT in interpreter (TSO).

Calls: Service Request Execution Routine

Exit: Returns to APLIINIT in module APLITINI of interpreter

APLFXIIM

Module: APLYUIIM

Called By: Many interpreter and translator routines.

Description: This is the TSO/VSPC executor call switch. It intercepts all calls from the VS APL interpreter and routes control to either the TSO or VSPC executor. This module is used only when VS APL/TSO and VS APL/VSPC have been link-edited into a single load module.

Calls: APLVSPC, APLTSO

Exit: Control passed to the appropriate executor

APLFXIIM

Module: APLFXIIM

Called By: Many interpreter and translator routines.

Description: In a CICS/VS environment, APLFXIIM intercepts executor calls from the interpreter, and passes them to the CICS/VS which is pointed to by PTHPARM1.

Exit: Returns

APLIINIT

Module: APLITINI

Called By: APLPCENT, APLFXIIM

Description: Sole entry point to interpreter from executor; receives control after executor has handled service request; restores interpreter's environment including changes resulting from workspace relocation.

Calls: APLFXIIM, ITLIBMSG, IATABREF, IASVOFF, ITSHV

Exit: If sign-on, APLFXIIM with YYCLEAR service request or ITCMLOAD; if load or clear, ITINPINI; if copy source, ITCMCOPO; if error, ITSYSERR; also to next instruction after service request (address in WSMNSI).

APLKADEF

Module: APLKADEF

Called By: Entry points APLKEMGR, APLKSSR

Description: Part of the CICS/VS executor. Determines if the user of the auxiliary processor is authorized to access the named resource in the requested fashion.

Exit: Returns

APLKADSP

Module: APLKADSP

Called By: Entry point APLKASON

Description: Part of the CICS/VS executor. Controls the user task.

Calls: Entry points APLKIFON, KADSP8, KYYOFF, APLACRCP, APLKISVE, APLACRCP, APLXGKON, KMARCO. CICS/VS macros DFHKC (WAIT), DFHSC (GETMAIN), DFHPC (SETXIT), APLKSON, APLKSOF

Exit: DFHPC (RETURN)

APLKAGBL

Module: APLKAGBL

Called By: Entry points APLKASON, KABOOTS

Description: Part of the CICS/VS executor. Initializes and shuts down the global table.

Calls: Entry points APLKLIBI, APLKLIBT, KINIEX, APLKSSMR, KDPFAB, KAPFXIT. Macros APLKTOFF. CICS/VS macros DFHPC (LOAD, DELETE), DFHKC (ENQDEQ)

Exit: DFHPC (RETURN)

APLKAHST

Module: APLKAHST

Called By: APLKHST macro

Description: Part of the CICS/VS executor. Records a histogram event.

Exit: Returns

APLKAMIX

Module: APLKAMIX

Called By: CICS/VS. Used when APLKASON attaches APLU task.

Description: Provides a CICS/VS mixed mode (command level/macro level) environment. It may be employed as the primary entry point for any CICS/VS task.

Calls: Entry point APLKMIX in APLKADSP.

Exit: Returns

APLKASON

Module: ALPKASON

Called By: CICS/VS

Description: Part of the sign-on process performed by the CICS/VS executor. Initiates a user APL session.

Calls: Entry points APLKADSP, APLKAGBL, APLKAGBL, APLKLIBR. Macros APLKT (TRAN). CICS/VS macros DFHKC (ATTACH, WAIT), DFHPC (RETURN, ABEND, LOAD, XCTL, SETXIT), DFHDC, DFHIC (GET, GETIME), DFHSC (GETMAIN), DFHTC (PUT, GET), DFHFC (RELEASE)

Exit: DFHPC (RETURN) to APLKTCTL, or to APLXGKT.

APLKEHCP

Module: APLKEHCP

Called By: Entry point APLKEMGR via CICS/VS macro DFHIC (PUT)

Description: Part of the destination management services provided by the CICS/VS executor. Provides support for the 3270 printer.

Calls: Entry point KTRTRAN. CICS/VS macros DFHSC (GETMAIN, FREEMAIN), DFHPC (RETURN, ABEND, SETXIT, LOAD), DFHIC (GET), DFHTC (PUT)

Exit: DFHPC (RETURN)

APLKEMGR

Module: APLKEMGR

Called By: Entry points APLXGKU, KTSLINE, APL132K, KTRHC

Description: Part of the destination management services provided by the CICS/VS executor. This is the initial entry point for all destination management service requests. Based on request, routes control to the appropriate service routine.

Calls: Entry points APLKEHCP, APLKADEF, KETWRITE. Macros APLKT (TRAN). CICS/VS macros DFHPC (LOAD), DFHIC (PUT), DFHKC (ENQ, DEQ), DFHSC (GETMAIN, FREEMAIN), DFHTC (LOCATE), DFHTD (PUT, GET, LOCATE)

Exit: Returns

APLKIFON

Module: APLKIFIX

Called BY: Entry point APLKADSP

Description: Part of the interpreter interface provided by the CICS/VS executor. Sets up a stack for the interpreter interface modules to use and an abend exit for the user transaction.

Calls: Entry points APLKLUIT, APLAIMT. Macros APLKIST, APLKEXIT, APLKMAIN (GET). CICS/VS macros DFHPC (ABEND)

Exit: KTOINTER, caller (error)

APLKISVI

Module: APLKISVI

Description: Part of the CICS/VS executor shared storage manager interface. Executes the following YYCODE service requests: YYSACC (set access control vector); YYSCIOY (copy); YYSOFF (sign off); YYSOFFER (offer); YYSON (sign on); YYSQUERY (query); YYSREF (reference); YYSRET (retract); YYSSPCE (specification).

Calls: Entry point KADEPON. Macros APLKSSMR, APLKWAIT, APLKMAIN

Exit: Returns

APLKLIBF

Module: APLKLIBF

Called By: Entry points APL121K, APLXFKFL

Description: Part of the library management services provided by the CICS/VS executor. Manages the data to and from data buffers for internal APL files under execution of the user task.

Calls: Entry point APLKLIBR. Macros APLKEXIT, APLKHIST, APLKG (LIBSERV, TYPE=WLIB, WDIR, UDIR, RLIB, CFILE, DFILE, UFILE) APLKWAIT

Exit: Returns

APLKLIBG

Module: APLKLIBG

Called By: Entry point LIBSTART

Description: Part of the library management services provided by the CICS/VS executor. Routes control to the appropriate subroutine for all synchronous I/O library requests.

Calls: Entry points KGWDIR, KGUDIR, KGLOAD, KGSAVE, KGDROP, KGCFILE, KGDFILE, KGRLIB, KGUFILE, KGWLIB. CICS/VS macros DFHPC (SETXIT, RETURN, ABEND), DFHKC (ENQ, DEQ, WAIT)

Exit: Returns

APLKLIBI

Module: APLKLIBB

Called By: Entry point APLKAGBL

Description: Part of the library management services provided by the CICS/VS executor. Prepares the APL library data set for processing, defines storage for and loads the free space bit maps from the library, and initializes the global table fields owned by the library.

Calls: Entry points KLGET, KLOPEN. CICS/VS macros DFHOC (CLOSE), DFHFC (GET, RELEASE), DFHSC (GETMAIN)

Exit: Returns

APLKLIBR

Module: APLKLIBR

Called By: APLKLIBU, APLKASON, APLKLIBF and APLKLIBG (all via GBLRDIR)

Description: Main and only entry point to the CICS/VS APL library services-read directory. It performs the synchronous I/O to read a record from the APL directory.

Calls: Macro DFHFC

Exit: Returns

APLKLIBR

Module: APLKLIBG

Called By: Entry points APLKASON, APLKLIBF, KCOPA, KLOAD, KGCFILE

Description: Part of the library management services provided by the CICS/VS executor. Reads an APL directory record from the APL directory data set.

Calls: CICS/VS macro DFHFC (GET)

Exit: Returns

APLKLIBT

Module: APLKLIBB

Called By: Entry point APLKAGBL

Description: Part of the library management services provided by the CICS/VS executor. Closes the APL library data set for APL processing and reopens it as a CICS/VS data set.

Calls: Entry point KLCLOS. CICS/VS macro DFHOC (OPEN)

Exit: Returns

APLKLUIT

Module: APLKLIBC

Called By: Entry point APLKIFON

Description: Part of the library management services provided by the CICS/VS executor. Provides the user with workspace when he initially signs on. Defines the initial workspace and reads the HI message records from the APL directory. Requests by a call to entry point KYYTYOI that the HI message records by displayed.

Calls: Entry points APLKSPEN, APLXERRM, KYXTYOI. Macros APLKEXIT, APLKPROC, APLKPOP. CICS/VS macro DFHSC (GETMAIN)

Exit: Returns

APLKLUTM

Module: APLKLIBC

Called By: Entry point KFOFF

Description: Part of the library management services provided by the CICS/VS executor. Returns workspace storage to CICS/VS when the user logs off APL.

Calls: Macro APLKEXIT. CICS/VS macro DFHSC (FREEMAIN)

Exit: Returns

APLKPFAP

Module: APLKASTB

Called By: DOS/VS page supervisor

Description: Part of the CICS/VS executor. For DOS/VS only. Allows overlap of page faults that occur during execution of the interpreter.

Exit: APLKPFOH

APLKPFOH

Module: APLKASTB

Called By: Entry point APLKPFAP

Description: Part of the CICS/VS executor. For DOS/VS only, puts the current user into a wait state so CICS/VS can dispatch other users.

Calls: Entry points KRSTEX, KSETEX. CICS/VS macro DFHKC (WAIT)

Exit: To interpreter at point of page fault

APLKSPRG

Module: APLKVEXC

Called By: The operating system. (This is the service program's entry point. It is specified on the EXEC statement.)

Description: Part of the APL library service program for CICS/VS. Drives the utility. Does initialization; calls KSPPIN to open the print and reader data sets; reads a command, calls KSPCMD to analyze it; calls KSPINT to open the KSPINT to open the necessary data sets; calls the proper command processor (KSPAUT, KSPCPY, KSPFMT, KSPIMP, OR KSPEXP); and calls KSPTRM to close the unique data sets associated with the command. This process is repeated until there is no more data. It then closes the system data sets.

Calls: Entry points KSPAUT, KSPCMD, KSPCPY, KSPDOS, KSPEXP, KSPIMP, KSPINT, KSPMSG, KSPPIN, KSPTRM, KSPFMT. OS macros GET, PUT, CLOSE, FREEMAIN. DOS macros CLOSE, EXCP, PUT, WAIT, FREEVIS

Exit: Returns

APLKSSR

Module: APLKSSVP

Called By: Entry point BOOTSTR Macros APLKSON, APLKSOF, APLKREF, APLKSPC, APLKCPY, APLKQRY, APLKOFR, APLKRET, APLKACHK, APLKACC

Description: Part of the CICS/VS shared storage manager. Handles all shared variable requests issued by module APLKISVI and the auxiliary processors.

Calls: Entry points APLKADEF, KCASE2Q, KCASE3Q, KCLEANUP, KFREESP, KGCOL, KGETSPAC, KIDSETUP, KPOSTWAI, KPPSEARC, KPROCOFF, KRETSUB, KSEIZE, KSINGAL

Exit: Returns

APLKSSUB

Module: APLKSSUB

Called By: Entry point KABOOTS

Description: Part of the CICS/VS shared storage manager. Obtains space for and initializes the shared memory.

Calls: CICS/VS macro DFHSC (GETMAIN)

Exit: Returns

APLKTCTL

Module: APLKTCTL

Called By: CICS/VS macros DFHIC (INITIATE) or DFHPC (XCTL)

Description: Part of the terminal management services provided by the CICS/VS executor. Handles terminal input operations and routes output operations to module APLKTCWR. Runs under the terminal transaction, a separate transaction from the APL user transaction. Processes requests originally initiated by the APLKTERM macro (type=requests of READ, WRITE, or RESTORE) issued in the APL user transaction. Also handles any input received when no APLKTERM request is being processed (when the terminal is in listen state).

Calls: Entry point APLKTCNR. Macros APLKT (LOCREQ, FINDF, TRAN), APLKTRCE, APLKHIST. CICS/VS macros DFHPC (RETURN), DFHTC (READ, WRITE), DFHSC (GETMAIN), DFHPC (SETXIT, ABEND, RETURN, LOAD)

Exit: DFHPC (RETURN)

APLKTCWR

Module: APLKTCWR

Called By: APLKTCTL

Description: Part of the terminal management services provided by the CICS/VS executor. Handles terminal output operations. Runs under the terminal transaction, a separate transaction from the APL user transaction.

Calls: Macros APLKT (TRAN), APLKTRCE. CICS/VS macros DFHSC (GETMAIN, FREEMAIN), DFHTC (WRITE)

Exit: APLKPOP

APLPAPAC

Module: APLPAPAB

Called By: PCSACC

Description: Sets access control vector for a variable shared with an internal auxiliary processor.

Exit: Returns; ERSAVEAR (Error)

APLPAPOF

Module: APLPAPAB

Called By: PCSOFFER

Description: Processes offer to share a variable with an internal auxiliary processor.

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

APLPAPPR

Module: APLPAPAB

Called By: PCSCOPY, PCSREF, PCSSPEC

Description: Processes copy, reference, and specification of a variable shared with an internal auxiliary processor.

Calls: APCREATE, APDROP, APFILSIZ, APIO, APOPEN, APPASSWD, APSHARE, APVIO, PRDDIR, PRDSEQ, PWRITE, ERMSGRTN, FSMFORMT, FSMWRITE, FSMREAD, FSMGET, FSMFORM, FSMSETC, FSMBUZZ FSMSUB3, FSMMTYPE, FSMMINT, FSMHCOPY, GDDMRCTL, GDDMSCTL, GDDMSDAT

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

APLPAPRT

Module: APLPAPAB

Called By: PCSRET

Description: Processes retraction of a variable shared with internal auxiliary processor.

Calls: ERMSGRTN, GDDMCRET, GDDMSOFF

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

APLPAPSF

Module: APLPAPAB

Called By: PCSOFF

Description: Retracts variables shared with internal auxiliary processors when user signs off.

Calls: ERMSGRTN, GDDMSOFF

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

APLPCENT

Module: APLPCOEX

Called By: VSPC Foreground Interface

Description: Serves as the sole entry point from VSPC to VS APL; checks for purpose of entry: for initialization, initializes control areas and VS APL workspace area; for asynchronous event, checks attentions, cancel output, program checks, forceoffs, line drop, and other termination situations.

Calls: ERTIMDAT

Exit: APLIINIT with a YYON service request (initialization); returns to VSPC (asynchronous events).

APLPCOAP

Module: APLPCOAP

Called By: VSPC executor modules APLSHVR and APLPAPAB reference this module

Description: List of auxiliary processors, relating the VSPC identification number to its corresponding VS APL VSPC auxiliary-processor name. Contains no executable code.

APLSCSSI

Module: APLSCSSI

Called By: CMS (original entry)

Description: This module executes in the CMS transient area. It locates the proper VS APL processing module and passes control to it.

Calls: Macros DMSEXS, DMSKEY, DIAG, FSSTATE, WRTERM, LINEDIT, NUCON, REGEQU, APLPATCH

Exit: Either to a shared segment or to the disk-resident VSAPL module.

APLSHACC

Module: APLSHACC

Called By: ASVPSERV

Description: Resets access control vector for one partner; creates new combined access control vector for both partners.

Calls: APLSHSRD, APLSHPST

Exit: Returns

APLSHBPB

Module: APLSHBPB

Called By: APLSHSON

Description: Constructs processor block in shared memory when a processor signs on to the shared variable processor.

Calls: APLSHGET

Exit: Returns

APLSHBVB

Module: APLSHBVB

Called By: APLSHOFR

Description: Constructs variable block in shared memory when a new variable is successfully offered to the shared variable processor.

Calls: APLSHGET

Exit: Returns

APLSHCPY

Module: APLSHCPY

Called By: ASVPSERV

Description: Provides latest value of a shared variable regardless of the current access state.

Calls: APLSHPUT, APLSHSRD

Exit: Returns

APLSHGET

Module: APLSHGET

Called By: APLSHBPB, APLSHBVB, APLSHOFR

Description: Gets a block of virtual storage from shared memory.

Calls: APLSHPUT

Exit: Returns

APLSHOFR

Module: APLSHOFR

Called By: ASVPSERV

Description: Processes a request to share a single variable; finds VAB for offer and fills other partner; constructs new VAB for new offer.

Calls: APLSHBVB, APLSHGET, APLSHSRD, APLSHPST

Exit: Returns

APLSHPST

Module: APLSCSVI

Called By: APLSHREF, APLSHACC, APLSHOFR, APLSHSPC, APLSHSUB, APLSHPUT

Description: Posts ECB for auxiliary processor associated with shared variable.

Exit: Returns

APLSHPUT

Module: APLSHPUT

Called By: APLSHCPY, APLSHREF, APLSHSOF, APLSHSPC, APLSHSUB, APLSHGET

Description: Returns block of virtual storage to shared memory; fills area with zeros.

Calls: APLSHPST

Exit: Returns

APLSHQUE

Module: APLSHQRE

Called By: ASVPSERV

Description: Provides information about a shared variable; fills in fields of SCV.

Exit: Returns

APLSHREF

Module: APLSHREF

Called By: ASVPSERV

Description: Provides latest value of a shared variable if not interlocked; moves value to buffer whose address is in SCVVALUE.

Calls: APLSHSRD, APLSHPST, APLSHPUT

Exit: Returns

APLSHRET

Module: APLSHRET

Called By: ASVPSERV

Description: Terminates offer of shared variable by calling processor of variable described in SCV.

Calls: APLSHSRD, APLSHSUB

Exit: Returns

APLSHSOF

Module: APLSHSOF

Called By: ASVPSERV

Description: Disconnects processor from shared variable processor; retracts all variables offered under processor's ID.

Calls: APLSHPUT, APLSHSUB

Exit: Returns

APLSHSON

Module: APLSHSON

Called By: ASVPSERV

Description: Connects a processor to the shared variable processor.

Calls: APLSHBPB

Exit: Returns

APLSHSPC

Module: APLSHSPC

Called By: ASVPSERV

Description: Specifies a new value for a shared variable or informs caller that value specified by a partner is waiting.

Calls: APLSHPUT, APLSHSRD, APLSHPST

Exit: Returns

APLSHSRD

Module: APLSHSRD

Called By: APLSHCPY, APLSHRET, APLSHSPC, APLSHREF, APLSHACC, APLSHOFR

Description: Searches index block for variable block with offer number equal to offer number in SCV; returns with pointer to block or error indication.

Exit: Returns

APLSHSUB

Module: APLSHSUB

Called By: APLSHRET, APLSHSOF

Description: Terminates an offer for calling routine of a variable.

Calls: APLSHPUT, APLSHPST

Exit: Returns

APLXACSO

Module: APLXAC

Called By: APL120, APL121, APL126

Description: Establishes the environment for the AP and sign-on to the shared storage manager (CMS/TSO).

Calls: Main storage management services, shared storage manager, and abend exit services. Macros: APLXSON, APLXMAIN, APLXADUM, APLXSFRE

Exit: Calls the offer exit return in the auxiliary processor.

APLXACSV

Module: APLXAC (CMS)

Called By: APL120, APL121, APL126

Description: Provides the services GET, PUT, COPY, AUTHCHECK, and ABORT between an auxiliary processor and the shared storage manager.

Calls: Main storage management services, shared storage manager, and abend exit services. Macros APLXMAIN, APLXSON

Exit: Returns

APLXACSV

Module: APLXAC (TSO)

Called By: APL120, APL121, APL126

Description: Provides the services GET, PUT, COPY, AUTHCHECK, and ABORT between an auxiliary processor and the shared storage manager.

Calls: Main storage management services, shared storage manager, and abend exit services. Macros APLXMAIN, APLXSON

Exit: Returns

APLXAKSO

Module: APLXAK

Called By: APL120, APL126

Description: Establishes the environment for the AP (CICS/VS).

Calls: Main storage management services, shared storage manager, abend exit services, and dump services, session manager message services and stack services.

Exit: Calls the offer exit return in the auxiliary processor.

APLXAKSV

Module: APLXAK

Called By: APL120, APL126

Description: Part of common AP services for CICS/VS. It provides the following services between an auxiliary processor and the shared storage manager in the CICS/VS environment: GET (reference the data that the user has specified in a shared variable), PUT (specify the data from the auxiliary processor buffer to shared sotrage), COPY (obtain the latest value of a shared variable without altering the setting of the current access state), AUTHCHECK (search the authorization table to locate the authorization code associated with the resource named, and ABORT (retract the variables in this set and pass control to the auxiliary processor's retract exit routine.

Calls: Main storage services, shared storage manager, ABEND exit services, dump services, session manager message routine, and stack management services. Macros APLKACHK, APLXMAIN, APLXBXIT, APLXDUMP, APLKOFR, APLKCPY, APLKREF, APLKSPC, APLKWAIT, APLKSCZ, APLXSTK

Exit: Returns

APLXAINP

Module: APLXASD (CMS)

Called By: APLXAC

Description: Analyzes input parameters for common AP services.

Calls: APLXMSSG (CMS) Macro APLDEFN

Exit: Returns

APLXAINP

Module: APLXAYD (TSO)

Called By: APLXAC

Description: Analyzes input parameters for common AP services.

Calls: APLXMYSG (TSO) Macros APLXMAIN

Exit: Returns

APLXAMSG

Module: APLXASD (CMS)

Called By: APLXAC

Description: Displays messages for common AP services.

Calls: APLYULNE Macros APLXEDIT, APLXSTK, APLXMAIN, APLXAFRE

Exit: Returns

APLXAMSG

Module: APLXAYD (TSO)

Called By: APLXAC

Description: Displays messages for common AP services.

Calls: APLERRM, (TSO) Macros APLKEDIT, APLXSTK, APLXMAIN, APLXAFRE

Exit: Returns

APLXBACK

Module: APLXSTAK

Called By: All stack processor entry points

Description: Returns to caller of SP module.

Calls: Common main storage services

Exit: Returns to instruction following call in calling program

APLXBSAB

Module: APLSCSVI

Called By: Various executor routines.

Description: Provides system-independent interface for abend services to the CMS executor and auxiliary processors.

Calls: Macro ABEND

Exit: Abnormal termination

APLXBSXT

Module: APLSCSVI

Called By: Many executor routines.

Description: Provides an abend exit service through a system-independent interface.

Calls: Macro STAE

APLXBYAB

Module: APLYUSVI

Called By: Various executor routines and auxiliary processors

Description: The caller requests that a particular abend be issued on his behalf by placing a binary abend code in register 1. This routine provides an abend request service through a system-independent interface (TSO).

Calls: Macro ABEND

Exit: Abnormal termination

APLXBYXT

Module: APLYUSVI

Called By: Various executor routines and auxiliary processors

Description: The caller requests that a particular routine be given control when an abend occurs. This routine provides an abend exit service through a system-independent interface; it also contains the ESTAE exit and retry routines (TSO).

Calls: Macros APLPTRGT, APLXXPTX, APLTSOGL, ESTAE, IHASDNA, SETRP

Exit: Returns

APLXCALL

Module: APLXSTAK

Called By: APLXSTAK stub code

Description: Calls an SP module.

Calls: Common main storage services

Exit: Returns to requested entry point

APLXDKMP

Module: APLXDKMP

Called By: Available for general use by any VS APL module

Description: Main and only entry point to the VS APL CICS/VS dump services module. It provides a system independent interface to the CICS/VS executor (and auxiliary processors) for common dump services. The CICS/VS executor command is employed for each range of addresses to be dumped.

Calls: Macro DFHDC

Exit: Returns

APLXDUCL

Module: APLXDUMP

Called By: APLYUINI, APLSCINI

Description: Closes DUMP data set at termination.

Calls: APLXMSSG, APLXMYSG. Macros IHADCB, DCB, OPEN, CLOSE, SNAP, APLXMAIN

Exit: Returns

APLXDUMP

Module: APLXDUMP

Called By: Various executor routines

Description: Provides system-independent interface for dump services to the CMS and TSO executors and auxiliary processors. The SNAP macro is used to request a range of addresses to be dumped to the APLDUMP DD file.

Calls: MAINS. Macros APLPATCH, IHADCB, DCB, OPEN, CLOSE, SNAP, APLXMAIN, DIAG

Exit: Returns

APLXDUOP

Module: APLXDUMP

Called By: APLYUINI, APLSCINI

Description: Called at initialization to open the DUMP data set.

Calls: Macros APLPATCH, IHADCB, DCB, OPEN, CLOSE, SNAP, APLXMAIN

Exit: Returns

APLXFINT

Module: APLXFSFL

Called By: APLSCINI

Description: Initializes buffers for AP 121 files and scrolling (CMS only).

Calls: APLXMSSG. Macros APLPATCH, APLSFID, APLXDMP, APLXEND, APLXFAB, APLXMAI, APLXMAIN, APLXMOD, APLXPROC, APLXPTH, APLXSTAK, FSREAD, FSWRITE

Exit: Returns

APLXFINT

Module: APLXFYFL

Called By: APLYUINI

Description: Initializes buffers for AP 121 files and scrolling (TSO only).

Calls: APLXMYSG. Macros ACB, APLXMAIN, APLXMOD, APLXSTAK, FSREAD, FSWRITE

Exit: Returns

APLXFKFL

Module: APLXFKFL

Called By: APLACSF

Description: This module provides a map, for the CICS/VS file system, from release 4 stack processors to release 3 register requirements and stack usage.

Calls: APLKLIBF

Exit: Returns

APLXFSFL

Module: APLXFSFL

Called By: APLSCINI, APL121, APLACSF

Description: Manages the movement of data to and from buffers for AP 121 files and scrolling (CMS only).

Calls: APLXDUMP, APLXMSSG, APLSCFID. Macros APLSFID, APLXMAIN, APLXMOD, CLOSE, ENDREQ, ERASE, FREEMAIN, GET, GETMAIN, IFGACB, IFGRPL, PUT, RPL

Exit: Returns

APLXFTRM

Module: APLXFSFL

Called By: APLSCINI, APL121, APLACSF

Description: Terminates a buffer service request for AP 121 files and scrolling (CMS only).

Calls: APLXDUMP, APLXMSSG. Macros APLSFID, APLXMAIN, APLXMOD, APLXSTAK, FSREAD, FSWRITE

Exit: Returns

APLXFTRM

Module: APLXFYFL

Called By: APLYUINI

Description: Terminates a buffer service request for AP 121 files and scrolling (TSO only).

Calls: APLXMYSG. Macros ACB, APLXFAB, APLXMAIN, APLXMOD, APLXSTAK, CLOSE, ENDREQ, ERASE, FREEMAIN, GET, GETMAIN, IFGACB, IFGRPL, PUT, RPL

Exit: Returns

APLXFYFL

Module: APLXFYFL

Called By: APLYUINI, APL121, APLACSF

Description: Manages the movement of data to and from buffers for AP 121 files and scrolling (TSO only).

Calls: APLXMYSG. Macros ACB, APLXMOD, APLXSTAK, CLOSE, ENDREQ, ERASE, FREEMAIN, GET, GETMAIN, IFGACB, IFGRPL, PUT, RPL

Exit: Returns

APLXGCAT

Module: APLXGCAT

Called By: Operating system or GDDM

Description: This is the attention processing module for CMS and TSO.

Exit: To routine in PTXATTN

APLXGCHC

Module: APLXGCHC

Called By: APLXGCOM

Description: This is the common APLXGDDM hardcopy request processing module which handles the following APLXG requests: FSOPEN, FSCLS, FSCOPY, FSLOG, GSCOPY, and QDEST.

Calls: GDDM APL print services, main storage services, APLXGDDM system-dependent modules (APLXGKU, APLXGS, or APLXGY). Macro APLXSTK

Exit: Returns

APLXGCOM

Module: APLXGCOM

Called By: APL session manager, AP126

Description: This is the GDDM interface module. APLXGCOM is the main entry point for all APLXG macro processing, and contains all processing routines common across all systems, except for hardcopy request support and attention support. Three types of requests are processed: APL special requests, GDDM requests with special considerations and pass-through requests.

Calls: APLXGCHC, ADMASP (GDDM entry point), and the following entry points defined through the VCT: GDDXE, DUMPX, MAINS, STKAB.

Exit: Returns

APLXGKON

Module: APLXGKON

Called By: APLKADSP (VS APL dispatcher)

Description: Contains CICS/VS-only support for the startup of the CICS/VS GDDX process, the synchronization of requests from the session manager, and instances of AP126.

Calls: APLXGCOM

Exit: Returns to dispatcher

APLXGKR

Module: APLXGKR

Called By: APLXGKRR

Description: Main and only entry point in the GDDX CICS/VS terminal manager retrofit module that converts GDDM calls made by the session manager into release 3 terminal manager calls, thus allowing the session manager to run when GDDM is not available.

Calls: Macros APLKEXIT, APLKMAIN, APLKTERM, DFHPC TYPE=ABEND and DFHIR TYPE=ENTRY

Exit: Returns

APLXGKRQ

Module: APLXGKRQ

Called By: Macro APLXG and APL126 (the session manager modules)

Description: Part of GDDX CICS/VS-only user transaction I/O support. It is invoked in CICS/VS via the VCTGDDX pointer when a request for APLXGDDM service is issued (through the APLXG macro). It then signals the APLXGKON routine to perform the request under a GDDX task and waits for it to do so.

Exit: Returns

APLXGKRR

Module: APLXGKRR

Called By: APLXGCOM, APLXGCHC

Description: Main and only entry point to the GDDX T.M. retrofit router module. It routes requests from APLXGCOM or APLXGCHC to APLXGKU if GDDM is to be used in the session, or to APLXGKR if the Release 3 terminal manager is to be used.

Calls: APLXGKR, APLXGKU

Exit: Returns

APLXGKT

Module: APLXGKT

Called By: CICS/VS as a result of an EXEC CICS/VS start command in APLXGKU, or an XCTL in APLKASON (CICS/VS sign on module)

Description: Main and only entry point to the root CICS/VS terminal transaction support module for GDDX. It contains CICS/VS-only routines for APLXG requests which must be executed from the terminal transaction. These comprise the following: SPINIT, FSFRCE, ASREAD, and FSSHOW. APLXGKT notifies the user transaction, as needed, of request completion, and synchronizes with the user transaction to avoid overlapping of calls to GDDM. APLXGKT also supplies the attention-handling support for CICS/VS.

Calls: ADMASP, APLAUATN. CICS/VS command level: ABEND, ADDRESS, ASSIGN, ENTER, HANDLE, POST, RECEIVE, RETURN, RETRIEVE and WAIT

Exit: Returns to CICS/VS

APLXGKU

Module: APLXGKU

Called By: APLXGCOM, APLXGCHC

Description: This is the mainline of the GDDX CICS/VS-only user transaction I/O support containing routines for the following: a) startup of CICS/VS GDDX task, b) synchronization of requests from session manager and AP126, c) GDDM path initialization, d) GDDM path termination, e) open a hardcopy destination (FSOPEN), f) passthrough request to GDDM under user transaction with proper synchronization, and g) I/O request to schedule a terminal transaction and wait for its completion.

Calls: ADMASP, KADEF via GBL. CICS/VS command level: FREEMAIN, RELEASE and SORT

Exit: Returns

APLXGS

Module: APLXGS

Called By: APLXGCOM, APLXGCHC

Description: This is the CMS-only support for APLXGDDM and contains routines that perform first-time initialization and hardcopy open register for APLXGDDM. It also provides the last-path CMS-only termination function.

Calls: ADMASP (GDDM entry point).

Exit: Returns

APLXGY

Module: APLXGY

Called By: APLXGCOM, APLXGCHC

Description: This is the entry point in module APLXGY. Its routines perform first path initialization and hardcopy open register for APLXGDDM, as required by system-dependent modules in all environments.

Calls: ADMASP (GDDM entry point).

Exit: Returns

APLXGYON

Module: APLXGY

Called By: APLYUINI

Description: This is the APL initialization entry from AP startup. It causes the AP task to gain control at routine GYCALL, which will invoke APLXGCOM when notified of a request and post the caller when the task is completed.

Calls: APLXGCOM (GDDM entry point). Macros APLXWAIT, APLXWPST

Exit: Signs off shared storage manager.

APLXGYRQ

Module: APLXGY

Called By: AP126, APL session manager (in TSO via VCTGDDX)

Description: This is the request processing entry point which receives control via the VCT when macro APLXG is issued. It causes a task switch to the APLXGYTA routine, waking up to return to caller when notified by APLXGYTA.

Calls: Macros APLXWAIT, APLXWPST

Exit: Returns

APLXMKSG

Module: APLXMKSG

Called By: Various executor routines.

Description: This is the main and only entry point to the storage management services module for CICS/VS which provides GETMAIN/FREEMAIN services to the caller (CICS/VS) through a system-independent interface.

Calls: Macros DFHSC, DFHSAADS

Exit: Returns

APLXMSSG

Module: APLXMSSG

Called By: Various executor routines.

Description: Provides GETMAIN/FREEMAIN services through a system-independent interface to the caller (CMS).

Calls: Macros DMSFREE, DMSFRET

Exit: Returns

APLXMYSG

Module: APLXMYSG

Called By: Available as a service routine

Description: This is the storage management services module for TSO which provides GETMAIN/FREEMAIN and associated services to the caller through a system-independent interface.

Calls: Macros GETMAIN, FREEMAIN

Exit: Returns

APLXPK

Module: APLXPK

Called By: Available for general use via PRTX label in VCT

Description: Main and only entry point to common executor print support in CICS/VS. It provides print requests OPEN, WRITE, and CLOSE, and transforms each request into an appropriate APLKEMGR call.

Calls: KEDEST via GBL

Exit: Returns

APLXPY

Module: APLXPY

Called By: APLXGDDM via APLCALLS

Description: This is the main entry point to the APL print module for TSO which satisfies the following TSO print requests: OPEN, WRITE, and CLOSE.

Calls: TSO QSAM file support, APL main storage services, LOAD/DELETE, and APL translation services. Macros OPEN, CLOSE, PUT, IHADCB, LOAD

Exit: Returns

APLXSTAK

Module: APLXSTAK

Called By: All stack protocol stack owners

Description: Create or destroy a stack.

Calls: Common main storage services.

Exit: Returns

APLXTRAN

Module: APLXTRAN

Called By: VS APL session manager, common AP services

Description: Provides various translation services.

Calls: Macros APLKZTOS, APLKSTOZ

Exit: Returns

APLXTREZ

Module: APLXTRAN

Called By: VS APL session manager, common AP services

Description: Translates a table from extended EBCDIC to ZCODE.

Calls: APLSCODE, APLKZTOS, APLKSTOZ

Exit: Returns

APLXTRZE

Module: APLXTRAN

Called By: VS APL session manager, common AP services

Description: Translates a table from ZCODE to extended EBCDIC.

Exit: Returns

APLXVERS

Module: APLXVERS

Called By: Any auxiliary processor. Common AP services.

Description: Provides various conversion services to convert one or more elements of a vector of values into another form.

Exit: Returns

APLXWKWP

Module: APLXWKWP

Called By: Attention, VS APL session manager separate task

Description: This is the main entry point to the VS APL CICS/VS wait/post services module which provides a system-independent interface for wait or post services to the CICS/VS user. Each request is transformed into an appropriate APLKEMGR call (CICS/VS).

Calls: APLKADSP (wait and post routines)

Exit: Returns

APLXWSWP

Module: APLSCSVI

Called By: Many executor routines.

Description: Provides system-independent interface for wait or post services to the CMS executor.

Calls: APLSHPST

Exit: Returns

APLXWYWP

Module: APLXWYWP

Called By: APLYUMSC, various executor routines and auxiliary processors.

Description: Provides system-independent interface for wait or post services to the TSO executor.

Calls: APLYUSVI (wait and post routines)

Exit: Returns to caller from post services; exits to dispatcher from wait.

APLXWYWP

Module: APLYUSVI

Called By: Various executor routines and auxiliary processors

Description: Provides wait and post services for system-independent task control. It includes a courtesy dispatch with a wait request of ECB pointer of zero (TSO).

Calls: APLSHPST. Macros APLPTRGT, APLTSOGL

Exit: Returns

APLYDAIR

Module: APLYDAIR

Called By: APLAM, APLXFYFL

Description: Allocates, frees, or deletes a data set, or checks its status.

Exit: Returns

APLYUCMD

Module: APLYUCMD

Called By: APLYU100

Description: Initializes all control blocks, calls the command scan, and builds the command name. The command module is now attached; it is passed a CPPL constructed by copying the CPPL passed to VS APL, but substituting the address of the built CBUF. The CMSECB is used as the ECB in the ATTACH because it is posted by the STAX exit. The TSOCMDAT bit is set to distinguish APLYU100 waiting from waiting caused by DELAY or MSG. When posted, the command subtask has either terminated normally or has been rendered nondispatchable by STAX. DAIR is now called with a request code of '2C' to mark the command subtask; the subtask can subsequently be detached. The line delete and character delete functions are resuppressed, and the QUAD-PW value is reestablished before returning to APLYU100.

For a full explanation of TSO command linkage and Terminal Monitor Program service routines, see Guide to Writing a Terminal Monitor Program and Command Processor.

Calls: Macros ATTACH, BLDL, LINK, STCC, STSIZE, WAIT, GETMAIN, FREEMAIN

Exit: Returns

APLYUCNV

Module: APLYUCNV

Called By: Various executor routines

Description: Imports into a VS APL/TSO sequential data set a VS APL workspace from a file created by one of the VS APL conversion programs (TSO).

Exit: Returns

APLYUEXC

Module: APLYUEXC

Called By: APLYUCMD

Description: Routine used to execute CLISTs.

Exit: Returns

APLYUFXI

Module: APLYUFXI

Called By: APLYUINI (VS APL initialization)

Description: Receives control after initialization and reacts to the success or failure of initialization (TSO).

Calls: Macros APLDEFN, YYCODE (local), ESTAE, APLEDIT

Exit: YYEXIT in APLFXIIM; EXREQUES(Error)

APLYUHSH

Module: APLYUHSH

Called By: APLYULIB

Description: This is the hasher/unhasher module which examines a lock and its key to determine if the workspace was saved by VS APL/TSO, and, if so, what the TSO owner userid is.

Exit: Returns

APLYULNE

Module: APLYULNE

Called By: Invocations produced by the 'APLEDIT' macros

Description: This is the interface module to the LINEDIT macro in the TSO environment. At entry, the PLIST code is decoded and expanded inside the work area so that it will be possible to easily access all its fields. The message header is then constructed and the message text is scanned, byte by byte. Whenever an ellipsis is found in the message text, an argument is taken from the 'SUBS' parameter list, the appropriate conversion is performed, and the result is substituted for the ellipsis. The resulting message is then copied into the specified buffer, the 'DISP' field is examined, and the appropriate action is taken.

Calls: APLYUTIO

Exit: Returns

APLYURVC

Module: APLYURVC

Called By: Auxiliary processor or module APLYUSHV using 'ASVP....' macro

Description: Links to shared storage manager (also called shared variable processor) from an auxiliary processor or the TSO executor on a shared variable service request.

Exit: Branches to entry point ASVPSERV in module APLYUSVI

APLYUTBL

Module: APLYUTBL

Called By: None (data only)

Description: This contains all translate tables for terminals.

Exit: None

APLYUTIO

Module: APLYUTIO

Called By: APLYUTYP

Description: This is the TSO nondisplay terminal interface which simulates CMS SVC 202 terminal input/output functions (TSO).

Calls: SCOTRT. Macros TPUT, TGET, APLDEFN

Exit: Returns

APLYUUSR

Module: APLYUUSR

Called By: APLYUINI

Description: This constitutes a sample installation-written initialization exit routine. It 1) allows any user with operator authority to save into or drop from public workspaces, and 2) scans for the ownership operand, and, if provided, forces the specification of a password.

Exit: Returns at +0 (Error—user is not authorized to continue); +4 (user is authorized to proceed with APL session).

APL100

Module: APLYU100

Called By: Control passed directly from shared variable processor

Description: Executes a TSO command.

Calls: APLYUSCN. Macros APLWSM, ASVPSON, ASVPQRY, ASVPOFR, ASVPREF, ASVPWAIT, ASVPRET, ASVPSPC, ASUSCV, APLPCV, APLSHSVP, ASVPSOF, APLEDIT, ABEND

Exit: TSO ABEND (Error)

APL100

Module: APL100

Called By: ASVPSERV; via Post on ECB

Description: Auxiliary processor AP100; executes CMS and CP commands while obeying the search rules for IMPEX and IMAP.

Calls: ASVPSRVC. Macros APLWSM, ASVPSON, ASVPQRY, ASVPOFR, ASVPREF, ASVPWAIT, ASVPRET, ASVPSPC, ASUSCV, APLPCV, APLSHSVP, ASVPSOF, LINEDIT, ABEND, NUCON, TSOBLKS, DMSFREE, DMSFRET, APLXBXIT

Exit: ASVPSRVC with wait request; CMS ABEND (Error)

APL100K

Module: APL100K

Called By: Entry point KMACRO

Description: Part of the CICS/VS command auxiliary processor. Issues CICS/VS commands and starts CICS/VS transactions.

Calls: Entry point APL100KO. Macros APLKOFR, APLKREF, APLKSPC, APLKWAIT, APLKEXIT, APLKRET, APLKACHK, APLKG (LIBSERV). CICS/VS macros DFHIC (PUT, INITIATE), DFHKC (ATTACH), DFHSC (GETMAIN, FREEMAIN), DFHSP

Exit: Returns

APL100KO

Module: APL100KO

Called By: Entry point APL100K

Description: Part of the CICS/VS command auxiliary processor. Connects CICS/VS transactions to the user terminal.

Calls: Any CICS/VS transaction. CICS/VS macros DFHPC (LOCATE, LINK, RETURN), DFHSC (GETMAIN)

Exit: DFHPC (RETURN)

APL101

Module: APLYU101

Called By: Shared variable processor APLYUSVI

Description: This is TSO's auxiliary processor AP101, whose function is to stack an APL input line.

Calls: APLYUSCN. Macros APLWSM, ASVPSON, ASVPQRY, ASVPOFR, ASVPREF, ASVPWAIT, ASVPRET, ASVPSPC, APLSCV, APLPCV, APLSHSVP, ASVPSOF, APLCCVO, APLEDIT, ABEND

Exit: Signs off to the TSO SSM

APL101

Module: APL101

Called By: ASVPSERV; via Post on ECB

Description: Auxiliary processor AP101; stacks lines to be used at next request for terminal input. In VM/SP systems, an attempt to stack 'HT' or 'RT' will result in the SET CMSTYPE commandbeing issued.

Calls: ASVPSRVC

Exit: ASVPSRVC with wait request; CMS ABEND (Error)

APL102

Module: APLYU102

Called By: Shared variable processor

Description: This is the TSO main storage access auxiliary processor. It displays storage for the user.

Calls: Macros APLIBITS, APLCMSGL, APLWSM, ASVPWAIT, ASVPSOF, ASVPSON, ASVPQRY, ASVPREF, ASVPSVP, ASVPRET, ASVPSPEC, ASVPSOFR, APLSHSUP, APLFSMP, APLFSMW, APLDFNUC, APLSYSTP, APLGLPTR, FREEMAIN, GETMAIN, SAVE, RETURN, IHAPSA, CVT, IKJTCB

Exit: Returns

APL102K

Module: APL102K

Called By: Entry point KMACRO

Description: The CICS/VS main storage access auxiliary processor. Displays storage for the user.

Calls: Macros APLKOFR, APLKREF, APLKWAIT, APLKSPC, APLKRET, APLKEXIT, APLKACHK

Exit: Returns

APL110

Module: APL110

Called By: ASVPSERV via Post on ECB

Description: Auxiliary processor AP110; reads and writes CMS disk files.

Calls: ASVPSRVC

Exit: ASVPSRVC with wait request; CMS ABEND (Error)

APL111

Module: APLYU111

Called By: Shared variable processor APLSCSVI

Description: This is the TSO auxiliary processor AP111 which reads and writes QSAM files.

Calls: APLYUSCN. Macros APLCCVI, APLCCVO, APLIREGS, APLWSM, APLZCODE, APLPCV, APLSCV, ASUSCV, APLSHSVP, ASVPOFR, ASVPQRY, ASVPREF, ASVPRET, ASVPSOF, ASVPSON, ASVPSPC, ASVPWAIT, ABEND, CLOSE, DCB, DCBD, FREEPOOL, GET, GETMAIN, APLEDIT, OPEN, PUT, FREEMAIN, ONABEND (LOCAL)

Exit: Signs off to the TSO SSM

APL111

Module: APL111

Called By: ASVPSERV via Post on ECB

Description: Auxiliary processor AP111; reads and writes files using CMS simulation of OS QSAM.

Calls: ASVPSRVC

Exit: ASVPSRVC with wait request; CMS ABEND (Error)

APL120

Module: APL120

Called By: Initialization (CMS and TSO), shared storage manager (CICS/VS)

Description: Communicates between the VS APL session manager commands and auxiliary processors.

Calls: APLASCHD, APLXAC

Exit: Returns

APL121

Module: APL121

Called By: CMS/TSO initialization

Description: This is the main entry point to the VS APL data file which creates, writes, updates, reads, and/or deletes VS APL object files.

Calls: APLXFYFL, APLXFSFL, APLXAC, APLXDUMP, APLXMSSG, APLXMYSG, and APLXSTAK. Macros APLXASO, APLXMAIN, APLXCAPS, APLCALLS

Exit: Returns

APL121K

Module: APL121K

Called By: Entry point KMACRO

Description: The CICS/VS APL format auxiliary processor. Creates, writes, updates, reads, and/or deletes APL object files.

Calls: Entry point APLKLIBF. Macros APLKOFR, APLKRET, APLKREF, APLKSPC, APLKEXIT, APLKWAIT, APLKACHK. CICS/VS macro DFHSC (GETMAIN, FREEMAIN)

Exit: Returns

APL123

Module: APL123

Called By: Control directly passed from shared variable processor

Description: This is the TSO/CMS auxiliary processor 123 which reads and/or writes VSAM files.

Calls: APLXMSSG, APLXMYSG. Macros APLCCVI, APLCCVO, APLSHSVP, ASVPACC, ASVPOFR, ASVPQRY, ASVPREF, ASVPRET, ASVPSOF, ASVPSON, ASVPSPC, ASPWAIT, ABEND, CLOSE, GET, PUT, OPEN, POINT, ERASE, MODCB, GENCB, TESTCB, SHOWCB, APLXMAIN, APLEDIT, APLXMAIN

Exit: Returns

APL123K

Module: APL123K

Called By: Entry point KMACRO

Description: The CICS/VS VSAM/ISAM file auxiliary processor. Reads from and writes to VSAM and ISAM data sets.

Calls: Macros APLKOFR, APLKRET, APLKSPC, APLKREF, APLKWAIT, APLKEXIT, APLKACHK. CICS/VS macros DFHSC (GETMAIN, FREEMAIN), DFHFC (GET, PUT, DELETE, GETAREA, RELEASE, SETL, GETNEXT, RESETL, ESETL)

Exit: Returns

APL124K

Module: APL124K

Called By: Entry point KMACRO

Description: The CICS/VS full screen manager auxiliary processor. Uses terminal manager routines, which are a part of the CICS/VS executor to handle all valid user requests

Calls: Macros APLKOFR, APLKRET, APLKREF, APLKSPC, APLKWAIT, APLKEXIT, APLKTERM (INIT, FORMAT, WRITE, READ, GETDATA, SETCUR, FLDATTR, GETFORM, HCOPY, ALARM, FINAL). CICS/VS macros DFHSC (GETMAIN, FREEMAIN)

Exit: Returns

APL125K

Module: APL125K

Called By: Entry point KMACRO

Description: The CICS/VS DL/I access auxiliary processor. Provides a DL/I interface for the CICS/VS user.

Calls: Macros APLKOFR, APLKRET, APLKSPC, APLKREF, APLKEXIT, APLKWAIT, APLKACHK, CALLDLI. CICS/VS macro DFHSC (GETMAIN, FREEMAIN)

APL126

Module: APL126

Called By: Initialization (CMS and TSO), shared storage manager (CICS/VS)

Description: This is the main entry point to the GDDM auxiliary processor which processes requests from a user (CMS, TSO, or CICS/VS) to be passed on to GDDX, and allows the user to 1) control the screen format of his terminal, 2) write to and read from the formatted screen, 3) erase screen fields, 4) copy screen images to a printer, 5) condition screen fields for light pen usage, and 6) read program function and attention keys. It also allows a user to specify a request (to AP126) that is not a GDDM call, but controls the AP options.

Calls: GDDM interface services (APLXGDDM), common AP SERVICES (APLXCAPS), conversion services (APLXVERS), stack management services, storage management services, abend services and dump services. Macros APLXAEAT, APLG, APLXMAIN, APLXASO, APLXBXIT, APLXCAPS

Exit: In CMS/TSO, stays active until the shared variable processor terminates. In CICS/VS, terminates when user signs off.

APL126T

Module: APL126T

Called By: GDDMRCTL

Description: This is the main entry name of the GDDM auxiliary processor table module which expands the macro APL126TB, once for each AP 126 GDDM request, to define entries in a GDDM request table set.

Calls: Macro APL126TB

APL132K

Module: APL132K

Called By: Entry point KMACRO

Description: The CICS/VS transient data auxiliary processor. Accesses CICS/VS transient data including both intrapartition queues and sequential devices.

Calls: Entry point APLKEMGR. Macros APLKOFR, APLKRET, APLKREF, APLKSPEC, APLKWAIT, APLKEXIT. CICS/VS macros DFHSC (GETMAIN, FREEMAIN)

Exit: Returns

APL139K

Module: APL139K

Called By: Entry point KMACRO

Description: The CICS/VS alternate input processor. Passes user-supplied data from the shared storage manager to the session manager.

Calls: Macros APLKOFR, APLKRET, APLKREF, APLKWAIT

Exit: Returns

APL210

Module: APLYU210

Called By: Shared variable processor APLYUSVI

Description: This is the BDAM auxiliary processor for TSO which reads and writes BDAM files.

Calls: APLYUSCN. Macros APLCCVI, APLCCVO, APLIREGS, APLWSM, APLZCODE, APLPCV, APLSCV, APLSHSVP, ASVPOFR, ASVPQRY, ASVPREF, ASVPRET, ASVPSOF, ASVPSON, ASVPSPC, ASVPWAIT, ABEND, CLOSE, DCB, DCBD, FREEPOOL, GET, GETMAIN, APLEDIT, OPEN, PUT, FREEMAIN

Exit: Signs off to the TSO SSM

APOPEN

Module: APLPAPCD

Called By: APLPAPPR

Description: Executes service request to internal auxiliary processors AP121 and AP122 to open a VSPC file for input, output, or update.

Calls: APDFN, ERMSGRTN

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

APPASSWD

Module: APLPAPCD

Called By: APLPAPPR

Description: Executes service request to internal auxiliary processors AP121 and AP122 to change the password of a VSPC file.

Calls: APDFN, ERMSGRTN

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

APSHARE

Module: APLPAPCD

Called By: APLPAPPR

Description: Executes service request to internal auxiliary processors AP121 and AP122 to change the share status of a VSPC file.

Calls: APDFN, ERMSGRTN

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

APVIO

Module: APLPAPCD

Called By: APLPAPPR

Description: Executes all service requests to internal auxiliary processor AP123.

Calls: APDFN, ERMSGRTN

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

ASVPSERV

Module: APLSCSVI

Called By: ASVPSRVC

Description: Determines type of shared variable request and calls routine to handle it. On return, schedules the next auxiliary processor that is ready to run; if none, returns to the interpreter at the instruction following its last shared variable service request.

Calls: APLSHACC, APLSHCPY, APLSHOFR, APLSHQUE, APLSHREF, APLSHRET, APLSHSOF, APLSHSON, APLSHACC, Auxiliary Processors

Exit: See description

ASVPSERV

Module: APLYUSVI

Called By: Various executor routines and auxiliary processors

Description: Determines the type of request and invokes the proper shared variable processor routine (TSO).

Calls: APLSHACC, APLSHCPY, APLSHOFR, APLSHQUE, APLSHREF, APLSHRET, APLSHSOF, APLSHSON, APLSHSPC. Macro APLSHPAR.

Exit: Returns

ASVPSRVC

Module: APLYURVC (TSO), ASVPSRVC (CMS)

Called By: User-written auxiliary processors or dynamically-loaded auxiliary processors

Description: Entry point to shared storage manager for VS APL.

Calls: (for TSO) APLYUSVI; Macros APLDEFN, APLPTRGT. (for CMS) APLSCSVI; Macros NUCON, APLPATCH.

Exit: ASVPSERV

BEXIT

Module: APLKASTB

Called By: Various executor routines.

Description: This is the main entry point to the VS APL CICS/VS abend services module which provides a system-independent interface for abend services to the CICS/VS executor and auxiliary processors (CICS/VS).

Calls: APLKADSP. Macro APLKEDIT

Exit: Returns

COIBM

Module: APLCOIBM

Called By: CMS

Description: Copyright notice and entry point from CMS to VS APL.

Exit: APL

CVCULL

Module: APLCCULL, APLOCULL

Called By: CVINIT

Description: Calls workspaces for selective conversion; gives CMS fileid to workspace for selected workspace; resolves filename conflicts. Rejects invalidly named workspaces which cannot be resolved.

Calls: CVRPRT

Exit: Returns

CVDATE

Module: APLCMISC, APLOMISC (only for OS/VS), APLQMISC

Called By: CVINIT

Description: Gets date from system.

Exit: Returns

CVDIRE

Module: APLCMISC, APLODIRE

Called By: CVINIT

Description: Builds shortened form of directory; dummy routine under CMS.

Calls: CVSLST

Exit: Returns

CVDISP

Module: APLCDISP, APLODISP, APLQDISP

Called By: CVFUNC

Description: Converts APL/360 codestring to VS APL copy transmission codes; for content conversion, converts or flags APL/360 idioms to VS APL equivalents.

Calls: CVTBCD

Exit: Returns

CVFUNC

Module: APLCFUNC, APLOFUNC, APLQFUNC

Called By: CVWKSP

Description: Converts format for all functions; converts content or replaces function.

Calls: CVDISP, CVRPRT, CVWSFN, CVSHIP, ITLINEO, ITOKENIZ, ITCLOSET

Exit: Returns

CVGDIR

Module: APLODIRE

Called By: CVSAVE

Description: Looks for PERLIB in shortened form of directory.

Exit: Returns

CVGRUP

Module: APLCGRUP, APLOGRUP, APLQGRUP

Called By: CVWKSP

Description: Enters an XM6 group name and its members' names into VS APL workspace.

Calls: ITSTSRCH, IESFIND

Exit: Returns

CVIBNM

Module: APLCIBNM, APLOIBNM, APLQIBNM

Called By: CVWKSP

Description: Generates unique three-character alphabetic underscored name for IBEAM simulator function.

Exit: Returns

CVINIT

Module: APLCINIT, APLOINIT, APLQINIT

Called By: Host operating system

Description: Sole entry and exit point for conversion program. Sets up and initializes conversion parameters and flags; establishes buffers and storage spaces for APL/360 workspace and directory (input) and VS APL workspace (output); reads workspace and directory from tape.

Calls: CVPARM, CVDATE, CVSPIE, CVCULL, CVWKSP, CVDIRE, CVRPRT, CVTBCD, CVIOER

Exit: Returns

CVIOER

Module: APLCMISC, APLOMISC

Called By: CVINIT

Description: Prints permanent input/output error messages.

Calls: CVPRTR

Exit: Returns

CVLEAR

Module: APLCLEAR, APLOLEAR, APLQLEAR

Called By: CVWKSP

Description: Initializes VS APL workspace.

Calls: CVTBCD, CVRPRT

Exit: Returns

CVPARM

Module: APLCPARM, APLOPARM, APLQPARM

Called By: CVINIT

Description: Sets conversion flags according to parameters; for selective conversion, builds selection list in SELIST.

Calls: CVPRTR

Exit: Returns

CVPRTR

Module: APLCMISC, APLOMISC, APLQMISC

Called By: CVRPRT, CVIOER, CVSPIE, CVPARM

Description: Prints conversion information on SYSPRINT (SYSLST).

Exit: Returns

CVRPRT

Module: APLCRPRT, APLORPRT, APLQRPRT

Called By: CVVARB, CVFUNC, CVWKSP, CVLEAR, CVINIT, CVSAVE, CVCULL

Description: Prints a detail line of conversion report; takes care of pagination.

Calls: CVPRTR

Exit: Returns

CVSAVE

Module: APLCSAVE, APLOSAVE, APLQSAVE

Called By: CVWKSP

Description: Saves converted VS APL workspace as a CMS file whose name is provided by APLCCULL routine. Saves as control intervals on APLOUT for VSPC.

Calls: CVRPRT, CVGDIR

Exit: Returns

CVSHIP

Module: APLCSHIP, APLOSHIP

Called By: CVWKSP, CVFUNC

Description: Tokenizes a multiline VS APL function into VS APL workspace.

Calls: ITLINEO, ITOKENIZ, ITCLOSET

Exit: Returns

CVSLST

Module: APLOSLST

Called By: CVDIRE

Description: Looks for given library number and workspace name in selective conversion list.

Exit: Returns

CVSPIE

Module: APLCSPIE, APLOSPIE, APLQSPIE

Called By: CVINIT, Host operating system

Description: Sets SPIE exit when called by CVINIT; when exit taken, prints error message, time stamp, PSW, and registers.

Calls: CVTBCD, CVPRTR

Exit: Returns; CVINIT (Recoverable Error); ABEND (Error)

CVTBCD

Module: APLCTBCD, APLOTBCD

Called By: CVLEAR, CVDISP, CVINIT, CVLEAR

Description: Determines internal type of data element; converts to Z-code representation to given format and data type.

Exit: Returns

CVTIDY

Module: APLCMISC, APLOTIDY, APLQMISC

Called By: CVWKSP

Description: Collects discarded material from VS APL workspace.

Exit: Returns

CVVARB

Module: APLCVARB, APLOVARB, APLQVARB

Called By: CVWKSP

Description: Enters APL/360 variables in VS APL workspace; for character, translates to VS APL Z-codes; for Boolean, reverses bits in every byte.

Calls: CVRPRT, IESFIND, ITSTSRCH

Exit: Returns

CVWKSP

Module: APLCWKSP, APLOWKSP, APLQWKSP

Called By: CVINIT

Description: Finds global objects in source workspace; calls appropriate routine to convert objects for VS APL workspace.

Calls: CVIBNM, CVLEAR, CVSHIP, CVRPRT, CVSAVE, CVGRUP, CVVARB, CVFUNC, CVTIDY

Exit: Returns

CVWSFN

Module: APLCWSFN, APLOWSFN

Called By: CVFUNC

Description: Replaces APL/360 WSFN with VS APL equivalent in VS APL Z-codes (copy transmission format).

Exit: Returns

DMSSCND

Module: APLYUSCN

Called By: Various TSO executor modules

Description: This is the entry point of the old parameter list format. It transforms an input command line into a series of 8-byte parameters.

Exit: Returns

DMSSCNN

Module: APLYUSCN

Called By: Various TSO executor modules

Description: This is the entry point of the new parameter list format. It transforms an input command line from a string of arguments into a series of 8-byte parameters.

Exit: Returns

ERENDEX

Module: APLPCOEX

Called By: VSPC service request and internal auxiliary processor routines

Description: Writes error messages to terminal and VSPC online log; ends

Calls: ERTIMDAT

Exit: Returns to VSPC (Error)

ERMSGRTN

Module: APLPSERR

Called By: All VSPC service request handling routines

Description: Writes error message to VSPC online log.

Exit: Returns

ERSAVEAR

Module: APLPCOEX

Called By: All VSPC service request routines

Description: Writes error messages to terminal and VSPC online log and ends execution, when save area block is full.

Calls: ERTIMDAT

Exit: Returns to VSPC (Error)

ERTIMDAT

Module: APLPSERR

Called By: PCSYSER, APLPCENT, ERSAVEAR, ERENDEX

Description: Places time and date in VSPC executor work area.

Exit: Returns

FREESTOR

Module: APLPAPGD

Called By: GDDMCRET, GDDMRCTL, GDDMSCTL, GDDMSDAT

Description: Frees storage blocks allocated for buffers by the VSPC version of AP 126.

Calls: Macros APLPENTR, ASUSRQ, APLPAPER, and APLPEXIT

Exit: Returns; ERSAVEAR (Error)

FSMBUZZ

Module: APLPAPFS

Called By: APLPAPPR

Description: For FSM internal auxiliary processor (VSPC), notes user request to sound the audible alarm at the display terminal at the next display screen read or write request.

Exit: Returns

FSMFORMT

Module: APLPAPFS

Called By: APLPAPPR

Description: Validity checks user's FSM field definitions and builds FSMFLD entries in FSM auxiliary processor work area for FSM internal auxiliary processor (VSPC).

Calls: FSMSUB1, ERMSGRTN, FSMSUB3

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

FSMGET

Module: APLPAPFS

Called By: APLPAPPR

Description: For FSM internal auxiliary processor (VSPC), processes user request for data read from display screen.

Calls: ERMSGRTN, FSMSUB1, FSMSUB3

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

FSMHCOPY

Module: APLPAPFS

Called By: APLPAPPR

Description: For FSM internal auxiliary processor (VSPC), processes user request to make a hard copy of the current display screen.

Calls: FSMSUB1, ERMSGRTN

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

FSMMINT

Module: APLPAPFS

Called By: APLPAPPR

Description: For FSM internal auxiliary processor (VSPC), notes user request to modify display intensity of defined display screen fields.

Calls: FSMSUB3

Exit: Returns

FSMMTYPE

Module: APLPAPFS

Called By: APLPAPPR

Description: For FSM internal auxiliary processor (VSPC), notes user request to modify type of defined display screen fields.

Calls: FSMSUB3

Exits: Returns

FSMREAD

Module: APLPAPFS

Called By: APLPAPPR

Description: For FSM internal auxiliary processor (VSPC), formats display screen if necessary, reads from display screen, and returns description of user's input.

Calls: FSMSUB1, FSMSUB2, ERMSGRTN

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

FSMRFORM

Module: APLPAPFS

Called By: APLPAPPR

Description: For FSM internal auxiliary processor (VSPC), processes user request for the format of the currently defined FSM fields.

Exit: Returns

FSMSETC

Module: APLPAPFS

Called By: APLPAPPR

Description: For FSM internal auxiliary processor (VSPC), notes user request to set cursor at a given location on subsequent display screen write requests.

Calls: FSMSUB3

Exit: Returns

FSMSUB1

Module: APLPAPFS

Called By: FSMFORMT, FSMREAD, FSMGET, FSMHCOPY

Description: Allocates additional storage from user's VSPC workspace quota for FSM internal auxiliary processor (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

FSMSUB2

Module: APLPAPFS

Called By: FSMWRITE, FSMREAD

Description: Builds VSPC display screen service request to define display screen fields and to write data to display screen.

Calls: FSMSUB1, ERMSGRTN

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

FSMSUB3

Module: APLPAPFS

Called By: FSMFORMT, FSMWRITE, FSMGET, FSMMTYPE, FSMMINT, FSMSETC, APLPAPPR, GDDMRCTL

Description: Converts floating point to integer, and flags negative values.

Exit: Returns

FSMWRITE

Module: APLPAPFS

Called By: APLPAPPR

Description: For FSM internal auxiliary processor (VSPC), formats display screen if necessary and writes to display screen.

Calls: FSMSUB2, ERMSGRTN, FSMSUB3

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

GDDMCRET

Module: APLPAPGB

Called By: APLPAPRT

Description: VSPC executor routine used to perform cleanup when an AP 126 CTL variable is retracted.

Calls: FREESTOR

Exit: Returns; ERSAVEAR (Error)

GDDMRCTL

Module: APLPAPGC

Called By: APLPAPPR

Description: Main entry point to the GDDM auxiliary processor for VSPC. User requests are interpreted, processed, and passed to GDDM. For more information, see description of entry point APL126, which has similar logic.

Calls: APLP126T. FREESTOR, FSMSUB3, GDDXINIT, GETSTOR Macro APLPAPSR

Exit: Returns; ERSAVEAR (Error)

GDDMSCTL

Module: APLPAPGB

Called By: APLPAPPR

Description: Entry point used to specify the control variables for the previous AP 126 request by moving it to the user's workspace.

Calls: FREESTOR

Exit: Returns; ERSAVEAR (Error)

GDDMSDAT

Module: APLPAPGB

Called By: APLPAPPR

Description: Entry point for VSPC AP 126 to specify DAT variable by moving character data to the user's workspace.

Calls: FREESTOR

Exit: Returns; ERSAVEAR (Error)

GDDMSOFF

Module: APLPAPGB

Called By: APLPAPSF, APLPAPRT

Description: Terminates GDDM after last path is retracted or during SSM sign-off.

Calls: ERMSGRTN. Macro ASUSRQ

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

GDDX

Module: APLPAPGD

Called By: GDDMRCTL

Description: Issues the GDDM request for GDDM and GDDX operations in a VSPC environment.

Calls: ERMSGRTN. Macro ASUSRQ

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

GDDXINIT

Module: APLPAPGD

Called By: GDDMRCTL

Description: Initializes the GDDX path for GDDM and GDDX operations in a VSPC environment.

Calls: ERMSGRTN. Macros ASUSRQ, APLPAPER, and APLEXIT

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

GETSTOR

Module: APLPAPGD

Called By: GDDMRCTL

Description: Allocates storage blocks required for buffers by the VSPC version of AP 126.

Calls: ERMSGRTN. Macro ASUSRQ

Exit: Returns; ERSAVEAR (Error), ERENDEX (Error)

IABNM

Module: APLIATRN

Called By: IEDYB, IACAL370, IAIPROD, IAREDU

Description: Calculates generalized combinations using floating-point arguments.

Calls: IAFACT

Exit: Returns; IEABEND (Error)

IACAL370

Module: APLIEXFR

Called By: Microcode

Description: Provides a table of one-word branches; each corresponds to one service; passes control to routines to process service or call appropriate appendage routine.

Calls: IATIDY, IAFLCL, IAFACTRL, IAROLL, IAIROLL, IAPOW, IALOG, IACIRCLE, IARESIDU, IABNM, IADEAL, IAQUADS, IAQUADSA, IAQDSPEC, IASHRPST, IASHADO, IAENCODE, IADECODE, IAGRADE, IATKDP, IAREDU, IASCAN, IAIPROD, IADYB, IAMDOM, IADDOM, IAMFORM, IADFORM, IAMSHARE, IADSHARE, IAEXECTE, IAROTA, IAMTRAN, IADTRAN, IACOMMA, IACMX, IESTOSTK, IELDSTK, IEINDB

Exit: Returns; IEABEND (Error)

IACHK

Module: APLIACHK

Called By: IASCOPY

Description: Verifies that data passed to the interpreter via the shared storage manager is correct.

Exit: Returns

IACIRCLE

Module: APLIACIR

Called By: IEDYB, IAIPROD, IAREDU, IACAL370

Description: Computes trigonometric functions (dyadic circle).

Calls: IALOGR, IAEXPR, IASQRT

Exit: Returns; IEABEND (Error)

IACMX

Module: APLIECMX

Called By: IACAL370

Description: Provides access to compress and expand routines for microcode.

Calls: IECMEX

Exit: Returns; IEABEND (Error)

IACOMMA

Module: APLIERHO

Called By: IACAL370

Description: Provides access to laminate and catenate routines for microcode.

Calls: IECOMMA

Exit: Returns; IEABEND (Error)

IADDOM

Module: APLIADOM

Called By: IEDYAD, IACAL370

Description: Performs matrix division.

Calls: IESFIND, IESGINIT, IESGETN, IASQRT

Exit: Returns

IADEAL

Module: APLIATRN

Called By: IEDYB, IACAL370

Description: Calculates a dyadic random value.

Calls: IESFIND, IESGINIT

Exit: Returns; IEABEND (Error)

IADECODE

Module: APLIADEC

Called By: IEDYAD, IACAL370

Description: Performs decode operation.

Calls: IESFIND, IESGINIT, IESGETN, IESFREE, IESGETV, IAPLFUN

Exit: Returns; IEABEND (Error)

IADFORM

Module: APLIAFOR

Called By: IEDYAD, IACAL370

Description: Performs dyadic format operation.

Calls: IAGFMT2, IATOBCD2, IESFIND, IESFREE, IESGETN, IESGINIT

Exit: Returns; IEABEND (Error)

IADSHARE

Module: APLIATRN

Called By: IACAL370, IEDYAD

Description: Processes dyadic system functions; if the function deals with shared variables, the appropriate routine in APLIASHF is called, otherwise the pertinent routine in APLIAQFN is called.

Calls: IAQSVO, IAQSVQ, IAQSVC, IAQNL

Exit: Returns; IEABEND (Error)

IADTRAN

Module: APLIATSP

Called By: IEDYAD, IACAL370

Description: Performs dyadic transpose operation.

Calls: IESFIND, IESGINIT, IESGETN, IESFREE

Exit: Returns; IEABEND

IADYB

Module: APLIEFCH

Called By: IACAL370

Description: Provides access to outer product routine for microcode.

Calls: IEDYB

Exit: Returns; IEABEND (Error)

IAENCODE

Module: APLIAENC

Called By: IEDYAD, IACAL370

Description: Performs encode operation.

Calls: IESFIND, IESGINIT, IAPLFUN, IESGETN, IESGETV, IARESIDU

Exit: Returns; IEABEND (Error)

IAEXECTE

Module: APLIATRN

Called By: IACAL370, IEMONAD

Description: Executes the execute primitive operation.

Calls: ITEMPFUN, IATIDY, IESFREE

Exit: Returns; IEABEND (Error)

IAEXNAME

Module: APLIATRN

Called By: IENAME

Description: Extends the address table in the operation stack.

Exit: Returns

IAEXPR

Module: APLIATRS

Called By: IACIRCLE, IAPOW

Description: Calculates the value of E raised to the specified power.

Exit: Returns; IEABEND (Error)

IAEXSTCK

Module: APLIATRN

Called By: IESCANG, IEFUNN, IESTOSTK, IEXARCH, ITEXECUT

Description: Extends the operation stack.

Exit: Returns

IAFACT

Module: APLIATRS

Called By: IABNM, IAFACTRL

Description: Computes the factorial of the indicated argument.

Exit: Returns; IEABEND (Error)

IAFACTRL

Module: APLIATRN

Called By: IEMONAD, IACAL370

Description: Calculates the factorial of a floating-point argument.

Calls: IAFACT

Exit: Returns

IAFCHNAM

Module: APLIANAM

Called By: IAQCR, IAQNC, IAQEX, IAQSVR, IAQSVC, IAQSVO

Description: Returns the internal name that is indicated by the specified row of the character item identified in WSMRGETV.

Calls: IATIDY, ITSTSRCH

Exit: Returns

IAFLCL

Module: APLIATRN

Called By: IEMONAD, IACAL370

Description: Calculates the value of the floor or ceiling of a floating-point argument.

Exit: Returns; IEABEND (Error)

IAGFMT

Module: APLIAGFM

Called By: IAGOUT, IAMFORM

Description: Determines data type of a given variable and returns the output format field width according to the type.

Calls: IALOGR, IATOBCD

Exit: Returns

IAGFMT2

Module: APLIAGFM

Called By: IADFORM

Description: Determines data type of a given variable and scans every nth element (n is user specified) for its sign and the maximum magnitude information according to the data type.

Calls: IALOGR, IATOBCD

Exit: Returns

IAGOUT

Module: APLIAGOU

Called By: IAQDSPEC, ITEXECUT

Description: Converts elements of a variable to Z-code representation in WSMBUFF for terminal output; issues YYTYO service request.

Calls: IAGFMT, IATOBCD, APLFXIIM

Exit: Returns; ITSYSERR (Error)

IAGRADE

Module: APLIAGRD

Called By: IEMONAD, IACAL370

Description: Performs grade-up and grade-down operations.

Calls: IESFIND, IESGINIT, IESGETN, IESFREE

Exit: Returns; IEABEND (Error)

IAHTSPEC

Module: APLIASYV

Called By: IASYSPEC, IAUNSHAD, IASYSPST

Description: Validates specified tab settings; sends valid settings or null settings to executor.

Calls: IESGINIT, IESGETN, APLFXIIM, IARTOI

Exit: Returns

IAIPROD

Module: APLIAPRD

Called By: IACAL370, IEDYAD

Description: Performs the inner product (matrix product) operation with the following combinations of arguments: vector/array, array/vector, array/array.

Calls: IESFIND, IESFREE, IARESIDU, IABNM, IAPOW, IALOG, IACIRCLE

Exit: Returns; IEABEND (Error)

IAIROLL

Module: APLIATRN

Called By: IEMONAD, IACAL370, IAROLL

Description: Calculates a monadic random from an integer argument.

Exit: Returns; IEABEND (Error)

IALOG

Module: APLIATRN

Called By: IEDYB, IACAL370, IAIPROD, IAREDU

Description: Calculates the logarithm of a floating-point argument.

Calls: IALOGR

Exit: Returns

IALOGR

Module: APLIATRS

Called By: IACIRCLE, IAPOW, IALOG, IAGFMT, IAGFMT2

Description: Calculates the value of the natural logarithm of the argument.

Exit: Returns; IEABEND (Error)

IAMDOM

Module: APLIADOM

Called By: IEMONAD, IACAL370

Description: Performs matrix inversion operation.

Calls: IESFIND, IESGETN, IESGINIT, IASQRT

Exit: Returns

IAMFORM

Module: APLIAFOR

Called By: IEMONAD, IACAL370

Description: Performs monadic format operation.

Calls: IAGFMT, IATOBCD, IESFIND

Exit: Returns; IEABEND (Error)

IAMSHARE

Module: APLIATRN

Called By: IACAL370, IEMONAD

Description: Processes monadic system functions; if the function deals with shared variables, the appropriate routine in APLIASHF is called, otherwise the appropriate routine in APLIAQFN is called.

Calls: IAQSVO, IAQSVC, IAQSVR, IAQFX, IAQSVQ, IAQCR, IAQEX, IAQDL, IAQNL, IAQNC

Exit: Returns; IEABEND (Error)

IAMTRAN

Module: APLIATSP

Called By: IEMONAD, IACAL370

Description: Performs monadic transpose operation.

Calls: IESFIND, IESGINIT, IESGETN, IESFREE

Exit: Returns; IEABEND

IAPLFUN

Module: APLIATRN

Called By: IASCAN, IADECODE, IAENCODE

Description: Finds the internal name for internal embedded VS APL functions.

Calls: IESNAME

Exit: Returns; IEABEND, ITSYSERR (Error)

IAPOW

Module: APLIATRN

Called By: IEDYB, IACAL370, IAIPROD, IAREDU

Description: Performs exponentiation for a floating-point argument.

Calls: IAEXPR, IALOGR, IASQRT

Exit: Returns; IEABEND (Error)

IAQCR

Module: APLIAQFN

Called By: IAMSHARE

Description: Produces the canonical form of a function.

Calls: IAVALNAM, IAFCHNAM, IESFIND, ITPRLINE, IESFREE

Exit: Returns; IEABEND, ITSYSERR (Error)

IAQDL

Module: APLIAQFN

Called By: IAMSHARE

Description: Delays the processing of a function for a specified interval; issues YYDELAY service request.

Calls: IESGINIT, IESFIND, APLFXIIM

Exit: Returns; IEABEND (Error)

IAQDSPEC

Module: APLIATRN

Called By: IACAL370, IESCANG

Description: Processes the following kinds of output: quad, quad prime; shared variable specifications, and system variable specifications.

Calls: IAGOUT, IASHSPEC, IASYSPEC

Exit: Returns; IEABEND (Error)

IAQEX

Module: APLIAQFN

Called By: IAMSHARE

Description: Performs the quad-EX function; that is, it erases the local value of names.

Calls: IAVALNAM, IAFCHNAM, ITDELETE, IESFIND

Exit: Returns; IEABEND (Error)

IAQFX

Module: APLIAQFN

Called By: IAMSHARE

Description: Establishes a function definition from the function's canonical form.

Calls: ITLINEO, ITOKENIZ, ITCLOSET, IESFIND, ITDELETE, IATIDY

Exit: Returns; IEABEND, ITSYSERR (Error)

IAQNC

Module: APLIAQFN

Called By: IAMSHARE

Description: Classifies the current types of name.

Calls: IAVALNAM, IAFCHNAM, IESFIND

Exit: Returns; IEABEND (Error)

IAQNL

Module: APLIAQFN

Called By: IAMSHARE, IADSHARE

Description: Performs the quad-NL system function; that is, returns a character matrix of variable names.

Calls: IESGINIT, IESGETN, IESFIND

Exit: Returns; IEABEND (Error)

IAQSVC

Module: APLIASHF

Called By: IAMSHARE, IADSHARE

Description: Executes both the monadic and dyadic quad-SVC functions.

Calls: APLFXIIM, IAVALNAM, IAFCHNAM, IESFIND, IESGINIT, IESGETN

Exit: Returns; IEABEND, ITSYSERR (Error)

IAQSVO

Module: APLIASHF

Called By: IAMSHARE, IADSHARE

Description: Executes both the monadic and dyadic quad-SVO functions.

Calls: IESGETN, APLFXIIM, IAVALNAM, IAFCHNAM, IASVON, IASFIND, IESGINIT, IESFREE, IESGETV, IESFIND, IARTRACT

Exit: Returns; IEABEND, ITSYSERR (Error)

IAQSVQ

Module: APLIASHF

Called By: IAMSHARE, IADSHARE

Description: Executes both the monadic and dyadic quad-SVQ functions.

Calls: APLFXIIM, IASVON, IATIDY, IESGINIT, IESGETN, IESFIND, IESFREE

Exit: Returns; IEABEND, ITSYSERR (Error)

IAQSVR

Module: APLIASHF

Called By: IAMSHARE

Description: Executes the monadic quad-SVR function.

Calls: IAVALNAM, IAFCHNAM, IASCOPY, IARTRACT, IESFIND

Exit: Returns; IEABEND (Error)

IAQUADS

Module: APLIATRN

Called By: IACAL370, IESCANG

Description: Processes the following kinds of input: quad, quad prime; shared variable reference, and system variable reference.

Calls: IASCOPY, IASYSREF, IESFIND, ITINPUT

Exit: Returns; IEABEND, ITSYSERR (Error)

IAQUADSA

Module: APLIATRN

Called By: IACAL370, IESCANG

Description: References shared or system variables for subscripted specification.

Calls: IASCOPY, IASYSREF

Exit: Returns; IEABEND (Error)

IAREDU

Module: APLIARED

Called By: IESCANG, IACAL370

Description: Performs reduction operation.

Calls: IAPOW, IARESIDU, IABNM, IALOG, IACIRCLE, IESFIND, IESGETN, IESGINIT, IESFREE, IESGETV

Exit: Returns; IEABEND (Error)

IARESIDU

Module: APLIATRN

Called By: IEDYB, IAREDU, IAIPROD, IACAL370, IAENCODE

Description: Calculates residue for floating-point arguments.

Exit: Returns

IAREVARY

Module: APLIAROT

Called By: IEMONAD

Description: Handles reversal of arrays by either performing the operation or by returning with a request for subscripting.

Calls: IESFIND, IESGINIT, IESFREE, IESGETN

Exit: Returns; IEABEND, ITSYSERR (Error)

IAROLL

Module: APLIATRN

Called By: IEMONAD, IACAL370

Description: Calculates monadic random value from a floating-point argument.

Calls: IAIROLL

Exit: Returns; IEABEND (Error)

IAROTA

Module: APLIAROT

Called By: IEDYAD, IACAL370

Description: Handles all cases of rotation of variables by either performing the operation or by returning with a request for subscripting.

Calls: IESFIND, IESGINIT, IESFREE, IESGETN

Exit: Returns; IEABEND, ITSYSERR (Error)

IARTOI

Module: APLIASYV

Called By: IASYSPEC, IAHTSPEC, IAUNSHAD

Description: Converts real value to integer.

Exit: Returns

IARTRACT

Module: APLIASHV

Called By: IAQSVR, IASHRPST, IAQSVO, ITDELETE

Description: Retracts or "unshares" a shared variable; issues YYSRET service request.

Calls: APLFXIIM, IAUNSHR

Exit: Returns; ITSYSERR (Error)

IASCAN

Module: APLIASCN

Called By: IESCANG, IACAL370

Description: Performs scan operation.

Calls: IAPLFUN, IESFIND, IESFREE, IESGINIT, IESGETN

Exit: Returns; IEABEND (Error)

IASCOPY

Module: APLIASHV

Called By: IAQSVR, IAQUADS, IAQUADSA, ITSHV

Description: References a shared variable by issuing YYSREF service request.

Calls: APLFXIIM, IACHK, IESFIND, IESFREE, IATIDY, IESNAME, IASFIND

Exit: Returns; ITSYSERR (Error)

IASFIND

Module: APLIANAM

Called By: IASYSPEC, IASYSREF, IASHSPEC, IAQSVO, IASCOPY

Description: Creates a named temporary copy of a value described by a stack entry argument.

Calls: IESNAME, IESFIND, IESFREE

Exit: Returns

IASHADO

Module: APLIASYV

Called By: IEFUNN, IACAL370

Description: Sends null tab settings to executor when quad-HT is localized.

Calls: APLFXIIM

Exit: Returns

IASHRPST

Module: APLIATRN

Called By: IACAL370, IEUNFN, IEINDD

Description: Performs one of the following actions: complete shared variable subscripted specification, completes system variable subscripted specification, retracts and unshares shared variable locals, or unshadows system variable local to a defined function.

Calls: IASYSPST, IAUNSHAD, IARTRACT, IASHSPEC

Exit: Returns

IASHSPEC

Module: APLIASHV

Called By: IAQDSPEC, IASHRPST

Description: Specifies a shared variable by issuing YYSPEC service request.

Calls: APLFXIIM, IASFIND, IESFREE

Exit: Returns; IEABEND, ITSYSERR (Error)

IASQRT

Module: APLIATRS

Called By: IACIRCLE, IAPOW, IAMDOM, IADDOM

Description: Calculates the square root of an argument.

Exit: Returns; IEABEND (Error)

IASVOFF

Module: APLIASHV

Called By: ITCMOFF, APLINIT

Description: Issues a YYSOFF service request to sign off from the shared variable processor.

Calls: APLFXIIM

Exit: Returns; ITSYSERR (Error)

IASVON

Module: APLIASHV

Called By: IAQSVO, IAQSVQ

Description: Issues a YYSON service request to access the shared variable processor.

Calls: APLFXIIM

Exit: Returns; ITSYSERR (Error)

IASYSPEC

Module: APLIASYV

Called By: IAQDSPEC

Description: Assigns a new value to a system variable.

Calls: IAHTSPEC, IASFIND, IARTOI, APLFXIIM, IESGETV, IESGINIT, ESFIND, IESFREE

Exit: Returns; IEABEND (Error)

IASYSPST

Module: APLIASYV

Called By: IASHRPST

Description: Completes subscripted specification of a system variable.

Calls: IESGETV, IAHTSPEC, IESFREE

Exit: Returns

IASYSREF

Module: APLIASYV

Called By: IAQUADS, IAQUADSA

Description: Processes system variable references; obtains current value of system variable.

Calls: APLFXIIM, IASFIND, IATIDY, IESFIND, ITTIMSUB, ITFNLNO, USASH

Exit: Returns; IEABEND (Error)

IATABREF

Module: APLIASYV

Called By: APLIINIT

Description: Gets current tab settings from executor; assigns value to quad-HT.

Calls: IESFREE, IESFIND, APLFXIIM

Exit: Returns

IATIDY

Module: APLIATRN

Called By: IEFIND, IACAL370, IAEXECTE, IAFCHNAM, IAQFX, IAQSVQ, IASCOPY, IASYSREF, ITCMGROU,

ITCMSYMB, ITSAVWS, ITCOPIN, ITINPUT

Description: Collects active value blocks in the low address end of free space to maximize the size of the unallocated block in free space; it adjusts the address table entries to reflect the change.

Exit: Returns; ITSYSERR (Error)

IATKDP

Module: APLIATAK

Called By: IETKDP, IACAL370

Description: Handles cases of take and drop where the left argument is either a vector of zero, or a vector containing more than one element.

Calls: IESGINIT, IESGETN, IESFREE, IESFIND

Exit: Returns; IEABEND (Error)

IATOBCD

Module: APLIATBC

Called By: IAGOUT, IAMFORM, ITCMQUOT, ITCMWSSI, ITPRWSID, ITCMSTAC, ITCMSYMB, IAGFMT, ITEXECUT, ITLIBMSG, ITPRLINE, ITPRNUM, IAGFMT2

Description: Determines internal type of a given data element; converts the element to Z-codes according to its format and data type.

Exit: Returns

IATOBCD2

Module: APLIATBC

Called By: IADFORM

Description: Accepts a floating-point value as input and converts it to decimal representation according to a given format; determines the number of significant digits in the decimal exponent and returns this value to the calling routine.

Exit: Returns

IAUNSHAD

Module: APLIASYV

Called By: IASHRPST

Description: Unshadows a system variable; discards local value; restores shadowed value.

Calls: APLFXIIM, IESFREE, IESGETV, IAHTSPEC, IARTOI

Exit: Returns

IAUNSHR

Module: APLIASHV

Called By: IARTRACT, ITSHV

Description: Removes the shared status from variable.

Calls: IESFREE

Exit: Returns

IAVALNAM

Module: APLIANAM

Called By: IAQNC, IAQEX, IAQCR, IAQSVO, IAQSVR, IAQSVC

Description: Validates the right argument of the following quad functions: CR, EX, NC, SVC, SVR, SVO.

Exit: Returns

IEABEND

Module: APLIEXAR

Called By: Exarch and appendage routines

Description: Processes abnormal termination or request for translator service.

Exit: IEXIT

IECHIX

Module: APLIEMND

Called By: IECMEX, IEMONAD, IESCANG

Description: Checks index of indexed operator.

Exit: Returns; IEABEND (Error)

IECMEX

Module: APLIECMX

Called By: IACMX, IEDYAD

Description: Carries out compress or expand primitive.

Calls: IEGINITL, IEGETNI, IEGTSPAC, IECHIX

Exit: IESCANG, IEABEND (Error)

IECOMMA

Module: APLIERHO

Called By: IACOMMA, IEDYAD

Description: Performs the catenate and laminate operations.

Calls: IEGTSPAC, IECOPY

Exit: IESCANG, IEABEND (Error)

IECONVR

Module: APLIEFCH

Called By: IEGINITI, IEGINITL, IEGETNI, IEGETNL, IESCANG

Description: Converts a real value to an integer.

Exit: Returns; IEABEND (Error)

IECOPY

Module: APLIERHO

Called By: IECOMMA, IEINDD, IERSHP, IETKDP

Description: Copies elements from one free space entry to another with data conversion if necessary.

Calls: IEGINITR, IEGETNR, IEGETNI

Exit: Returns

IEDATTN

Module: APLIEFXR

Called By: Microcode

Description: Handles double attention or quantum end discovered by microcode.

Exit: IEABEND

IEDYAD

Module: APLIESCA

Called By: IEMONAD, IESCANG

Description: Sets up arguments for dyadic operations. Calls or exits to routine that performs them.

Calls: IEGETV, IAENCODE, IAROTA, IADSHARE, IADTRAN, IADDOM, IADFORM, IADECODE, IAIPROD

Exit: IEDYB, IERSHP, IETKDP, IEEPSIOT, IECMEX, IECOMMA, IEINDB, IEABEND (Error)

IEDYB

Module: APLIEFCH

Called By: IEDYAD, IADYB

Description: Performs dyadic scalar and outer product operations.

Calls: IEGETV, IEGINITI, IEGINITL, IEGINITR, IEGETNI, IEGETNL, IEGETNR, IAPOW, IACIRCLE, IEGTSPAC, IESPACST, IEFIND, IEFREE, IADEAL, IARESIDU, IABNM, IALOG

Exit: IESCANG, IEABEND (Error)

IEEPSIOT

Module: APLIEPSI

Called By: IEDYAD, IADYB

Description: Carries out membership and "index of" operations.

Calls: IESPACST, IEGINITR, IEGETNR

Exit: IESCANG, IEABEND (Error)

IEFIND

Module: APLIESPA

Called By: IEGTSPAC, IESFIND, IEDYB, IEINDD, IEMONAD, IESCANG, IESYNN

Description: Allocates a block of free space and the next available internal name. Input is length in bytes.

Calls: IENAME, IATIDY

Exit: Returns; IEABEND (Error)

IEFREE

Module: APLIESPA

Called By: IESFREE, IEDYB, IEGOGOMN, IEGOGOSC, IEINDD, IEMONAD, IESCANG, IEUNFN

Description: Frees an object. Frees internal name if temporary, and block of free space if remote.

Exit: Returns

IEFUNN

Module: APLIEFNM

Called By: IESCANG

Description: Builds a function call block on the operation stack and prepares to execute a function.

Calls: IEGETV, IESYNN, IAEXSTCK, IASHADO

Exit: IESCANG, IEABEND (Error)

IEGETNI

Module: APLIEFCH

Called By: IESGETN, IECMEX, IECOPY, IEDYB, IEINDD, IEMONAD, IERSHP

Description: Gets the next element of a multi-element argument, or the only element of a single-element argument and returns the integer value for that argument.

Calls: IECONVR

Exit: Returns; IEABEND (Error)

IEGETNL

Module: APLIEFCH

Called By: IESGETN, IEDYB

Description: Gets the next element of a multi-element argument, or the only element of a single-element argument and returns the integer value for that argument.

Calls: IECONVR

Exit: Returns; IEABEND (Error)

IEGETNR

Module: APLIEFCH

Called By: IESGETN, IECOPY, IEDYB, IEEPSIOT, IEMONAD

Description: Gets the next element of a multi-element argument, or the only element of a single-element argument and returns the integer value for that argument.

Exit: Returns

IEGETV

Module: APLIEFCH

Called By: IESGETV, IEDYAD, IEDYB, IEFUNN, IEINDD, IEMONAD, IERSHP, IESCANG

Description: Sets up argument block for fetching of elements.

Exit: Returns; IEABEND (Error)

IEGINITI

Module: APLIEFCH

Called By: IESGINIT, IEDYB, IEGOGOMN, IEINDD, IEMONAD, IERSHP, IETKDP

Description: Gets the first element of an argument, and returns the integer value for that element.

Calls: IECONVR

Exit: Returns; IEABEND (Error)

IEGINITL

Module: APLIEFCH

Called By: IESGINIT, IECMEX, IEDYB

Description: Gets the first element of an argument, and returns the logical value for that argument.

Calls: IECONVR

Exit: Returns; IEABEND (Error)

IEGINITR

Module: APLIEFCH

Called By: IESGINIT, IECOPY, IEDYB, IEEPSIOT, IEINDD, IEMONAD

Description: Gets the first element of an argument, and returns the real value for that element.

Exit: Returns; IEABEND (Error)

IEGOGOMN

Module: APLIEFNM

Called By: IEMONAD

Description: Processes a normal branch operation.

Calls: IEFREE, IEGINITI

Exit: IESCANG, IEUNFN, IEXIT, IEABEND (Error)

IEGOGOSC

Module: APLIEFNM

Called By: IESCANG

Description: Processes fast branch operation.

Calls: IEFREE

Exit: IESCANG, IEUNFN, IEXIT, IEABEND (Error)

IEGTSPAC

Module: APLIESPA

Called By: IESPACST, IECMEX, IECOMMA, IEDYB, IEINDD, IEMONAD, IERSHP, IETKDP

Description: Allocates a block of free space and the next available internal name. Input is descriptor, element count, and rank.

Calls: IEFIND

Exit: Returns; IEABEND (Error)

IEINDB

Module: APLIEIDX

Called By: IACAL370, IEDYAD, IEMONAD

Description: Completes the transpose and rotate operations.

Calls: IEINDD

Exit: IESCANG, IEABEND (Error)

IEINDD

Module: APLIEIDX

Called By: IEINDB, IESCANG

Description: Performs subscripted reference and subscripted assignment type of subscripting.

Calls: IEGETV, IEGINITI, IEGETNI, IEGINITR, IEFIND, IEGTSPAC, IESPACST, IEFREE, IECOPY, IASHRPST

Exit: IESCANG, IEABEND (Error)

IELDSTK

Module: APLIEXAR

Called By: IACAL370, IEXARCH

Description: Loads microcode stack registers from operation stack.

Exit: Returns

IEMONAD

Module: APLIEMND

Called By: IESCANG

Description: Performs some monadic operations. Calls or exits to routines that perform other monadic operations.

Calls: IESPACST, IEGTSPAC, IEFIND, IESYNN, IECHIX, IENAME, IEFREE, IEGETV, IEGINITI, IEGINITR, IEGETNI, IEGETNR, IAREVARY, IAIROLL, IAROLL, IAFACTRL, IAFLCL, IAGRADE, IAEXECTE, IAMTRAN, IAMDOM, IAMFORM, IAMSHARE

Exit: IEDYAD (monadic operations done as dyadic), IEGOGOMN (branch), IEINDB (reverse, transpose), IESCANG (operation completed), IEABEND (Error)

IENAME

Module: APLIESPA

Called By: IEFIND, IESFIND, IESNAME, IEMONAD, IEUNFN, IESCAN

Description: Finds the next available entry in the address table.

Calls: IAEXNAME

Exit: Returns; IEABEND (Error)

IERSHP

Module: APLIERHO

Called By: IEDYAD

Description: Performs the reshape operation.

Calls: IEGETV, IEGINITI, IEGETNI, IEGTSPAC, IECOPY

Exit: IESCANG, IEABEND (Error)

IESCANG

Module: APLIESCA

Called By: IEXARCH, IECMEX, IEINDD, IEMONAD, IEEPSIOT, IECOMMA, IERSHP, IETKDP, IEDYB, IEFUNN, IEGOGOMN, IEGOGOSC, IEUNFN

Description: Basic interpreter module; receives control when there is a function statement to be scanned and executed; scans the statement; does syntax analysis; selects next action to be performed; processes result of an operation; resumes statement scan.

Calls: IEFIND, IEFREE, IEGETV, IECHIX, IECONVR, IESYNN, IENAME, IAEXSTCK, IAREDU, IASCAN, IAQUADS, IAQUADSA, IAQDSPEC

Exit: IEDYAD (dyadic operations), IEMONAD (monadic operations), IEINDD (subscripting), IEFUNN (function call), IEGOGOSC (branch), IEXIT (end of input), IEABEND (Error)

IESFIND

Module: APLIESPA

Called By: Appendage and translator routines

Description: Provides access to IENAME and IEFIND for non-exarch routines.

Calls: IENAME, IEFIND

Exit: Returns

IESFREE

Module: APLIESPA

Called By: Appendage and translator routines

Description: Provides access to IEFREE for non-exarch routines.

Calls: IEFREE

Exit: Returns

IESGETN

Module: APLIEFCH

Called By: IADECODE, IAENCODE, IASYSPEC, IASYSPST, IAUNSHAD, IAQSVO, IAREDU

Description: Provides access to IEGETNI, IEGETNL, and IEGETN for non-exarch routines.

Calls: IEGETNI, IEGETNL, IEGETNR

Exit: Returns; IEABEND (Error)

IESGETV

Module: APLIEFCH

Called By: IADECODE, IAENCODE, IASYSPEC, IASYSPST, IAUNSHAD, IAQSVO, IAREDU

Description: Provides access to IEGETV for non-exarch routines.

Calls: IEGETV

Exit: Returns; IEABEND (Error)

IESGINIT

Module: APLIEFCH

Called By: Appendage routines

Description: Provides access to IEGINITI, IEGINITL, and IEGINITR for non-exarch routines.

Calls: IEGINITI, IEGINITL, IEGINITR

Exit: Returns; IEABEND (Error)

IESNAME

Module: APLIESPA

Called By: IASFIND, IAPLFUN, IASCOPY

Description: Provides access to IENAME for use of non-exarch routines.

Calls: IENAME

Exit: Returns

IESPACST

Module: APLIESPA

Called By: IEDYB, IEEPSIOT, IEINDD, IEMONAD

Description: Allocates a block of free space and the next available internal name. Input is descriptor and model variable.

Calls: IEGTSPAC

Exit: Returns; IEABEND (Error)

IESTOSTK

Module: APLIEXAR

Called By: IACAL370, IEXARCH

Description: Stores microcode register stack items in operation stack.

Calls: IAEXSTCK

Exit: Returns; IEABEND (Error)

IESUNFUN

Module: APLIEFNM

Called By: ITERRORS

Description: Removes function call block from the operation stack.

Calls: IEUNFN

Exit: Returns

IESYNN

Module: APLIESPA

Called By: IEFUNN, IEMONAD, IESCANG

Description: Makes a copy or a synonym of a variable.

Calls: IEFIND

Exit: Returns; IEABEND (Error)

IETKDP

Module: APLIETAK

Called By: IEDYAD

Description: Performs take and drop operations; if the left argument is greater than one element, or is a vector of zero elements, IATKDP is called.

Calls: IEGINITI, IEGTSPAC, IECOPY, IATKDP

Exit: IESCANG, IEABEND (Error)

IEUNFN

Module: APLIEFNM

Called By: IESUNFUN, IEGOGOMN, IEGOGOSC

Description: Removes a function call block from the operation stack; restores the workspace to its status at the time the function was called.

Calls: IEFREE, IENAME, IASHRPST

Exit: IESCANG, IEABEND (Error)

IEXARCH

Module: APLIEXAR

Called By: ITEXECUT

Description: Sets up operation stack for exarch or microcode, calls one. If microcode called, handles return to translator.

Calls: IAEXSTCK, IESTOSTK, IELDSTK, Microcode

Exit: IESCANG (Exarch), IEXIT (End of input), IEABEND (Error)

IEXIT

Module: APLIEXAR

Called By: IESCANG, IEABEND, IEGOGOSC, IEGOGOMN, IEXARCH

Description: Returns to translator when error is found, for services (print, trace, stop, escape, attention), or end of operation stack.

Exit: Returns to ITEXECUT

ITBFTYO

Module: APLITSUB

Called By: ITERRORS, ITFDEDIT, ITCMSI, ITCMSINL

Description: Prints via YYTYO service request, the contents of WSMBUFF.

Calls: APLFXIIM

Exit: Returns

ITBLDID

Module: APLITIDS

Called By: ITSTSRCH, ITBLDQD, ITSYSCMD

Description: Isolates a printname string and determines its length; those characters beyond the maximum length are ignored.

Exit: Returns

ITBLDQD

Module: APLITIDS

Called By: ITLINEO, ITOKENIZ, ITFDNWLN

Description: Validates a name beginning with a QUAD, and translates it to a token.

Calls: ITBLDID

Exit: Returns

ITCKALPN

Module: APLITSUB

Called By: ITCMGROU, ITCMERAS

Description: Checks a string of characters for initial alphabetic, followed by alphameric.

Exit: Returns

ITCLOSET

Module: APLITFDC

Called By: ITFDCLOS, ITCOPIN, IAQFX, CVFUNC, CVSHIP, ITFDOPEN

Description: Finds space for a function object, and returns a temporary name.

Calls: IESFIND

Exit: Returns

ITCMCLEA

Module: APLITCML

Called By: ITSYSCMD

Description: Executes the)CLEAR command.

Calls: APLFXIIM, ITLIBMSG

Exit: Returns

ITCMCONT

Module: APLITCMT

Called By: ITSYSCMD

Description: Executes the)CONTINUE command.

Calls: ITSAVWS, ITCMOFF

Exit: Returns; ITINPINI (command issued in quad-input), ITSYSERR (Error)

ITCMCOPO

Module: APLITCPO

Called By: APLIINIT

Description: Transmits objects from a copy source workspace to a copy sink workspace via a YYCOPO service request.

Calls: ITPRLINE, ITSTSRCH, APLFXIIM, ITUSAG

Exit: Returns; ITSYSERR (Error)

ITCMCOPY

Module: APLITCMC

Called By: ITSYSCMD

Description: Initiates and terminates)COPY command processing.

Calls: ITCOPIN, ITLIBMSG, APLFXIIM

Exit: Returns; ITSYSERR (Error)

ITCMDOST

Module: APLITCMS

Called By: ITCMSTAC, ITCMSYMB

Description: Relocates the pointers, and moves the stack area or symbol table area around when a change in size has been indicated.

Calls: ITSQUIRT

Exit: Returns

ITCMDROP

Module: APLITCML

Called By: ITSYSCMD

Description: Executes the)DROP command.

Calls: APLFXIIM, ITLIBMSG

Exit: Returns

ITCMERAS

Module: APLITCME

Called By: ITSYSCMD

Description: Prepares for execution of)ERASE command; calls ITDELETE routine to complete processing.

Calls: ITSTSRCH, ITNAMINI, ITUSAG, ITSQUIRT, ITDELETE, ITPRINTC, ITLOUT, IESFREE, APLFXIIM, ITCKALPN

Exit: Returns; ITSYSERR (Error)

ITCMFNS

Module: APLITCMF

Called By: ITSYSCMD

Description: Calls ITCMFVG to print a)FNS report.

Calls: ITCMFVG

Exit: Returns

ITCMFVG

Module: APLITCMF

Called By: ITCMFNS, ITCMVARS, ITCMGRPS

Description: Executes the)FNS,)VARS, and)GRPS commands; it finds, sorts, and prints the object names.

Calls: ITUSAG, ITXBLNL, ITSQUIRT, ITLOUT

Exit: Returns

ITCMGROU

Module: APLITCMG

Called By: ITSYSCMD

Description: Executes the)GROUP command.

Calls: ITSTSRCH, ITUSAG, IESFIND, ITLOUT, APLFXIIM, IATIDY, IESFREE, ITSQUIRT, ITPRINTC, ITCKALPN

Exit: Returns

ITCMGRP

Module: APLITCMG

Called By: ITSYSCMD

Description: Executes the)GRP command.

Calls: ITPRINTC, ITUSAG, ITSTSRCH, ITPRNAME, ITLOUT

Exit: Returns; ITSYSERR (Error)

ITCMGRPS

Module: APLITCMF

Called By: ITSYSCMD

Description: Calls ITCMFVG to print a)GRPS report.

Calls: ITCMFVG

Exit: Returns

ITCMLIB

Module: APLITCML

Called By: ITSYSCMD

Description: Executes the)LIB command.

Calls: APLFXIIM, ITLOUT, ITLIBMSG

Exit: Returns

ITCMLOAD

Module: APLITCML

Called By: ITSYSCMD, APLIINIT

Description: Executes the)LOAD command.

Calls: APLFXIIM, ITLIBMSG

Exit: Returns

ITCMMSG

Module: APLITCMT

Called By: ITSYSCMD

Description: Executes the)MSG command.

Calls: APLFXIIM

Exit: Returns; ITFORCOF

ITCMOFF

Module: APLITCMT

Called By: ITCMCONT, ITSYSCMD

Description: Executes the)OFF command.

Calls: IASVOFF, APLFXIIM, ITLIBMSG

Exit: Returns; ITSYSERR (Error)

ITCMOPR

Module: APLITCMT

Called By: ITSYSCMD

Description: Executes the)OPR command.

Calls: APLFXIIM

Exit: Returns; ITFORCOF

ITCMPCOP

Module: APLITCMC

Called By: ITSYSCMD

Description: Initiates and terminates)PCOPY command processing.

Calls: ITCOPIN, ITLIBMSG, APLFXIIM

Exit: Returns; ITSYSERR (Error)

ITCMQUOT

Module: APLITCML

Called By: ITSYSCMD

Description: Executes the)QUOTA command.

Calls: IATOBCD, ITLOUT, APLFXIIM

Exit: Returns

ITCMSAVE

Module: APLITCML

Called By: ITSYSCMD

Description: Executes the)SAVE command.

Calls: ITSAVWS

Exit: Returns; ITINPINI (command issued in quad-input)

ITCMSI

Module: APLITCMI

Called By: ITSYSCMD

Description: Executes the)SI command.

Calls: ITPRINTC, ITPRNAME, ITPRNUM, ITXBLNL, ITFNLNO, ITSQUIRT, ITUSASH, ITBFTYO

Exit: Returns; ITSYSERR (Error)

ITCMSINL

Module: APLITCMI

Called By: ITSYSCMD

Description: Executes the)SINL command.

Calls: ITPRINTC, ITFNLNO, ITPRNUM, ITXBLNL, ITPRNAME, ITSQUIRT, ITUSASH, ITBFTYO

Exit: Returns; ITSYSERR (Error)

ITCMSTAC

Module: APLITCMS

Called By: ITSYSCMD

Description: Executes the)STACK command.

Calls: IATOBCD, ITCMDOST, ITLOUT, ITSQUIRT

Exit: Returns

ITCMSYMB

Module: APLITCMS

Called By: ITSYSCMD

Description: Executes the)SYMBOLS command.

Calls: ITCMDOST, IATOBCD, ITLOUT, ITSQUIRT, IATIDY, APLFXIIM

Exit: Returns; ITSYSERR (Error)

ITCMVARS

Module: APLITCMF

Called By: ITSYSCMD

Description: Calls ITFCMFVG to print a)VARS report.

Calls: ITCMFVG

Exit: Returns

ITCMWSID

Module: APLITCML

Called By: ITSYSCMD

Description: Executes the)WSID command.

Calls: APLFXIIM, ITPRWSID, ITLIBMSG

Exit: Returns

ITCMWSSI

Module: APLITCML

Called By: ITSYSCMD

Description: Executes the)WSSIZE command.

Calls: ITLOUT, IATOBCD

Exit: Returns

ITCOPIN

Module: APLITCPI

Called By: ITCMCOPY, ITCMPCOP

Description: Receives data from a copy source workspace via YYCOPI service request; defines it in the active workspace.

Calls: ITSTSRCH, ITUSAG, IESFIND, ITLINEO, ITCLOSET, ITPRNAME, ITPRINTC, ITSQUIRT, ITOKENIZ, ITLOUT, APLFXIIM, IESFREE, ITDELETE, IATIDY

Exit: Returns; ITSYSERR (Error)

ITDELETE

Module: APLITCME

Called By: ITCMERAS, IAQEX, IAQFX, ITCOPIN

Description: Erases the variable, function, or group.

Calls: IESFREE, ITUSADF, ITFDKILL, IARTRACT

Exit: Returns; ITSYSERR (Error)

ITEMPFUN

Module: APLITFUN

Called By: ITINPUT, IAEXECTE

Description: Builds a temporary function in free space for immediate execution, for quad-input, or for the primitive function execute.

Calls: ITOKENIZ, IESFIND

Exit: Returns

ITERRORS

Module: APLITERR

Called By: ITEXECUT, ITSAVWS, ITINPUT, ITTYIZ

Description: Handles execution time errors; cleans up the operation stack, as required.

Calls: IESUNFUN, IESFREE, ITFNLNO, ITPRLINE, ITPRFNLN, ITBFTYO, ITLOUT, ITXBLNL, ITSQUIRT, APLFXIIM

Exit: Returns, if error in quad-input; ITINPINI, ITSYSERR (Error)

ITEXECUT

Module: APLITEX

Called By: ITINPUT

Description: Establishes an environment for the interpreter and then uses APLCALL to call IEXARCH; it handles normal or exceptional returns.

Calls: ITFNLNO, ITFETCH, IESFREE, ITPRFNLN, IAGOUT, ITPRINTC, IAEXSTCK, ITERRORS, IEXARCH, ITSQUIRT, IATOBCD, ITXBLNL, ITLOUT, APLFXIIM

Exit: Returns; ITSYSERR (Error)

ITFDCLOS

Module: APLITFDC

Called By: ITFDEDIT

Description: Closes a function definition by converting source to internal text.

Calls: ITLINEO, ITOKENIZ, ITCLOSET, ITUSADF, IESFREE, ITUSAG, ITFDTSOF, APLFXIIM, ITFDKILL

Exit: Returns

ITFDCVT

Module: APLITNCV

Called By: ITFDEDIT

Description: Converts function line numbers to internal form.

Calls: ITNUMCVT

Exit: Returns

ITFDEDIT

Module: APLITFDE

Called By: ITFDOPEN, ITINPUT

Description: Processes an input line entered in function definition mode; performs editing actions.

Calls: ITBFTYO, ITUSAG, ITLOUT, ITXBLNL, ITPRLINE, ITTYIZ, ITLINEO, ITFDCLOS, IESFREE, IESFIND, ITFDCVT, ITPRNUM, ITFDKILL, ITFDNWLN, APLFXIIM

Exit: Returns

ITFDKILL

Module: APLITFDC

Called By: ITFDEDIT, ITDELETE, ITFDCLOS

Description: Takes the user out of function definition mode.

Calls: IESFREE

Exit: Returns

ITFDNWLN

Module: APLITFDN

Called By: ITFDEDIT

Description: Stores a new function-statement in free space; enters names occurring in it in the symbol table.

Calls: IESFIND, ITSTSRCH, ITBLDQD

Exit: Returns

ITFDOPEN

Module: APLITFDO

Called By: ITINPUT

Description: Examines a function open request, and either rejects it or sets the edit globals to enter function definition mode.

Calls: ITUSAG, ITLINEO, ITFDEDIT, ITCLOSET, ITFDTSOF, IESFIND

Exit: Returns

ITFDTSOF

Module: APLITFDC

Called By: ITFDCLOS, ITFDOPEN

Description: Turns off all trace and stop bits in a function that is being locked.

Exit: Returns

ITFETCH

Module: APLITFCH

Called By: ITEXECUT

Description: Gets an integer value from the ravel of an M-entry and returns an element count.

Exit: Returns

ITFNLNO

Module: APLITSUB

Called By: ITEXECUT, IASYSREF, ITCMSI, ITCMSINL, ITERRORS

Description: Returns a line number corresponding to a given offset into a function.

Exit: Returns; ITSYSERR (Error)

ITFORCOF

Module: APLITINP

Called By: ITINPUT, ITTYIZ, ITCMOPR, ITCMMSG

Description: Forces a terminal user off VS APL when the executor so indicates by issuing a)CONT command.

Exit: ITSYSCMD

ITININT

Module: APLITNCV

Called By: ITSYSCMD

Description: Converts an integer constant to internal form.

Calls: ITNUMCVT

Exit: Returns

ITINPINI

Module: APLITINP

Called By: APLIINIT, ITCMSAVE, ITCMCONT, ITERRORS

Description: Provides an entry point to ITINPUT to begin execution of a newly loaded workspace or to resume execution after an error.

Exit: ITINPUT

ITINPUT

Module: APLITINP

Called By: ITINPINI, IAQUADS

Description: Prompts and receives terminal input.

Calls: ITTYIZ, ITSYSCMD, ITFDEDIT, ITFDOPEN, ITEMPFUN, ITEXECUT, ITTYERR, ITPRNUM, APLFXIIM, ITLOUT, IATIDY

ITLIBMSG

Module: APLITCML

Called By: ITCMPCOP, ITCMCOPY, ITCMCLEA, ITCMDROP, ITCMLIB, ITCMLOAD, ITSAVWS, ITCMWSID, ITCMOFF, APLIINIT

Description: Prints message after library service request processing.

Calls: ITLOUT, APLFXIIM, ITPRWSID, ITTIME, IATOBCD

Exit: Returns

ITLINEO

Module: APLITHDR

Called By: ITFDOPEN, ITFDEDIT, ITFDCLOS, ITCOPIN, IAQFX, CVSHIP, CVFUNC

Description: Inspects line zero of a function for correct syntax, and then constructs the function header codestring.

Calls: ITSTSRCH, ITBLDQD

Exit: Returns

ITLOUT

Module: APLITSUB

Called By: ITCMERAS, ITCMFVG, ITCMGROU, ITCMLIB, ITERRORS, ITEXECUT, ITFDEDIT, ITCMSTAC, ITCMSYMB, ITLIBMSG, ITPRWSID, ITCMWSSI, ITCMGRP, ITCMQUOT, ITCOPIN, ITINPUT

Description: Drops trailing blanks from data in WSMBUFF; appends a new line, and prints the line.

Calls: APLFXIIM

Exit: Returns

ITNAMINI

Module: APLITCME

Called By: ITCMERAS

Description: Initializes the name list printout for "Objects Not Found" and "Objects Not Copied" messages.

Exit: Returns; ITSYSERR (Error)

ITNUMCVT

Module: APLITNCV

Called By: ITFDCVT, ITOKENIZ, ITININT

Description: Converts numeric constant character strings into internal form.

Exit: Returns

ITOKENIZ

Module: APLITLXS

Called By: ITEMPFUN, ITFDCLOS, ITCOPIN, IAQFX, CVSHIP, CVFUNC

Description: Scans a string of text and converts it to a codestring.

Calls: ITNUMCVT, ITSTSRCH, ITBLDQD

Exit: Returns

ITPRFNLN

Module: APLITSUB

Called By: ITEXECUT, ITERRORS

Description: Takes the internal name of a function and an offset into it and puts the printname and line number in WSMBUFF.

Calls: ITPRNAME, ITXBLNL, ITSQUIRT

Exit: Returns

ITPRINTC

Module: APLITSUB

Called By: ITCOPIN, ITCMERAS, ITCMGROU, ITCMGRP, ITCMSI, ITCMSINL, ITEXECUT, ITPRNAME

Description: Takes a single character and catenates it to the current line in WSMBUFF.

Calls: APLFXIIM

Exit: Returns

ITPRLINE

Module: APLITPRL

Called By: ITFDEDIT, ITERRORS, ITCMCOPO, IAQCR

Description: Takes the internal name of a function and a line number within that function, and displays the line in the workspace area requested by the caller.

Calls: IATOBCD

Exit: Returns; ITSYSERR (Error)

ITPRNAME

Module: APLITSUB

Called By: ITCMGRP, ITCMSI, ITCOPIN, ITCMSINL, ITPRFNLN

Description: Takes the internal name of an object and catenates its printname to current line in WSMBUFF.

Calls: ITPRINTC, ITSQUIRT

Exit: Returns

ITPRNUM

Module: APLITSUB

Called By: ITFDEDIT, ITCMSINL, ITCMSI, ITINPUT

Description: Takes the function editor's representation of a line number and puts the bracketed line number in WSMBUFF.

Calls: IATOBCD

Exit: Returns

ITPRWSID

Module: APLITCML

Called By: ITCMWSID, ITLIBMSG

Description: Converts a workspace identifier as defined in PDSD to printable form, puts it in WSMBUFF, and prints it.

Calls: IATOBCD, ITLOUT

Exit: Returns

ITSAVWS

Module: APLITCML

Called By: ITCMSAVE, ITCMCONT

Description: Saves a workspace.

Calls: ITSHV, IATIDY, ITERRORS, APLFXIIM, ITLIBMSG

Exit: Returns

ITSHV

Module: APLITCML

Called By: ITSAVWS, APLIINIT

Description: Copies or retracts each shared variable in the workspace.

Calls: IASCOPY, APLFXIIM, IAUNSHR

Exit: Returns; ITSYSERR (Error)

ITSQUIRT

Module: APLITSUB

Called By: ITCOPIN, ITCMDOST, ITCMERAS, ITCMFVG, ITCMGROU, ITCMSTAC, ITCMSYMB, ITEXECUT, ITCMSI, ITCMSINL, ITERRORS, ITPRNAME, ITPRFNLN

Description: Takes a string of characters and concatenates them with the current line in WSMBUFF.

Calls: APLFXIIM

Exit: Returns

ITSTSRCH

Module: APLITIDS

Called By: ITLINEO, ITCMGRP, ITCMGROU, ITCMCOPO, IAFCHNAM, CVGRUP, CVVARB, ITCOPIN, ITCMERAS, ITFDNWLN, ITOKENIZ

Description: Finds or enters a printname in the symbol table and returns its internal name.

Calls: ITBLDID, IESFIND

Exit: Returns; ITSYSERR (Error)

ITSYSCMD

Module: APLITCMD

Called By: ITINPUT, ITFORCOF

Description: Analyzes syntax of system commands and executes those commands by calling the proper translator routine. Before executing each command, the executor is called (YYCMO).

Calls: ITBLDID, APLFXIIM, ITININT, ITCMCLEA, ITCMCONT, ITCMCOPY, ITCMDROP, ITCMERAS, ITCMFNS, ITCMGROU, ITCMGRP, ITCMGRPS, ITCMLIB, ITCMLOAD, ITCMMSG, ITCMOFF, ITCMOPR, ITCMPCOP, ITCMQUOT, ITCMSAVE, ITCMSI, ITCMSINL, ITCMSTAC, ITCMSYMB, ITCMVARS, ITCMWSID, ITCMWSSI

Exit: Returns

ITSYSERR

Module: APLITINI

Called By: Interpreter and translator routines.

Description: Builds system error information; requests executor to type information on user terminal and system log; takes dump of workspace.

Calls: APLFXIIM

Exit: APLFXIIM with YYCLEAR service request.

ITTIME

Module: APLITSUB

Called By: ITLIBMSG

Description: Formats the date and time in the output buffer when it is given a time value.

Calls: ITTIMSUB

Exit: Returns

ITTIMSUB

Module: APLITSUB

Called By: ITTIME, IASYSREF

Description: Calculates the year, month, day, hour, minute, second, and millisecond values in WSMITSTR from a time value.

Exit: Returns

ITTYERR

Module: APLITSUB

Called By: ITINPUT, ITTYIZ

Description: Prints the error report for an error discovered during initial string processing.

Calls: APLFXIIM

Exit: Returns

ITTYIZ

Module: APLITINP

Called By: ITINPUT, ITFDEDIT

Description: Handles possible errors occurring after YYTYI and YYTYOI.

Calls: APLFXIIM, ITTYERR

Exit: Returns; ITFORCOF, ITSYSERR, ITERRORS (Error)

ITUSADF

Module: APLITUSG

Called By: ITDELETE, ITFDCLOS

Description: Marks all pendant and suspended occurrences of a name as damaged.

Exit: Returns

ITUSAG

Module: APLITUSG

Called By: ITFDOPEN, ITFDEDIT, ITFDCLOS, ITCMGRP, ITCMFVG, ITCMERAS, ITCMGROU, ITCOPIN, ITCMCOPO

Description: Gets the most global referent of an internal name by examining the operation stack.

Exit: Returns

ITUSASH

Module: APLITCMI

Called By: IASYSREF, ITCMSI, ITCMSINL

Description: Shows an object as it was defined when a pendant or suspended function was active.

Exit: Returns; ITSYSERR (Error)

ITXBLNL

Module: APLITSUB

Called By: ITEXECUT, ITPRFNLN, ITCMFVG, ITCMSI, ITCMSINL, ITFDEDIT, ITERRORS

Description: Deletes trailing blanks from a line in the buffer, and appends a new line character.

Calls: APLFXIIM

Exit: Returns

KABEXIT

Module: APLKADSP

Called By: CICS/VS on program checks or abnormal termination

Description: Part of the control of the user session task performed by the CICS/VS executor. Handles abnormal terminations.

Calls: Entry points KYYOFF, KPCREG, KIFONEXT. Macro APLKEXIT. CICS/VS macros DFHDC, DFHPC (RESETXIT), DFHIR, DFHSC (GETMAIN)

Exit: Any process abend exit routine; IFONEXT, APLKADSP

KABOOTS

Module: APLKASTB

Called By APLKSON macro

Description: Part of the CICS/VS executor. Initializes the global table and/or the shared storage manager.

Calls: Entry points APLKAGBL, APLKSSUB. CICS/VS macros DFHSC (GETMAIN, FREEMAIN), DFHPC (ABEND, LOAD), DFHKC (ATTACH, WAIT)

Exit: Returns or APLKSSR

KADEPON

Module: APLKADSP

Called By: Entry point APLKSVI

Description: Part of the CICS/VS executor. Initiates dependent auxiliary processors.

Calls: CICS/VS macros DFHSC (GETMAIN), DFHPC (LOAD)

Exit: Returns

KCASE2Q

Module: APLKSSUB

Called By: Entry point APLKSSR

Description: Part of the CICS/VS executor shared storage manager interface. Handles queries for all items related to caller.

Exit: Returns

KCASE3Q

Module: APLKSSUB

Called By: Entry point APLKSSR

Description: Part of the CICS/VS executor shared storage manager interface. Handles queries for all items related to caller and listed partners.

Exit: Returns

KCATOFF

Module: APLKMSCB

Called By: Entry point APLFXIIM

Description: Part of the interpreter interface provided by the CICS/VS executor. Executes the YYATOFF service request (a request to turn off the asynchronous bits in the PERTERM header).

Exit: Returns

KCDELAY

Module: Entry point APLKMSCA

Called By: APLFXIIM

Description: Part of the interpreter interface provided by the CICS/VS executor. Executes the YYDELAY service request (a request to delay processing for x seconds).

Calls: Macros APLKEXIT, APLKG (CANCEL, DELAY), APLKWAIT

Exit: Returns

KCDUMP

Module: APLKMSCA

Called By: Entry point APLFXIIM

Description: Part of the interpreter interface provided by the CICS/VS executor. Executes the YYDUMP service request (a request to dump the user's workspace and PERTERM header).

Calls: Macro APLKEXIT. CICS/VS macro DFHDC (PARTIAL)

Exit: Returns

KCLEANUP

Module: APLKSSUB

Called By: Entry points KPROCOFF, APLKSSR

Description: Part of the CICS/VS executor shared storage manager interface. Retracts all variables when a perproc entry in the processor table is marked for deletion.

Calls: Entry point KRETSUB

Exit: Returns

KCLEAR

Module: APLKLIBU

Called By: Entry point APLFXIIM

Description: Part of the library management services provided by the CICS/VS executor. Processes the user's)CLEAR request.

Calls: Macro APLKEXIT. CICS/VS macro DFHSC (GETMAIN, FREEMAIN)

Exit: Returns

KCMBL

Module: APLKMSCB

Called By: Entry point APLFXIIM

Description: Part of the interpreter interface provided by the CICS/VS executor. Indicates that the YYMBL service is not supported.

Exit: Returns

KCOPA

Module: APLKLIBU

Called By: Entry point APLFXIIM

Description: Part of the library management services provided by the CICS/VS executor. Processes the user's)COPY and)PCOPY requests.

Calls: Entry points APLKLIBR, APLKSPEN, APLAPASS. Macros APLKG (LIBSERV, TYPE=LOAD), APLKHIST, APLKWAIT. CICS/VS/VS macro DFHSC (GETMAIN), DFHFC (GET, RELEASE)

Exit: Returns

KCOPI

Module: APLKLIBU

Called By: Entry point APLFXIIM

Description: Part of the library management services provided by the CICS/VS executor. Assists in processing of the user's)COPY and)PCOPY requests by moving data into the sink workspace.

Exit: Returns

KCOPO

Module: APLKLIBU

Called By: Entry point APLFXIIM

Description: Part of the library management services provided by the CICS/VS executor. Assists in processing of the user's)COPY and)PCOPY requests by accepting data objects from the source workspace.

Exit: Returns

KCOPZ

Module: APLKLIBU

Called By: Entry point APLFXIIM

Description: Part of the library management services provided by the CICS/VS executor. Gains control during processing of a user's)COPY or)PCOPY request when either the source or sink workspace has no more data to provide or copy. When the terminating YY code YYCOPZ is entered for the source workspace, the address space of the source workspace is returned to CICS/VS.

Calls: Macro APLKHIST. CICS/VS macro DFHSC (FREEMAIN)

Exit: Returns

KCQAI

Module: APLKMSCA

Called By: Entry point APLFXIIM

Description: Part of the interpreter interface provided by the CICS/VS executor. Executes the YYQAI service request (a request for terminal time information).

Calls: Macro APLKHIST (CALC)

Exit: Returns

KCQUOTA

Module: APLKMSCB

Called By: Entry point APLFXIIM

Description: Part of the interpreter interface provided by the CICS/VS executor. Executes the YYQUOTA service request (a request for user quota information).

Exit: Returns

KCQZ

Module: APLKIFIX

Called By: Entry point APLFXIIM

Description: Part of the interpreter interface provided by the CICS/VS executor. Executes the YYQZ service request (a request for quantum end handling).

Calls: KRSTEX. Macros APLKHIST, DFHKC (CHAP, WAIT)

Exit: Returns

KCSYSER

Module: APLKMSCA

Called By: Entry point APLFXIIM

Description: Part of the interpreter interface provided by the CICS/VS executor. Executes the YYSYSER service request (a request to write the system error message).

Calls: Macros APLKEXIT, APLKT (TRAN, G). CICS/VS macro DFHDC (PARTIAL)

Exit: Returns

KCTABS

Module: APLKMSCB

Called By: Entry point APLFXIIM

Description: Part of the interpreter interface provided by the CICS/VS executor. Executes the YYTABS service request (a request to set or retrieve previously set tab settings).

Exit: Returns

KCTIME

Module: APLKMSCA

Called By: Entry point APLKMSCA

Description: Part of the interpreter interface provided by the CICS/VS executor. Executes the YYTIME service request (a request for the time of day).

Exit: Returns

KCTRAN

Module: APLKMSCB

Called By: Entry point APLFXIIM

Description: Part of the interpreter interface provided by the CICS/VS executor. Indicates that the YYTRAN service is not supported.

Exit: Returns

KCWIDTH

Module: APLKMSCB

Called By: Entry point APLFXIIM

Description: Part of the interpreter interface provided by the CICS/VS executor. Executes the YYWIDTH service request (a request to set the width of output to the terminal).

Exit: Returns

KDPCREG

Module: APLKDOPS or APLKVOPS

Called By: Entry point ABEXIT

Description: The operating system dependent interface provided as part of the CICS/VS executor. Gets program check registers from CICS/VS control blocks.

Exit: Returns

KDPFAB

Module: APLKVOPS

Called By: APLKAGBL

Description: Part of the CICS/VS executor for OS/VS systems. Takes no action in this environment.

Exit: Returns

KDPFAP

Module: APLKDOPS

Called By: Entry point APLKAGBL

Description: The operating system dependent interface provided as part of the CICS/VS executor. For DOS only, removes the page fix exit, and restores the CICS/VS timer.

Calls: DDSNC. Macros PFIX, SETPFA, STXIT, SETIME, APL macros, APLKTRCE

Exit: Returns

KDROP

Module: APLKLIBU

Called By: Entry point APLFXIIM

Description: Part of the library management services provided by the CICS/VS executor. Processes the user's)DROP requests.

Calls: Entry point APLKLIBG (through use of the APLKG macro). Macros APLKG (LIBSERV, TYPE=DROP), APLKWAIT

Exit: Returns

KFREESP

Module: APLKSSUB

Called By: Entry points APLKSSR, KRETSUB

Description: Part of the CICS/VS executor shared storage manager interface. Marks a given area in shared storage free for use.

Exit: Returns

KGCFILE

Module: APLKLIBV

Called By: Entry point APLKLIBG (by APLKLIBF through APLKG macro)

Description: Part of the library management services provided by the CICS/VS executor. Provides the global library service of creating a file in the APL library.

Calls: Entry points KLALLOC, APLKLIBR, KLDEALOC, KLPUT. CICS/VS macro DFHFC (GET, PUT, RELEASE, GETAREA)

Exit: Returns

KGCOL

Module: APLKSSUB

Called By: Entry points APLKSSR, KGETSPAC

Description: Part of the CICS/VS executor shared storage manager interface. Cleans up shared memory by packing it.

Exit: Returns

KGDFILE

Module: APLKLIBV

Called By: Entry point APLKLIBG (by APLKLIBF through APLKG macro)

Description: Part of the library management services provided by the CICS/VS executor. Provides the global library service of deleting auxiliary processor 121 files.

Calls: Entry points KLDEALOC, KLPUT. CICS/VS macros DFNFC (GET, PUT, DELETE), DFHSP (USER)

Exit: Returns

KGDROP

Module: APLKLIBG

Called By: APLKLIBG

Description: This provides a workspace drop service for KDROP.

Calls: KLRDBITM, KLPUT, KLDEALPC. Macros (CICS/VS) DFHFC (GET, PUT, and RELEASE)

Exit: Returns

KGETSPAC

Module: APLKSSUB

Called By: Entry point APLKSSR

Description: Part of the CICS/VS executor shared storage manager interface. Gets space in shared storage for the value or name of a shared variable.

Calls: Entry point KGCOL

Exit: Returns

KGLOAD

Module: APLKLIBG

Called By: APLKLIBG

Description: This provides a workspace load service for KLOAD.

Calls: KLGET. Macros (CICS/VS) DFHFC (GET)

Exit: Returns

KGSAVE

Module: APLKLIBG

Called By: APLKLIBG

Description: This provides a workspace service for KSAVE.

Calls: KLRDBITM, KLALLOC, KLPUT, KLDEALOC. Macros (CICS/VS) DFHFC (GET, PUT, and GETAREA)

Exit: Returns

KGUDIR

Module: APLKLIBG

Called By: APLKLIBG

Description: This updates directory records.

Calls: GRELPRM2 exit routine defined by the caller of APLKLIBG. CICS/VS macros DFHFC (GET, PUT)

Exit: Returns

KGUFILE

Module: APLKLIBV

Called By: Entry point APLKLIBG (by APLKLIBF through APLKG macro)

Description: Part of the library management services provided by the CICS/VS executor. Provides the global library service of file extend support for auxiliary processor AP121 files.

Calls: Entry points KLALLOC, KLDEALOC, KLPUT. Macro APLKEXIT. CICS/VS macros DFHFC (GET, PUT, RELEASE), DFHSC (GETMAIN, FREEMAIN)

Exit: Returns

KGWDIR

Module: APLKLIBG

Called By: APLKLIBG

Description: This writes a directory record.

Calls: Macros (CICS/VS) DFHC (GETAREA,PUT)

Exit: Returns

KGWLIB

Module: APLKLIBV

Called By: Entry point APLKLIBG (by APLKLIBF through APLKG macro)

Description: Part of the library management services provided by the CICS/VS executor. Provides the global library service of writing a control interval to the APL library.

Calls: Entry point KLPUT. CICS/VS macro DFHFC (GET. PUT, RELEASE)

Exit: Returns

KIDSETUP

Module: APLKSSUB

Called By: Entry point APLKSSR

Description: Part of the CICS/VS executor shared storage manager interface. Sets up a doubleword ID for the two shared partners.

Exit: Returns

KIFCNEXT

Module: APLKIFIX

Called By: Entry point KABEXIT

Description: Part of the interpreter interface provided by the CICS/VS executor. Handles abnormal conditions occurring in the user transaction.

Calls: Entry points KPGMCHK, KRSTEX. Macro APLKEXIT. CICS/VS macro DFHDC

Exit: Caller, KTOINTERP, KIQUEND, APLKEXIIM

KINIEX

Module: APLKDOPS

Called By: Entry point APLKAGBL

Description: The operating system dependent interface provided as part of the CICS/VS executor. Sets up a page fault exit, and replaces the CICS/VS timer with an APL timer.

Exit: Returns

KINIEX

Module: APLKVOPS

Called By: Entry point APLKAGBL

Description: Part of the CICS/VS executor for OS/VS systems. Takes no action in this environment.

Exit: Returns

KLALLOC

Module: APLKLIBA

Called By: Entry points KGSAVE, KGCFILE, KGUFILE

Description: Part of the library management services provided by the CICS/VS executor. Allocates the requested number of control intervals from the free space bit maps that describe the allocation status of the library.

Exit: Returns

KLCLOS

Module: APLKVOPS

Called By: Entry point APLKLIBT

Description: The operating system dependent interface provided as part of the CICS/VS executor. Issues VSAM CLOSE requests against the APL library.

Calls: CICS/VS macros DFHFC (LOCATE), DFHTR (USER). Operating system macro OPEN (VSAM)

Exit: Returns

KLDEALOC

Module: APLKLIBA

Called By: Entry points KGSAVE, KGDFILE, KGCFILE, KGUFILE, KGDROP

Description: Part of the library management services provided by the CICS/VS executor. Deallocates the workspaces allocated to CICS/VS from the free space bit maps.

Exit: Returns

KLGET

Module: APLKVOPS, APLKDOPS

Called By: Entry points KGLOAD, APLKLIBI, KGRLIB

Description: The operating system dependent interface provided as part of the CICS/VS executor. Issues VSAM GET requests against the APL library.

Calls: CICS/VS macros DFHTR (USER), DFHKC (WAIT). Operating system macros GET (VSAM), CHECK (VSAM—issued by APLKVOPS only)

Exit: Returns

KLIB

Module: APLKLIBU

Called By: Entry point APLFXIIM

Description: Part of the library management services provided by the CICS/VS executor. Processes the user's)LIB requests.

Calls: CICS/VS macro DFHFC (GETNEXT, ESETL, SETL)

Exit: Returns

KLOAD

Module: APLKLIBU

Called By: Entry point APLFXIIM

Description: Part of the library management services provided by the CICS/VS executor. Processes the interpreter)LOAD request.

Calls: Entry points APLKLIBR, APLKSPEN, APLAPASS. Macros APLKG (LIBSERV, TYPE=LOAD), APLKWAIT, APLKHIST, APLKEXIT. CICS/VS macros DFHFC (RELEASE), DFHSC (GETMAIN (CLASSPROGRAM), FREEMAIN)

Exit: Returns

KLOPEN

Module: APLKVOPS

Called By: Entry point APLKLIBI

Description: The operating system dependent interface provided as part of the CICS/VS executor. Issues VSAM OPEN requests against the APL library.

Calls: CICS/VS macros DFHFC (LOCATE), DFHTR (USER) Operating system macro OPEN (VSAM)

Exit: Returns

KLPUT

Module: APLKVOPS

Called By: Entry points APLKLIBG, KGWLIB, KGCFILE, KGDFILE, KGUFILE

Description: The operating system dependent interface provided as part of the CICS/VS executor. Issues VSAM PUT requests for APL library updates.

Calls: CICS/VS macros DFHKC (WAIT), DFHTR (USER) Operating system macros PUT (VSAM), CHECK (VAM—issued by APLKVOPS only)

Exit: Returns

KLRDBITM

Module: APLKLIBV

Called By: APLKLIBV

Description: This reads the allocation bit map into storage.

Calls: KLGET

Exit: Returns

KMACRO

IModule: APLKADSP

Called By: APLKADSP

Description: Part of the control of the user session task performed by the CICS/VS executor. Dispatches processes within the user task.

Calls: APLKIFON, APLXGKON, APLACRCP, any IBM or user-writtem dependent auxiliary processor. Macros APLKSON, APLKSOF. CICS/VS macros DFHTR, DFHSC (GETMAIN, FREEMAIN), DFHKC (WAIT)

Exit: Returns

KPASS

Module: APLKLIBU

Called By: Entry point APLFXIIM

Description: Part of the library management services provided by the CICS/VS executor. Processes the user's request to change the sign on password.

Calls: APLAPASS, APLKSPEN

Exit: Returns

KPGMCHK

Module: APLKIFIX

Called By: Entry point KIFONEXT

Description: Part of the interpreter interface provided by the CICS/VS executor. Handles program checks occurring in the interpreter.

Exit: Returns

KPOSTWAI

Module: APLKSSUB

Called By: Entry point APLKSSR

Description: Part of the CICS/VS executor shared storage manager interface. Posts all ECBs that are waiting on space.

Calls: Entry point KSINGAL

Exit: Returns

KPPSEARC

Module: APLKSSUB

Called By: Entry point APLKSSR

Description: Part of the CICS/VS executor shared storage manager interface. Searches perproc entries in the processor table for a given user.

Exit: Returns

KPROCOFF

Module: APLKSSUB

Called By: Entry point APLKSSR

Description: Part of the CICS/VS executor shared storage manager interface. Signs off all processors with the same specified external ID.

Calls: Entry point KCLEANUP

Exit: Returns

KRETSUB

Module: APLKSSUB

Called By: Entry points APLKSSR, KCLEANUP

Description: Part of the CICS/VS executor shared storage manager interface. Retracts a shared variable.

Calls: Entry point KFREESP

Exit: Returns

KRSTEX

Module: APLKDOPS

Called By: APLFXIIM, KIFONEXT, KIQUEND, KASUSPND

Description: Part of the CICS/VS executor for DOS/VS systems. Cancels the APL time-slice timer and restores the CICS/VS times previously cancelled by the KSETEX routine.

Calls: Macros TTIMER, SETIME

Exit: Returns

KRSTEX

Module: APLKVOPS

Called By: Entry points APLFXIIM, KIFONEXT, KIQUEND

Description: The operating system dependent interface provided as part of the CICS/VS executor. Restores the system's timer exit when the interpreter is not in control.

Calls: Macros STIMER, TTIMER

Exit: Returns

KSAVE

Module: APLKLIBU

Called By: Entry point APLFXIIM

Description: Part of the library management services provided by the CICS/VS executor. Processes the user's)SAVE or)CONTINUE requests.

Calls: Entry points APLKSPEN, APLAPASS. Macros APLKG (LIBSERV, TYPE=SAVE), APLKWAIT, APLKHIST

Exit: Returns

KSEIZE

Module: APLKSSUB

Called By: Entry point APLKSSR

Description: Part of the CICS/VS executor shared storage manager interface. Gets control of a shared variable.

Exit: Returns

KSETEX

Module: APLKDOPS

Called By: KIQUEND, KTOINTER, KASUSPND

Description: Part of the CICS/VS executor for DOS/VS systems. Sets up the APL time-slice timer.

Calls: Macros SETIME, TTIMER

Exit: Returns

KSETEX

Module: APLKVOPS

Called By: Entry point KIQUEND, KTOINTER

Description: The operating system dependent interface provided as part of the CICS/VS executor. Sets up the timer exit for the interpreter and the self-contained yycode service routine.

Calls: Macros TTIMER, STIMER

Exit: Return

KSINGAL

Module: APLKSSUB

Called By: Entry points APLKSSR, KPOSTWAI

Description: Part of the CICS/VS executor shared storage manager interface. Posts a given ECB.

Exit: Returns

KSPALD

Module: APLKVALD

Called By: Entry points KSPCPY, KSPIMP

Description: Part of the APL library service program for CICS. Allocates and deallocates control intervals in 4K block from the APL library data set by turning bits in the freespace descriptor map on and off.

Exit: Returns

KSPAUT

Module: APLKVAUT

Called By: Entry point APLKSPRG

Description: Part of the APL library service program for CICS. Processes the AUTH control statement and checks the validity of the user by reading his profile from the APL directory (for user level authorization) or by accessing the directory update password in module APLKPASS (for library level authorization).

Calls: Entry points KSPMSG, KSPLBI

Exit: Returns

KSPCMD

Module: APLKVCMD

Called By: Entry point APLKSPRG

Description: Part of the APL library service program for CICS. Calls entry point KSPSCN to scan the command collected by module APLKSPRG (entry point APLKVEXC) and convert it to a set of parameters in SPGPARMA (part of the service program global table). This part of the module also contains the syntax tables for the service program control statements.

Calls: Entry points KSPMSG, KSPSCN

Exit: Returns

KSPCPY

Module: APLKVCPY

Called By: Entry point APLKSPRG

Description: Implements the COPY command for the APL library service program for CICS/VS. Moves a range of file names from the ownership set of each user of an input data set to an output data set. Either of these data sets may be the APL library and directory. Also processes the tape written by the conversion utility.

Calls: Entry points KSPLBI, KSPLBO, KSPMSG, KSPTPO, KSPALLOC, KSPDEALC

Exit: Returns

KSPDOS

Module: APLKDDOS (DOS only)

Called By: Entry points KSPDSO, KSPINT, APLKSPRG, KSPDSI, KSPLBI, KSPMSG, KSPTRM, KSPTPO

Description: Part of the APL library service program for CICS/VS. Simulates OS/VS QSAM support in the DOS/VS environment. Processes the ENVIRONMENT command.

Calls: DOS modules IJFSZZWZ, IJFUZZZZ, IJGUIZZZ, IJGUOZZZ, IJGQOZZZ; VSAM modules IKQVLAB, IKQVDTPE. Macros GETVIS, FREEVIS, WAIT, CDLOAD, CNTRL, OPEN, CLOSE, BTWAIT, EXCP

Exit: Returns (If end-of-file condition, returns to caller's EODAD exit.)

KSPDSI

Module: APLKVDSI

Called By: Entry point KSPIMP

Description: Part of the APL library service program for CICS/VS. Reads records from the APL input data set for KSPIMP to process.

Calls: Entry point KSPDOS (DOS only). Macro GET (QSAM, OS only).

Exit: Returns

KSPDSO

Module: APLKVDSO

Called By: Entry point KSPEXp

Description: Part of the APL library service program for CICS/VS. Writes records from the APL library to a batch data set.

Calls: Entry points KSPMSG, KSPDOS (DOS only). Macro PUT (QSAM, OS only)

Exit: Returns

KSPEXP

Module: APLKVEXP

Called By: Entry point APLKSPRG

Description: Part of the APL library service program for CICS/VS. Processes the EXPORT command. Checks authoization and converts workspaces and data files in the APL library to a format acceptable for import to VS APL under other supported environments.

Calls: Entry points KSPDSO, KSPLBI, KSPMSG.

Exit: Returns

KSPFMT

Module: APLKVFMT

Called By: Entry point APLKSPRG

Description: Part of the APL library service program for CICS/VS. Implements the format command. Formats the library into 4K blocks. Writes the free space profile record in the APL directory. If USERS is specified on the FORMAT command, writes directory entries for libraries 1, 2, and 314159.

Calls: Entry point KSPMSG. Macros OPEN, CLOSE, PUT, GET

Exit: Returns

KSPIMP

Module: APLKVIMP

Called By: Entry point APLKSPRG

Description: Part of the APL library service program for CICS/VS. Processes IMPORT control statements. Converts exported APL files and workspaces to the format of the APL library, calling entry point KSPALLOC to allocate space, entry point KSPDSI to read the batch data set, and entry point KSPLBO to write the files and workspaces to the library.

Calls: Entry points KSPLBO, KSPMSG, KSPDSI, KSPLBI, KSPALLOC, KSPDEALC

Exit: Returns

KSPINT

Module: APLKVINT

Called By: Entry point APLKSPRG

Description: Part of the APL library service program for CICS/VS. Performs initialization for each command and ensures that data sets required by the command being processed are open. Opens the APL library and directory and checks the validity of the library data set to ensure that it belongs to the APL directory data set.

Calls: Entry points KSPDOS (DOS only), KSPMSG. Macros OPEN, GET, GETMAIN, SHOWCB, PUT, CLOSE, RDJFCB (OS only); OPEN, GET, GETVIS, SHOWCB, PUT, CLOSE, VERIFY (DOS only)

Exit: Returns

KSPLBI

Module: APLKVLBI

Called By: Entry points KSPAUT, KSPCPY, KSPEXP, KSPIMP

Description: Part of the APL library service program for CICS/VS. Retrieves records from either an APL backup/archive tape or APL library for the processors of the AUTH, EXPORT, and COPY control statements.

Calls: Entry points KSPDOS (DOS only) Macros GET, PUT

Exit: Returns

KSPLBO

Module: APLKVLBO

Called By: Entry points KSPCPY, KSPIMP

Description: Part of the APL library service program for CICS/VS. Inserts or updates profiles, directory entries, and data sets in the APL library.

Calls: Entry point KSPMSG. Macros GET, PUT, ENDREQ

Exit: Returns

KSPMSG

Module: APLKVMSG

Called By: Entry points KSPCPY, KSPDSO, KSPAUT, APLKSPRG, KSPEXP, KSPIMP, KSPINT, KSPLBO, KSPPIN, KSPTPO, KSPCMD, KSPFMT

Description: Part of the APL library service program for CICS/VS. Puts all output messages and data from the APL library service program modules on the SYSPRINT (for OS) or SYSLIST (for DOS) data set.

Calls: Entry point KSPDOS (DOS only). Macros PUT (OS only); GETIME and COMRG (DOS only)

Exit: Returns

KSPPIN

Module: APLKVPIN

Called By: Entry point APLKSPRG

Description: Part of the APL library service program for CICS/VS. Opens the OS data sets required to run the utility job step.

Calls: Entry point KSPMSG Macros OPEN, GET, SHOWCB (OS only); OPEN, GET, SHOWCB, VERIFY (DOS only)

Exit: Returns

KSPSCN

Module: APLKVCMD

Called By: KSPCMD

Description: Parses the command collected by module APLKSPRG (entry point APLKVEXC) according to tables in module APLKVCMD (entry point KSPCMD).

Exit: Returns

KSPTPO

Module: APLKVTPO

Called By: Entry point KSPCPY

Description: Part of the APL library service program for for CICS/VS. Used by the COPY command to create a backup or archive sequential data set from a APL library.

Calls: Entry points KSPMSG, KSPDOS (DOS only)

Exit: Returns

KSPTRM

Module: APLKVTRM

Called By: Entry point APLKSPRG

Description: Part of the APL library service program for CICS/VS. Closes OS data sets and releases temporary space.

Calls: Entry point KSPDOS (DOS only) Macros CLOSE, FREEPOOL (OS only)

Exit: Returns

KTIMEREX

Module: APLKDOPS

Called By: DOS/VS supervisor

Description: Part of the CICS/VS executor. Sets quantum end indicator for the interpreter.

Calls: Macro EXIT(IT)

Exit: To CICS/VS timer exit or returns via DOS/VS exit

KTIMEREX

Module: APLKVOPS

Called By: OS/VS supervisor

Description: The operating system dependent interface provided as part of the CICS/VS executor. Handles end-of-time-slice situations while the interpreter is executing. This is an exit routine that has been defined to OS/VS by the SETEX routine. Sets quantum end indicator for the interpreter.

Exit: Returns

KTOINTER

Module: APLKIFIX

Called By: Entry points APLKIFON, KIFONEXT

Description: Part of the interpreter interface provided by the CICS/VS executor. Sets up the timer exit and the registers to be used by the interpreter and then passes control to the interpreter.

Calls: Entry point KSETEX. Macro APLKTRCE

Exit: Interpreter (APLIINIT)

KTRAL

Module: APLKTRQO

Called By: Entry point KTRRT

Description: Part of the terminal management services provided by the CICS/VS executor. Handles APLKTERM macro TYPE=ALARM minor requests if the request is not specified in combination with other requests.

Calls: Macro APLKT (SCHED)

Exit: Returns

KTRCU

Module: APLKTRQO

Called By: Entry points KTRFA, KTRFM, KTRRT, KTRWR. Macro APLKT (SETCUR)

Description: Part of the terminal management services provided by the CICS/VS executor. Handles APLKTERM macro TYPE=SETCUR minor requests whether the request is specified alone or in combination with other requests.

Calls: Macro APLKT (SCHED, FINDF)

Exit: Returns

KTRFA

Module: APLKTRQO

Called By: Macro APLKTERM (FLDATTR)

Description: Part of the terminal management services provided by the CICS/VS executor. Handles APLKTERM macro TYPE=FLDATTR major requests. Minor TYPE= requests of SETCUR and ALARM may be combined with this request.

Calls: Entry point KTRCU. Macros APLCTRCE, APLKT (SCHED).

Exit: Returns

KTRFI

Module: APLKTREQ

Called By: Macro APLKTERM (Final)

Description: Part of the terminal management services provided by the CICS/VS executor. Handles APLKTERM macro TYPE=FINAL major requests. No minor requests may be combined with this request. Clears PTK pointers and resets PTK flags to indicate that the specified screen interface is inactive (the standard APL screen interface or, if the APLKTERM macro parameter OPT=ALT is specified, the alternate interface). Frees storage associated with the interface. If the interface associated KTSCHED entry point is called to communicate with the terminal transaction if necessary.

Calls: Macros APLKTRCE, APLKT (SCHED) CICS/VS macros DFHSC (FREEMAIN), DFHIC (INITIATE from entry point APLKTCTL)

Exit: Returns

KTRFM

Module: APLKTRQO

Called By: Macro APLKTERM (FORMAT)

Description: Part of the terminal management services provided by the CICS/VS executor. Handles APLKTERM macro TYPE=FORMAT major requests. Minor TYPE= requests of ALARM and SETCUR may be combined with this request. Format requests may be for a full format or, if the OPT=REFORM parameter is specified on the APLKTERM macro, a partial reformat.

Calls: Entry point KTRCU. Macros APLKTRCE, APLKT (FCHECK, SCHED, CLEAR). CICS/VS macro DFHSC (GETMAIN, FREEMAIN)

Exit: Returns

KTRGD

Module: APLKTREQ

Called By: Macro APLKTERM (GETDATA)

Description: Part of the terminal management services provided by the CICS/VS executor. Handles APLKTERM macro TYPE=GETDATA major requests. Returns data from specified fields unless OPT=NOTDATA is specified on the APLKTERM macro, in which case the length of the field is returned for each field specified.

Calls: Macros APLKTRCE, APLKT (CLEAR, TRAN)

Exit: Returns

KTRGF

Module: APLKTREQ

Called By: Macro APLKTERM (GETFORM)

Description: Part of the terminal management services provided by the CICS/VS executor. Handles APLKTERM macro TYPE=GETFORM major requests. No minor requests may be combined with this request. Returns a description of the current screen format to the caller.

Calls: Macros APLKTRCE, APLKT (CLEAR)

Exit: Returns

KTRHC

Module: APLKTRQO

Called By: Entry point KTRRT or KTRWR if APLKTERM macro specifies TYPE=HCOPY

Description: Part of the terminal management services provided by the CICS/VS executor. Handles APLKTERM macro TYPE=HCOPY requests whether the request is specified alone or in combination with a TYPE=WRITE request.

Calls: Entry point APLKEMGR. Macros APLKT (SCHED, CLEAR, HLINE)

Exit: Returns

KTRIN

Module: APLKTREQ

Called By: Macro APLKTERM (INIT)

Description: Part of the terminal management services provided by the CICS/VS executor. Handles APLKTERM macro TYPE=INIT major requests. No minor requests may be combined with this request. Gets storage for and initializes control blocks and buffers for the standard APL screen interface or, if the OPT=ALT parameter was specified on the APLKTERM macro, for the alternate APL screen interface. Provides the default screen format for the specified screen interface.

Calls: Macro APLKTRCE. CICS/VS macros DFHSC (GETMAIN)

Exit: Returns

KTRRD

Module: APLKTREQ

Called By: Macro APLKTERM (READ)

Description: Part of the terminal management services provided by the CICS/VS executor. Handles APLKTERM macro TYPE=READ major requests. Minor TYPE= requests of ALARM, SETCUR, and RESTORE may be combined with this request. Uses the KTSCHED routine to shcedule terminal read requests. When the read operation is completed, returns a description of the input to the caller.

Calls: Macros APLKTRCE, APLKT (SCHED, CLEAR, SETCUR, FINDF)

Exit: Returns

KTRRS

Module: APLKTRQO

Called By: Entry point KTRRT

Description: Part of the terminal management services provided by the CICS/VS executor. Records APLKTERM macro TYPE=RESTORE minor requests if the request is not specified in combination with other requests.

Calls: Macro APLKT (SCHED)

Exit: Returns

KTRRT

Module: APLKTREQ

Called By: Macro APLKTERM (ALARM, SETCUR, HCOPY, or RESTORE) when APLKTERM is not used in combination with a major request

Description: Part of the terminal management services provided by the CICS/VS executor. A routing routine that calls the appropriate terminal manager routines when the APLKTERM macro is used without specifying a major request type. All routing is done using the APLKTREQ address table.

Calls: Entry points KTRAL, KTRCU, KTRHC, KTRRS. Macro APLKTRCE

Exit: Returns

KTRTRAN

Module: APLKTRAN

Called By: Entry point APLKEHCP. Macro APLKT (TRAN)

Description: Part of the terminal management services provided by the CICS/VS executor. Also linked to as part of hard copy print transactions. Translates data and, optionally, moves the data to wherever the caller specifies.

Exit: Returns

KTRWR

Module: APLKTRQO

Called By: Macro APLKTERM (WRITE)

Description: Part of the terminal management services provided by the CICS/VS executor. Handles APLKTERM macro TYPE=WRITE major requests. Minor TYPE= requests of SETCUR, ALARM, HCOPY, and RESTORE may be combined with this request. Causes a write operation to be shceduled but, if the OPT=WAIT parameter on the APLKTERM macro is omitted, may return to the caller before the write operation has completed.

Calls: Entry points KTRCU, KTRHC. Macros APLKTRCE, APLKT (SCHED, CLEAR, TRAN)

Exit: Returns

KTSCHED

Module: APLKTSRV

Called By: Macro APLKT (SCHED)

Description: Part of the terminal management services provided by the CICS/VS executor. Synchronizes requests. If required, schedules terminal transaction. May wait for completion of request.

Calls: Macros APLKWAIT, APLKPOST. CICS/VS Macro DFHPC (ABEND, DUMP, INITIATE for entry point APLKTCTL)

Exit: Returns or DFHPC (ABEND)

KTSCLEAR

Module: APLKTSRV

Called By: Macro APLKT (CLEAR)

Description: Part of the terminal management services provided by the CICS/VS executor. Clears the logical screen buffer and resets data lengths.

Exit: Returns

KTSFCHK

Module: APLKTSRV

Called By: Macro APLKT (FCHECK)

Description: Part of the terminal management services provided by the CICS/VS executor. Performs a validity check of the screen format.

Exit: Returns

KTSFNDF

Module: APLKTSRV

Called By: Macro APLKT (FINDF)

Description Part of the terminal management services provided by the CICS/VS executor. Given a row and column address, finds the associated field.

Exit: Returns

KTSLINO

Module: APLKTSRV

Called By: Macro APLKT (HLINE)

Description: Part of the terminal management services provided by the CICS/VS executor. Prepares a line of full screen copy.

Calls: Entry point APLKEMGR

Exit: Returns

KTSLOCID

Module: APLKTSRV

Called By: Macro APLKT (LOCID)

Description: Part of the terminal management services provided by the CICS/VS executor. Locates the TSF (terminal screen status) for a specified field ID if the field is defined.

Exit: Returns

KTSLOCR

Module: APLKTSRV

Called By: Macro APLKT (LOCREQ)

Description: Part of the terminal management services provided by the CICS/VS executor. Executing under the terminal transaction, locates the next request to be processed.

Exit: Returns

KWSID

Module: APLKLIBU

Called By: Entry point APLFXIIM

Description: Part of the library management services provided by the CICS/VS executor. Processes the user's)WSID requests.

Calls: APLAPASS

Exit: Returns

KYYTYOI

Module: APLKIFIX

Called By: APLKXIIM

Description: Provides an terface to the session manager for the terminal I/O requests (YYTYO, YYTYI, YYTYOI) made by the interpreter code.

Calls: APLKSPEN, APLATYO, APLATYI, APLATYOI

Exit: Returns

KYYOFF

Module: APLKMSCB

Called By: Entry points APLFXIIM, ABEXIT

Description: Part of the interpreter interface provided by the CICS/VS executor. Executes YYOFF service requests (a request to terminate the session).

Calls: Entry points APLATERM, APLKLUTM. Macros APLKEXIT, APLKHIST, APLKG (LIBSERV).

Exit: Returns

OFF121X

Module: APL121

Called By: APLXAC

Description: This is the offer exit routine.

Calls: Macros APLXAEAT, APLXCAPS

Exit: Returns

PCATOFF

Module: APLPMISC

Called By: APLFXIIM

Description: Executes YYATOFF service request (VSPC).

Exit: Normal; ERSAVEAR (Error)

PCCLEAR

Module: APLPLIBS

Called By: APLFXIIM

Description: Executes YYCLEAR service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCCMD

Module: APLPMISC

Called By: APLFXIIM

Description: Executes YYCMD service request for the VSPC executor (issues the VSPC service request WCMD).

Calls: ERMSGRTN. Macro ASUSRQ

Exit: Returns; ERSAVEAR (Error)

PCCOPA

Module: APLPLIBS

Called By: APLFXIIM

Description: Executes YYCOPA service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCCOPI

Module: APLPMISC

Called By: APLFXIIM

Description: Executes YYCOPI service request (VSPC).

Exit: Returns; ERSAVEAR (Error)

PCCOPO

Module: APLPMISC

Called By: APLFXIIM

Description: Executes YYCOPO service request (VSPC).

Exit: Returns; ERSAVEAR (Error)

PCCOPZ

Module: APLPMISC

Called By: APLFXIIM

Description: Executes YYCOPZ service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCDELAY

Module: APLPTYIO

Called By: APLFXIIM

Description: Executes YYDELAY service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCDROP

Module: APLPLIBS

Called By: APLFXIIM

Description: Executes YYDROP service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCDTYI

Module: APLPTYIO

Called By: APLFXIIM, PCDTYOI

Description: Executes YYTYI service request for display terminal (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCDTYO

Module: APLPTYIO

Called By: APLFXIIM

Description: Executes YYTYO service request for display terminal (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCDTYOI

Module: APLPTYIO

Called By: APLFXIIM

Description: Executes YYTYOI service request for display terminal (VSPC).

Calls: PCDTYI

Exit: Returns; ERSAVEAR (Error)

PCDUMP

Module: APLPSERR

Called By: APLFXIIM

Description: Executes YYDUMP service request (VSPC).

Exit: Returns

PCLIB

Module: APLPLIBS

Called By: APLFXIIM

Description: Executes YYLIB service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCLOAD

Module: APLPLIBS

Called By: APLFXIIM

Description: Executes YYLOAD service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCMBL

Module: APLPTYIO

Called By: APLFXIIM

Description: Executes YYMBL service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCOFF

Module: APLPMISC

Called By: APLFXIIM

Description: Executes YYOFF service request (VSPC).

Exit: Returns to VSPC

PCPASS

Module: APLPMISC

Called By: APLFXIIM

Description: Executes YYPASS service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCQAI

Module: APLPMISC

Called By: APLFXIIM

Description: Executes YYQAI service request (VSPC).

Exit: Returns; ERSAVEAR (Error)

PCQUOTA

Module: APLPMISC

Called By: APLFXIIM

Description: Executes YYQUOTA service request (VSPC).

Exit: Returns; ERSAVEAR (Error)

PCQZ

Module: APLPMISC

Called By: APLFXIIM

Description: Executes YYQZ service request (VSPC).

Exit: Returns; ERSAVEAR (Error)

PCRWAIT

Module: APLPTYIO

Called By: APLFXIIM

Description: Executes YYRWAIT service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCSACC

Module: APLPSHVR

Called By: APLFXIIM

Description: Executes YYSACC service request (VSPC).

Calls: APLPAPAC, ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCSAVE

Module: APLPLIBS

Called By: APLFXIIM

Description: Executes YYSAVE service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCSCOPY

Module: APLPSHVR

Called By: APLFXIIM

Description: Executes YYSCOPY service request (VSPC).

Calls: APLPAPPR, ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCSOFF

Module: APLPSHVR

Called By: APLFXIIM

Description: Executes YYSOFF service request (VSPC).

Calls: APLPAPSF, ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCSOFFER

Module: APLPSHVR

Called By: APLFXIIM Descrption: Executes YYSOFFER service request (VSPC).

Calls: APLPAPOF, ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCSON

Module: APLPSHVR

Called By: APLFXIIM

Description: Executes YYSON service request (VSPC).

Calls: ERMSGRTN

Exit Returns; ERSAVEAR (Error)

PCSQUERY

Module: APLPSHVR

Called By: APLFXIIM

Description: Executes YYSQUERY service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCSREF

Module: APLPSHVR

Called By: APLFXIIM

Description: Executes YYSREF service request (VSPC).

Calls: APLPAPPR, ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCSRET

Module: APLPSHVR

Called By: APLFXIIM

Description: Executes YYSRET service request (VSPC).

Calls: APLPAPRT, ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCSSPEC

Module: APLPSHVR

Called By: APLFXIIM

Description: Executes YYSSPEC service request (VSPC).

Calls: APLPAPPR, ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCSYSER

Module: APLPSERR

Called By: APLFXIIM

Description: Executes YYSYSER service request (VSPC).

Calls: ERTIMDAT, ERMSGRTN

Exit: Returns; ERENDEX (Error)

PCTABS

Module: APLPTYIO

Called By: APLFXIIM

Description: Executes YYTABS service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCTIME

Module: APLPMISC

Called By: APLFXIIM

Description: Executes YYTIME service request (VSPC).

Exit: Returns; ERSAVEAR (Error)

PCTRAN

Module: APLPTYIO

Called By: APLFXIIM

Description: Executes YYTRAN service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCTYI

Module: APLPTYIO

Called By: APLFXIIM, PCTYOI

Description: Executes YYTYI service request for typewriter terminal (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCTYO

Module: APLPTYIO

Called By: APLFXIIM, PCTYOI

Description: Executes YYTYO service request for typewriter terminal (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PCTYOI

Module: APLPTYIO

Called By: APLFXIIM

Description: Executes YYTYOI service request for typewriter terminal (VSPC).

Calls: PCTYI, PCTYO

Exit: Returns; ERSAVEAR (Error)

PCWIDTH

Module: APLPTYIO

Called By: APLFXIIM

Description: Executes YYWIDTH service request (VSPC).

Exit: Returns; ERSAVEAR (Error)

PCWSID

Module: APLPLIBS

Called By: APLFXIIM

Description: Executes YYWSID service request (VSPC).

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR (Error)

PRDDIR

Module: APLPAPCD

Called By: APLPAPPR

Description: Does direct read from a VSPC file for internal auxiliary processor AP121 and AP122.

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR, ERENDEX (Error)

PRDSEQ

Module: APLPAPCD

Called By: APLPAPPR

Description: Does sequential read from a VSPC file for internal auxiliary processors AP121 and AP122.

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR, ERENDEX (Error)

PWRITE

Module: APLPAPCD

Called By: APLPAPPR

Description: Does sequential write and direct update to a VSPC file for internal auxiliary processors AP121 and AP122.

Calls: ERMSGRTN

Exit: Returns; ERSAVEAR, ERENDEX (Error)

RET121X

Module: APL121

Called By: APLXAC

Description: This is the retract exit procedure which issues a close for the FAB, freemains any CTLBUF or DATBUF storage areas, and discards the stack. The CTL and DAT variables are retracted.

Calls: Macro APLXSTAK

Exit: Returns

SCAPL

Module: APLSCFXI

Called By: SCSPIE, APL

Description: Establishes an entry point to APLFXIIM at startup or after program check (CMS).

Exit: APLFXIIM

SCATOFF

Module: APLSCTYP

Called By: APLFXIIM

Description: Executes YYATOFF service request (CMS).

Exit: Returns; SCSAVOFL(Error)

SCATOFF

Module: APLYUTYP

Called By: SCFXI via macro APLCCALL

Description: This entry point turns off attention and cancels output bits in PERTERM (TSO).

Calls: Macros APLDEFN, APLCENTR, APLCEXIT

Exit: Returns

SCATTN

Module: APLSCTYP

Called By: CMS, STAX Exit Routine

Description: Performs system functions in response to attention signal (CMS).

Exit: Returns to CMS.

SCATTN

Module: APLYUTYP

Called By: Host system via STAX edit routine by user

Description: This entry point provides supervisor support for attention (TSO).

Calls: APLYUTRM. Macros STATUS, APLDEFN, POST, ESTAE, TCLEARQ, APLYUPRG, APLPTRGT

Exit: Returns

SCCLEAR

Module: APLSCMSC

Called By: APLFXIIM

Description: Executes YYCLEAR service request (CMS).

Exit: Returns; SCSAVOFL

SCCLEAR

Module: APLYUMSC

Called By: APLYUFXI

Description: Executes the YYCLEAR service request (TSO). YYCLEAR resets the size of the active workspace to either a specified size (PDSSIZE) or a default size (CMSMAXWS). It also changes the ID of the active workspace to that of a clear workspace.

Exit: Returns; SCSAVOFL(Error)

SCCMD

Module: APLYUMSC

Called By: APLYUFXI

Description: This executes the YYCMD in TSO.

Calls: APLYUCMD

Exit: Returns

SCCMD

Module: APLSCMSC

Called By: APLSCFXI

Description: Executes the YYCMD service request in CMS.

Exit: Returns.

SCCOPA

Module: APLSCOPY

Called By: APLFXIIM

Description: Executes YYCOPA service request (CMS).

Calls: SCLOAD

Exit: Returns; SCSAVOFL(Error)

SCCOPI

Module: APLSCOPY

Called By: APLFXIIM

Description: Executes YYCOPI service request (CMS).

Exit: Returns; SCSAVOFL(Error)

SCCOPO

Module: APLSCOPY

Called By: APLFXIIM

Description: Executes YYCOPO service request (CMS).

Exit: Returns; SCSAVOFL(Error)

SCCOPZ

Module: APLSCOPY

Called By: APLFXIIM

Description: Executes YYCOPZ service request (CMS).

Exit: Returns; SCSAVOFL(Error)

SCDELAY

Module: APLSCMSC

Called By: APLFXIIM

Description: Executes YYDELAY service request (CMS).

Exit: Returns; SCSAVOFL(Error)

SCDELAY

Module: APLYUMSC

Called By: APLYUFXI

Description: Executes the YYDELAY service request (TSO). YYDELAY sets the timer for the time period given in WSMPARM1/2, and puts the virtual machine into an enabled wait state until either the timer goes off or the user signals attention.

Exit: Returns: SCSAVOFL(Error)

SCDPA2

Module: APLSCDPY

Called By: JCEXTINT on PA2

Description: Handles cancel-output signal for display terminal under CMS.

Exit: Returns to CMS

SCDROP

Module: APLSCLIB

Called By: APLFXIIM

Description: Executes YYDROP service request (CMS).

Calls: SCFID. Macro APLSFID

Exit: Returns; SCSAVOFL(Error)

SCDROP

Module: APLYULIB

Called By: APLYUFXI

Description: Executes the YYDROP service request (TSO). Drop processing involves the following steps:

- Allocate DISP=OLD to verify data set handling conditions.
- Deallocate DISP=KEEP since a protect cannot be issued while allocated.

- Issue PROTECT SVC to unprotect the data set.
- Issue PROTECT SVC (PURGE) to scratch the data set.
- Issue CATALOG SVC to uncatalog the data set.

Calls: SCSAVOFL (via macro APLCENTR)

Exit: Returns; SCSAVOFL(Error)

SCDTYI

Module: APLSCDPY

Called By: SCDTYOI, APLFXIIM

Description: Executes YYTYI service request for display terminal (CMS).

Exit: Returns; SCSAVOFL(Error)

SCDTYI

Module: APLYUDPY

Called By: APLFXIIM, SCDTYOI

Description: Executes the YYTYI service request for display terminal (TSO).

Exit: Returns; SCSAVOFL(Error)

SCDTYIO

Module: APLYUDPY

Called By: APLFXIIM

Description: Executes the YYTYOI service request for display terminal (TSO).

Calls: SCDTYO, SCDTYI, STCKPOP, SCSAVOFL

Exit: Returns; SCSAVOFL(Error)

SCDTYO

Module: APLSCDPY

Called By: SCDTYOI, APLFXIIM

Description: Executes YYTYO service request for display terminal (CMS).

Exit: Returns; SCSAVOFL(Error)

SCDTYO

Module: APLYUDPY

Called By: APLFXIIM, SCDTYOI

Description: Executes the YYTYO service request for display terminal (TSO).

Exit: Returns; SCSAVOFL(Error)

SCDTYOI

Module: APLSCDPY

Called By: APLFXIIM

Description: Executes YYTYOI service request for display terminal input (CMS).

Calls: SCDTYO, SCDTYI

Exit: Returns; SCSAVOFL(Error)

SCDUMP

Module: APLSCERR

Called By: APLFXIIM

Description: Executes the YYDUMP service request (CMS).

Exit: Returns; SCSAVOFL(Error)

SCDUMP

Module: APLYUERR

Called By: APLYUFXI

Description: Executes the YYDUMP service request (TSO).

Exit: Returns; SCSAVOFL(Error)

SCENDAPL

Module: APLSCINI

Called By: SCOFF

Description: Terminates VS APL under CMS.

Exit: To CMS or logoff from VM/370

SCENDAPL

Module: APLYUINI

Called By: APLYUMSC

Description: Terminates VS APL under TSO.

Exit: Returns to TSO.

SCEXTINV

Module: APLSCVI

Called By: CMS on an external interrupt

Description: This scans shared memory to look for an ICB describing the interrupt code, and calls the exit routine.

Calls: ICBADDR routine if interrupt is found

Exit: Returns to CMS after calling exit. If no exit is called, it goes to address in CHSEIOLD.

SCFID

Module: APLSCFID

Called By: APLSCLIB, APLXFSFL, APLSCMSC, APLSCINT

Description: Given a name and library number, this routine generates a CMS file identifier and accesses the CMS disk.

Calls: CP LINK and DETACH, CMS STATE, DMSLAD. Macros APLPATCH, APLDEFN, APLSFID, APLSOPT, FSCBD, FSTD, ADT, NUCON, FSSTATE, DIAG

Exit: Returns

SCLIB

Module: APLSCLIB

Called By: APLFXIIM

Description: Executes YYLIB service request (CMS).

Calls: APLSCFID. Macros APLSFID, APLSOPT

Exit: Returns; SCSAVOFL(Error)

SCLIB

Module: APLYULIB

Called By: APLYUFXI

Description: Executes the YYLIB service request (TSO). An internal routine, LIBDSN, is called to build an OS data set name for this library; an indicator will be returned if the library definition entry does not exist in the catalog. The higher level qualifiers are moved to the DSN buffer for a DAIR DAP14 parameter block, and DAIR is called to return all the qualifiers in CMSBUFF. These qualifiers are converted to Z-code and passed back to the interpreter in WSMBUFF. New lines are inserted according to the current value of PTHWIDTH.

Calls: IKJDAIR

Exit: Returns; SCSAVOFL(Error)

SCLOAD

Module: APLSCLIB

Called By: SCCOPA, APLFXIIM

Description: Executes YYLOAD service request (CMS).

Exit: Returns; SCSAVOFL(Error)

SCLOAD

Module: APLYULIB

Called By: APLYUFXI or APLYUCPY

Description: Executes the YYLOAD service request (TSO). An internal routine, LIBDSN, is invoked to build an OS data set name and validate access authority to the library. LIBALLOC is then called to allocate the workspace with DISP=SHR. A prototype DCB is copied to the global work area and opened for BSAM chained scheduled access. To ensure that the)LOAD request is successful, various checks are made to ensure workspace fit and validity of workspace security code. If the BLKSIZE is smaller than the size of WSMBUFF, that area will be used to temporarily hold the first block of the workspace. A buffer will, otherwise, be GETMAINed from free space and used to hold the block. If all validity checks are met, the data portion of the first block is copied to its proper location in the workspace. The remaining blocks are read in using chained scheduled BSAM. When the last block has been read, a check is made to determine if the workspace did not have a recorded ownership code. If so, it is assumed to have been recently imported from CMS or VSPC; if this is true and if the workspace is not from someone else's shareable library, it is closed, reopened for output, and the first block is rewritten with the correct ownership code. Finally, the data set is closed, freed by a call to LIBFREE, and, if a buffer has been obtained, it is FREEMAINed.

Calls: IKJDAIR

Exit: Returns

SCMBL

Module: APLSCMSG

Called By: APLFXIIM

Description: Executes YYMBL service request (CMS).

Exit: Returns

SCMBL

Module: APLYUMSG

Called By: SCFXI via macro APLCCALL when interpreter issues APLSVCC YYMBL

Description: This is the message blocking and unblocking routine. It executes STCOM requests to block and unblock messages (TSO). The ')MSG ON/OFF' function is provided by employing the 'STCOM YES/NO' macro to invoke TSO SVC 94.

Calls: Macros APLDEFN, APLCENTR, APLCEXIT, STCOM

Exit: Returns

SCMICRO

Module: APLSCINI

Called By: An entry card is included to determine the instruction address from the load map.

Description: Identifies an instruction to patch when it is desired to disable the VS APL microcode assist (TSO).

Exit: Not applicable

SCMICRO

Module: APLYUINI

Called By: An entry card is included to determine the instruction address from the load map.

Description: Identifies an instruction to patch when it is desired to disable the VS APL microcode assist (TSO).

Exit: Not applicable

SCOFF

Module: APLSCMSC

Called By: APLFXIIM

Description: Executes SCOFF service request (CMS).

Exit: SCENDAPL

SCOFF

Module: APLYUINI

Called By: APLYUFXI

Description: Executes the SCOFF service request (TSO).

Exit: SCENDAPL

SCPASS

Module: APLSCMSC

Called By: APLFXIIM

Description: Executes YYPASS service request in CMS.

Exit: Returns

SCPASS

Module: APLYUMSC

Called By: APLYUFXI

Description: Executes the YYPASS (change sign-on password) service request (TSO).

Exit: Returns

SCQAI

Module: APLSCMSC

Called By: APLFXIIM

Description: Executes YYQAI service request (CMS).

Exit: Returns

SCQAI

Module: APLYUMSC

Called By: APLYUFXI

Description: Executes the YYQAI service request (TSO). YYQAI supplies current values related to the distinguished variable QUAD-AI to the interpreter. These values comprise user's accumulated CPU usage, terminal connect time, keying time, and default account number (1001).

Exit: Returns

SCQUOTA

Module: APLSCMSC

Called By: APLFXIIM

Description: Executes YYQUOTA service request (CMS).

Exit: Returns

SCQUOTA

Module: APLYUMSC

Called By: APLYUFXI

Description: Executes the YYQUOTA service request (TSO). YYQUOTA puts information about quotas for library space, workspace size, and shared variable sizes into workspace fields so that the interpreter can print them.

Exit: Returns

SCQZ

Module: APLSCMSC

Called By: APLFXIIM

Description: Executes YYQZ service request (CMS).

Exit: Returns

SCRWAIT

Module: APLSCMSG

Called By: APLFXIIM

Description: Executes YYRWAIT service request (CMS).

Exit: Returns

SCRWAIT

Module: APLYUMSG

Called By: SCFXI via macro APLCCALL when interpreter issues YYRWAIT

Description: Waits until a user presses attention to unlock the keyboard after sending a message. This function is not provided for display terminals, to prevent lockout when TCAM has no read pending on the terminal. Instead, an immediate return is made with a normal return code.

Calls: Macros APLDEFN, APLCENTR, WAIT, APLCEXIT

Exit: Returns

SCSAVE

Module: APLSCLIB

Called By: APLFXIIM

Description: Executes YYSAVE service request (CMS).

Exit: Returns

SCSAVE

Module: APLYULIB

Called By: APLYUFXI

Description: Executes the YYSAVE service request (TSO). An internal routine, LIBDSN, is invoked to build the OS data set name and to determine password ownership of the library. An attempt is now made to allocate the data set with DISP=OLD. If the data set is now found, it is allocated with a DISP=NEW, and space determined by the current data in the active workspace. The data set is opened. To write the file to disk, the file is prefixed with an 80-byte header.

Output is written directly from the workspace to the data set using chained scheduled BSAM output. When all the data is written, the LIBUSN is called to close and free the data set and the 80 bytes are restored to WSMBUFF.

Exit: Returns

SCSAVOFL

Module: APLSCERR

Called By: CMS executor routines.

Description: Handles overflow of executor save area stack (CMS).

Exit: ABEND

SCSAVOFL

Module: APLYUERR

Called By: Macro APLCENTR

Description: Handles overflow of supervisor save area stack (TSO).

Exit: ABEND

SCSPIE

Module: APLSCERR

Called By: CMS, SPIE Exit

Description: For interpreter, auxiliary processor, or shared variable processor program check, returns to CMS with PIE PSW altered so that control is passed to SCAPL with a YYPRGX service request; for supervisor program check, issues ABEND macro.

Exit: See description

SCSPIE

Module: APLYUERR

Called By: MVS Program Interrupt Handler

Description: SPIE exit routine for interpreter and supervisor (TSO).

Exit: For interpreter or shared variable processor program check, returns to MVS with PIE PSW altered to resume at 'DOYYPRGX' in this module. DOYYPRGX will then exit to the interpreter by giving control to its main entry point, APLIINIT. For supervi program check, issues ABEND macro with system code '0CX'.

SCSRETR

Module: APLYUSHV

Called By: APLYUFXI

Description: This entry point executes the YYSRET service request (TSO).

Calls: ASVPSRVC, SCSAVOFL. Macros APLCENTR, APLCEXIT, ASVPON, ASVPOFR, ASVPRET, ASVPQRY, ASVPACC, ASVPSPC, ASVPREF, ASVPCPY, ASVPSOF, ASVPWAIT, WAIT

Exit: Returns

SCSTAE

Module: APLSCERR

Called By: CMS STAE Exit

Description: Address stops virtual machine to allow user to use CP commands to take storage dump or perform other problem determination actions (CMS).

Exit: Return to CMS ABEND Handler

SCSTAE

Module: APLYUERR

Called By: MVS ABEND Handler

Description: This is the ABEND (STAE) exit routine which receives control for all abends. The basic thrust of error recovery is to get the active workspace saved in CONTINUE, and then to cause TSO to reinvoke a clean copy of VS APL which will reload the CONTINUE workspace and continue processing. There are two principal kinds of abends: 1) X22 and X3E abends brought about by operator cancel, timing, TCAM error, etc. In these instances, the CONTINUE workspace is saved normally; 2) all other abends constitute error situations in which the CONTINUE workspace is marked nonloadable.

Exit: Returns to MVS ABEND Handler

SCSVACC

Module: APLSCSHV

Called By: APLFXIIM

Description: Executes YYSACC service request (CMS).

Calls: ASVPSRVC

Exit: Returns; SCSAVOFL(Error)

SCSVACC

Module: APLYUSHV

Called By: APLYUFXI

Description: This entry point executes the YYSACC service request (TSO).

Calls: ASVPSRVC. Macros APLCENTR, APLCEXIT, ASVPON, ASVPOFR, ASVPRET, ASVPQRY, ASVPACC, ASVPSPC, ASVPREF, ASVPCPY, ASVPSOF, ASVPWAIT, WAIT

Exit: Returns

SCSVCOPY

Module: APLSCSHV

Called By: APLFXIIM

Description: Executes YYSCOPY service request (CMS).

Calls: ASVPSRVC

Exit: Returns: SCSAVOFL(Error)

SCSVCOPY

Module: APLYUSHV

Called By: APLYUFXI

Description: This entry point executes the YYSCOPY service request (TSO).

Calls: ASVPSRVC, SCSAVOFL. Macros APLCENTR, APLCEXIT, ASVPON, ASVPOFR, ASVPRET, ASVPQRY, ASVPACC, ASVPSPC, ASVPREF, ASVPCPY, ASVPSOF, ASVPWAIT, WAIT

Exit: Returns

SCSVOFF

Module: APLSCSHV

Called By: APLFXIIM

Description: Executes YYSOFF service request (CMS).

Calls: ASVPSRVC

Exit: Returns; SCSAVOFL(Error)

SCSVOFF

Module: APLYUSHV

Called By: APLYUFXI

Description: This entry point executes the YYSOFF service request (TSO).

Calls: ASVPSRVC, SCSAVOFL. Macros APLCENTR, APLCEXIT, ASVPON, ASVPOFR, ASVPRET, ASVPQRY, ASVPACC, ASVPSPC, ASVPREF, ASVPCPY, ASVPSOF, ASVPWAIT, WAIT

Exit: Returns

SCSVOFFR

Module: APLSCSHV

Called By: APLFXIIM

Description: Executes YYSOFFER service request (CMS).

Calls: ASVPSRVC

Exit: Returns; SCSAVOFL(Error)

SCSVOFR

Module: APLYUSHV

Called By: APLYUFXI

Description: This entry point executes the YYSOFFER service request (TSO).

Calls: ASVPSRVC, SCSAVOFL. Macros APLCENTR, APLCEXIT, ASVPON, ASVPOFR, ASVPRET, ASVPQRY, ASVPACC, ASVPSPC, ASVPREF, ASVPCPY, ASVPSOF, ASVPWAIT, WAIT

Exit: Returns

SCSVON

Module: APLSCSHV

Called By: APLFXIIM

Description: Executes YYSON service request (CMS).

Calls: ASVPSRVC

Exit: Returns; SCSAVOFL(Error)

SCSVON

Module: APLYUSHV

Called By: APLYUFXI

Description: This entry point executes the YYSON service request (TSO).

Calls: ASVPSRVC, SCSAVOFL. Macros APLCENTR, APLCEXIT, ASVPON, ASVPOFR, ASVPRET, ASVPQRY, ASVPACC, ASVPSPC, ASVPREF, ASVPCPY, ASVPSOF, ASVPWAIT, WAIT

Exit: Returns

SCSVPINI

Module: APLSCSVI

Called By: APL

Description: Initializes shared memory and a task block for the interpreter and for each auxiliary processor running with VS APL under CMS.

Calls: CMS to establish external interrupt handler and each auxiliary processor to initialize. Macro HNDEXT.

Exit: To first auxiliary processor; as a result of its sign-on request, control is passed to ASVPSERV which exits to the next auxiliary processor; when all auxiliary processors have been called, control returns to routine APL.

SCSVQUER

Module: APLSCSHV

Called By: APLFXIIM

Description: Executes YYSQUERY service request (CMS).

Calls: ASVPSRVC

Exit: Returns; SCSAVOFL(Error)

SCSVQUER

Module: APLYUSHV

Called By: APLYUFXI

Description: This entry point executes the YYSQUERY service request (TSO).

Calls: ASVPSRVC, SCSAVOFL. Macros APLCENTR, APLCEXIT, ASVPON, ASVPOFR, ASVPRET, ASVPQRY, ASVPACC, ASVPSPC, ASVPREF, ASVPCPY, ASVPSOF, ASVPWAIT, WAIT

Exit: Returns

SCSVREF

Module: APLSCSHV

Called By: APLFXIIM

Description: Executes YYSREF service request (CMS).

Calls: ASVPSRVC

Exit: Returns; SCSAVOFL(Error)

SCSVREF

Module: APLYUSHV

Called By: APLYUFXI

Description: This entry point executes the YYSREF service request (TSO).

Calls: ASVPSRVC, SCSAVOFL. Macros APLCENTR, APLCEXIT, ASVPON, ASVPOFR, ASVPRET, ASVPQRY, ASVPACC, ASVPSPC, ASVPREF, ASVPCPY, ASVPSOF, ASVPWAIT, WAIT

Exit: Returns

SCSVRETR

Module: APLSCSHV

Called By: APLFXIIM

Description: Executes YYSRET service request (CMS).

Calls: ASVPSRVC

Exit: Returns; SCSAVOFL(Error)

SCSVSPEC

Module: APLSCSHV

Called By: APLFXIIM

Description: Executes YYSSPEC service request (CMS).

Calls: ASVPSRVC

Exit: Returns; SCSAVOFL(Error)

SCSVSPEC

Module: APLYUSHV

Called By: APLYUFXI

Description: This entry point executes the YYSSPEC service request (TSO).

Calls: ASVPSRVC, SCSAVOFL. Macros APLCENTR, APLCEXIT, ASVPON, ASVPOFR, ASVPRET, ASVPQRY, ASVPACC, ASVPSPC, ASVPREF, ASVPCPY, ASVPSOF, ASVPWAIT, WAIT

Exit: Returns

SCSYSER

Module: APLSCERR

Called By: APLFXIIM

Description: Executes the YYSYSER service request (CMS).

Exit: Returns; SCSAVOFL(Error)

SCSYSER

Module: APLYUERR

Called By: APLYUFXI

Description: Executes the YYSYSER service request (TSO).

Exit: Returns; SCSAVOFL(Error)

SCTABS

Module: APLSCTYP

Called By: APLFXIIM

Description: Executes YYTABS service request (CMS).

Exit: Returns; SCSAVOFL(Error)

SCTABS

Module: APLYUTYP

Called By: SCFXI via macro APLCCALL

Description: This entry point provides supervisor support for QUAD-HT (TSO).

Calls: Macros APLDEFN, APLCENTR, APLCEXIT

Exit: Returns

SCTIME

Module: APLSCMSC

Called By: APLFXIIM

Description: Executes YYTIME service request (CMS).

Exit: Returns; SCSAVOFL(Error)

SCTIME

Module: APLYUMSC

Called By: APLYUFXI

Description: Executes the YYTIME service request (TSO). YYTIME finds the current time and date in VS APL standard time format and returns it in the first eight bytes of WSMSVLRQ.

Exit: Returns; SCSAVOFL(Error)

SCTRAN

Module: APLSCMSG

Called By: APLFXIIM

Description: Executes YYTRAN service request (CMS).

Exit: Returns; SCSAVOFL(Error)

SCTRAN

Module: APLYUMSG

Called By: SCFXI via macro APLCCALL when interpreter issues APLSVCC YYTRAN

Description: Executes WTO or TPUT to transmit a message to operator or user (TSO). The message is translated from Z-code to lowercase EBCDIC. If it is to go to the operator, a 4-byte

WTO prefix is generated, and the WTO macro is used to send it. TPUT is used, otherwise, to send the message to the TSO user specified in the command. For MVS, the 'USERIDL' form of TPUT is issued.

Calls: Macros APLDEFN, APLCENTR, APLCEXIT, APLCCVO, TPUT, WTO

Exit: Returns; SCSAVOFL(Error)

SCTYI

Module: APLSCTYP

Called By: SCTYOI, APLFXIIM

Description: Executes YYTYI service request for typewriter terminal (CMS).

Exit: Returns; SCSAVOFL(Error)

SCTYI

Module: APLYUTYP

Called By: SCFXI via macro APLCCALL

Description: This entry point performs terminal input (TSO).

Calls: APLYUTRM. Macros APLDEFN, APLCENTR, APLCEXIT

Exit: Returns

SCTYO

Module: APLSCTYP

Called By: SCTYOI, APLFXIIM

Description: Executes YYTYO service request for typewriter terminal (CMS).

Exit: Returns; SCSAVOFL(Error)

SCTYO

Module: APLYUTYP

Called By: SCFXI via macro APLCCALL

Description: This entry point performs terminal output (TSO).

Calls: APLYUTRM. Macros APLDEFN, APLCENTR, CHKSINK (LOCAL macro), APLCEXIT

Exit: Returns

SCTYOI

Module: APLSCTYP

Called By: APLFXIIM

Description: Executes YYTYOI service request for typewriter terminal and read input (CMS).

Calls: SCTYO, SCTYI

Exit: Returns; SCSAVOFL(Error)

SCTYOI

Module: APLYUTYP

Called By: SCFXI via macro APLCCALL

Description: This entry point transmits a prompt to a terminal, then performs terminal input (TSO).

Calls: APLYUTIO, SCTYO, SCTYI. Macros APLDEFN, APLCENTR, APLCCALL, CHKSINK, APLCEXIT

Exit: Returns

SCWIDTH

Module: APLSCTYP

Called By: APLFXIIM

Description: Executes YYWIDTH service request (CMS).

Exit: Returns; SCSAVOFL (Error)

SCWIDTH

Module: APLYUTYP

Called By: SCFXI via macro APLCCALL

Description: This entry point provides supervisor support for QUAD-PW (TSO).

Calls: Macros APLDEFN, APLCENTR, APLCEXIT

Exit: Returns

SCWSID

Module: APLSCMSC

Called By: APLFXIIM

Description: Executes YYWSID service request (CMS).

Exit: Returns: SCSAVOFL(Error)

SCWSID

Module: APLYUMSC

Called By: APLYUFXI

Description: Executes the YYWSID service request (TSO). In response to a)WSID command, YYWSID either changes the active workspace ID or returns the current ID.

Calls: HELPENQ

Exit: Returns; SCSAVOFL(Error)

SSSATACH

Module: APLSCTYP

Called By: APLFXIIM

Description: Executes YYWIDTH service request (CMS).

Exit: Returns; SCSAVOFL(Error)

SSSATACH

Module: APLYUSSH

Called By: Operating system

Description: This entry point intercepts the ATTACH SVC so that those IQEs that are needed can be propagated to the new TCB (TSO).

Exit: Returns

SSSROUTR

Module: APLYUSSH

Called By: Operating system

Description: This SVC intercept is called when an intercepted SVC is issued under the current TCB (TSO).

Exit: Returns

SSSSVC

Module: APLYUSSH

Called By: APLYUINI

Description: This entry point creates or deletes SVC subscreens and their associated intercept routines (TSO).

Calls: Macros FESTAE, FREEMAIN, GETMAIN, SETLOCK, TESTAUTH, MODESET

Exit: Returns

SECTION 4. DIRECTORY

This section includes two lists of entry points and the names of the modules in which they appear. The first list is given alphabetically by entry point: the second is given alphabetically by module name. In each list, column 3 gives the numbers of diagrams from Section 2 of this book in which the entry point or module is referred to.

FOR DOS/VS: The conversion modules for DOS/VS differ from those for OS/VS. These modules are functionally the same, but the DOS/VS modules are designed to interface with DOS/VS and the OS/VS modules with OS/VS. The OS/VS modules begin with the characters APLO; the DOS/VS modules begin with the characters APLD. To avoid unnecessary repetition in this publication, only the OS/VS names are used in this publication wherever possible. Unless explicitly noted otherwise, substitute the prefix APLD for APLO when using this publication for DOS/VS VS APL.

ENTRY POINTS AND MODULE NAMES SORTED BY ENTRY POINTS

Entry Point or Routine Name	Module Micro-fiche Name	Method of Operation Diagram
APCREATE	APLPAPCD	1.1.1
APDFN	APLPAPCD	
APDROP	APLPAPCD	1.1.1
APFILSIZ	APLPAPCD	1.1.1
APIO	APLPAPCD	1.1.1
APL	APL	
	APLYUINI	1.4
APLACCBE	APLACCBE	
APLACDSL	APLACDSL	8.2.1
APLACHLP	APLACHLP	8.2.1
APLACMDF	APLACNDP	
APLACMDX	APLACMDX	8.2.1
APLACMER	APLACQRY	
APLACNDP	APLACNDP	8.2.1
APLACOPL	APLACOPY	
APLACOPY	APLACOPY	
APLACPRM	APLACPRM	8.2.1
APLACPRO	APLACPRO	8.2.1
APLACQRY	APLACQRY	8.2.1
APLACQUE	APLACQUE	8.2.1
APLACRCP	APLACRCP	8.2.1
APLACRDA	APLACRDA	8.2.1
APLACRSA	APLACRSA	8.2.1
APLACSF	APLACSF	8.2.1
APLACXCM	APLACXCM	8.2.1
APLAD	APLAD	
APLADMSG	APLADMSG	8.2.1
APLADSON	APLACRDA	
APLADTTM	APLADTTM	
APLAERRM	APLASCHD	8.2
APLAESTK	APLAESTK	
APLAINIT	APLASCHD	8.2
APLALINE	APLALINE	
APLAMODE	APLACNDP	
APLAPAGE	APLALINE	
APLAPASS	APLASCHD	8.2
APLATERM	APLASCHD	8.2
APLATYI	APLASCHD	8.2
APLATYO	APLASCHD	8.2
APLATYOI	APLASCHD	8.2
APLAUALT	APLAK	
	APLASA	
	APLAYA	
APLAUATN	APLAKP	
	APLASP	
	APLAYP	
APLAUCAE	APLAK	
	APLAS	
	APLAY	
APLAUNCO	APLAK	
	APLAS	
	APLAY	
APLAUPRO	APLAK	
	APLAS	
	APLAY	
APLAUSRX	APLAUSRX	
APLAXCMD	APLASCHD	8.2
APLFXIIM	APLFXIIM	
	APLKIFIX	1.3
	APLPFXIM	1.1
	APLSCFXI	1.2
	APLYUFXI	1.4
	APLYUIIM	
APLIINIT	APLITINI	
APLKADEF	APLKADEF	1.3
APLKADSP	APLKADSP	1.3
APLKAGBL	APLKAGBL	1.3
APLKAHST	APLKAHST	
APLKAMIX	APLKAMIX	
APLKASON	APLKASON	1.3
APLKEHCP	APLKEHCP	1.3
APLKEMGR	APLKEMGR	1.3
APLKIFON	APLKIFIX	1.3
APLKISVI	APLKISVI	1.3
APLKLIBF	APLKLIBF	1.3
APLKLIBG	APLKLIBG	1.3

Entry Point or Routine Name	Module Microfiche Name	Method of Operation Diagram
APLKLIBI	APLKLIBB	1.3
APLKLIBR	APLKLIBG	1.3
	APLKLIBR	1.3
APLKLIBT	APLKLIBB	1.3
APLKLUIT	APLKLIBC	
APLKLUTM	APLKLIBC	
APLKPFAP	APLKASTB	1.3
APLKPFOH	APLKASTB	1.3
APLKSPRG	APLKVEXC	7.0
APLKSSR	APLKSSVP	1.3, 1.3.1
APLKSSUB	APLKSSUB	1.3, 1.3.1
APLKTCTL	APLKTCTL	1.3
APLKTCWR	APLKTCWR	1.3
APLPAPAC	APLPAPAB	1.1.1
APLPAPOF	APLPAPAB	1.1.1
APLPAPPR	APLPAPAB	1.1.1
APLPAPRT	APLPAPAB	1.1.1
APLPAPSF	APLPAPAB	1.1.1
APLPCENT	APLPCOEX	1.1
APLPCOAP	APLPCOAP	
APLP126T	APLP126T	
APLSCSSI	APLSCSSI	
APLSHACC	APLSHACC	1.2.1
APLSHBPB	APLSHBPB	1.2.1
APLSHBVB	APLSHBVB	1.2.1
APLSHCPY	APLSHCPY	1.2.1
APLSHGET	APLSHGET	1.2.1
APLSHOFR	APLSHOFR	1.2.1
APLSHPST	APLSCSVI	1.2.1
APLSHPUT	APLSHPUT	1.2.1
APLSHQUE	APLSHQRE	
APLSHREF	APLSHREF	1.2.1
APLSHRET	APLSHRET	1.2.1
APLSHSOF	APLSHSOF	1.2.1
APLSHSON	APLSHSON	1.2.1
APLSHSPC	APLSHSPC	1.2.1
APLSHSRD	APLSHSRD	1.2.1
APLSHSUB	APLSHSUB	1.2.1
APLXACSO	APLXAC	8.3
APLXACSV	APLXAC	
APLXAKSO	APLXAK	8.3
APLXAKSV	APLXAK	8.3
APLXAINP	APLXASD	8.3
	APLXAYD	8.3
APLXAMSG	APLXASD	8.3
	APLXAYD	8.3
APLXBACK	APLXSTAK	
APLXBSAB	APLSCSVI	
APLXBSXT	APLSCSVI	
APLXBYAB	APLYUSVI	1.2.1, 8.4.1, 8.4.3
APLXBYXT	APLYUSVI	1.2.1, 8.4.1, 8.4.3
APLXCALL	APLXSTAK	
APLXDKMP	APLXDKMP	
APLXDUCL	APLXDUMP	
APLXDUMP	APLXDUMP	
APLXDUOP	APLXDUMP	
APLXFINT	APLXFSFL	
	APLXFYFL	
APLXFSFL	APLXFSFL	
APLXFTRM	APLXFSFL	
	APLXFYFL	
APLXFYFL	APLXFYFL	
APLXGCAT	APLXGCAT	
APLXGCHC	APLXGCHC	
APLXGCOM	APLXGCOM	
APLXGKON	APLXGKON	
APLXGKR	APLXGKR	
APLXGKRQ	APLXGKRQ	
APLXGKRR	APLXGKRR	
APLXGKT	APLXGKT	
APLXGKU	APLXGKU	
APLXGS	APLXGS	
APLXGY	APLXGY	
APLXGYON	APLXGY	
APLXGYRQ	APLXGY	
APLXMKSG	APLXMKSG	
APLXMSSG	APLXMSSG	
APLXMYSG	APLXMYSG	
APLXPK	APLXPK	
APLXPY	APLXPY	
APLXSTAK	APLXSTAK	
APLXTRAN	APLXTRAN	
APLXTREZ	APLXTRAN	
APLXTRZE	APLXTRAN	
APLXVERS	APLXVERS	
APLXWKWP	APLXWKWP	
APLXWSWP	APLSCSVI	
APLXWYWP	APLXWYWP	
	APLYUSVI	1.2.1, 8.4.1, 8.4.3
APLYDAIR	APLYDAIR	
APLYUCMD	APLYUCMD	
APLYUCNV	APLYUCNV	
APLYUEXC	APLYUEXC	
APLYUFXI	APLYUFXI	1.4
APLYUHSH	APLYUHSH	
APLYULNE	APLYULNE	
APLYURVC	APLYURVC	
APLYUTBL	APLYUTBL	
APLYUTIO	APLYUTIO	
APLYUUSR	APLYUUSR	
APL100	APLYU100	
	APL100	1.2.2, 1.4.1
APL100K	APL100K	1.3.2
APL100KO	APL100KO	1.3.2
APL101	APLYU101	
	APL101	1.2.2, 1.4.1
APL102	APLYU102	1.4.1
APL102K	APL102K	1.3.2
APL110	APL110	1.2.2
APL111	APLYU111	
	APL111	1.2.2, 1.4.1
APL120	APL120	8.4.1
APL121	APL121	
APL121K	APL121K	1.3.2, 8.4.3
APL123	APL123	1.2.2, 1.4.1 1.3.2
APL123K	APL123K	1.3.2
APL124K	APL124K	1.3.2
APL125K	APL125K	1.3.2
APL126	APL126	8.4.2, 8.4.3
APL126T	APL126T	
APL132K	APL132K	1.3.2
APL139K	APL139K	1.3.2
APL210	APLYU210	1.4.1
APOPEN	APLPAPCD	1.1.1
APPASSWD	APLPAPCD	1.1.1
APSHARE	APLPAPCD	1.1.1
APVIO	APLPAPCD	1.1.1
ASVPSERV	APLSCSVI	1.2.1, 8.4.3
	APLYUSVI	1.2.1, 8.4.1, 8.4.3

Entry Point or Routine Name	Module Microfiche Name	Method of Operation Diagram
ASVPSRVC	APLYURVC	
	ASVPSRVC	
BEXIT	APLKASTB	
COIBM	APLCOIBM	
CVCULL	APLCCULL	
	APLOCULL	
CVDATE	APLCMISC	
	APLOMISC	
	APLQMISC	
CVDIRE	APLCMISC	
	APLODIRE	
CVDISP	APLCDISP	
	APLODISP	
	APLQDISP	
CVFUNC	APLCFUNC	
	APLOFUNC	
	APLQFUNC	
CVGDIR	APLODIRE	
CVGRUP	APLCGRUP	
	APLOGRUP	
	APLQGRUP	
CVIBNM	APLCIBNM	
	APLOIBNM	
	APLQIBNM	
CVINIT	APLCINIT	6.0
	APLOINIT	6.0
	APLQINIT	6.0
CVIOER	APLCMISC	
	APLOMISC	
CVLEAR	APLCLEAR	
	APLOLEAR	
	APLQLEAR	
CVPARM	APLCPARM	
	APLOPARM	
	APLQPARM	
CVPRTR	APLCMISC	
	APLOMISC	
	APLQMISC	
CVRPRT	APLCRPRT	
	APLORPRT	
	APLQRPRT	
CVSAVE	APLCSAVE	
	APLOSAVE	
	APLQSAVE	
CVSHIP	APLCSHIP	
	APLOSHIP	
CVSLST	APLOSLST	
CVSPIE	APLCSPIE	
	APLOSPIE	
	APLQSPIE	
CVTBCD	APLCTBCD	
	APLOTBCD	
CVTIDY	APLCMISC	
	APLOTIDY	
	APLQMISC	
CVVARB	APLCVARB	
	APLOVARB	
	APLQVARB	
CVWKSP	APLCWKSP	
	APLOWKSP	
	APLQWKSP	
CVWSFN	APLCWSFN	
	APLOWSFN	
DMSSCND	APLYUSCN	
DMSSCNN	APLYUSCN	
ERENDEX	APLPCOEX	1.1
ERMSGRTN	APLPSERR	

Entry Point or Routine Name	Module Microfiche Name	Method of Operation Diagram
ERSAVEAR	APLPCOEX	1.1
ERTIMDAT	APLPSERR	
FREESTOR	APLPAPGD	
FSMBUZZ	APLPAPFS	1.1.1
FSMFORMT	APLPAPFS	1.1.1
FSMGET	APLPAPFS	1.1.1
FSMHCOPY	APLPAPFS	1.1.1
FSMMINT	APLPAPFS	1.1.1
FSMMTYPE	APLPAPFS	1.1.1
FSMREAD	APLPAPFS	1.1.1
FSMRFORM	APLPAPFS	1.1.1
FSMSETC	APLPAPFS	1.1.1
FSMSUB1	APLPAPFS	1.1.1
FSMSUB2	APLPAPFS	1.1.1
FSMSUB3	APLPAPFS	1.1.1
FSMWRITE	APLPAPFS	1.1.1
GDDMCRET	APLPAPGB	1.1.1
GDDMRCTL	APLPAPGC	1.1.1
GDDMSCTL	APLPAPGB	1.1.1
GDDMSDAT	APLPAPGB	1.1.1
GDDMSOFF	APLPAPGB	1.1.1
GDDX	APLPAPGD	1.1.1
GDDXINIT	APLPAPGD	1.1.1
GETSTOR	APLPAPGD	1.1.1
IABNM	APLIATRN	4.1.5
IACAL370	APLIEXFR	
IACHK	APLIACHK	
IACIRCLE	APLIACIR	
IACMX	APLIECMX	
IACOMMA	APLIERHO	
IADDOM	APLIADOM	4.1.3
IADEAL	APLIATRN	4.1.5
IADECODE	APLIADEC	4.1.3
IADFORM	APLIAFOR	4.1.3
IADSHARE	APLIATRN	4.1.5
IADTRAN	APLIATSP	
IADYB	APLIEFCH	
IAENCODE	APLIAENC	4.1.3
IAEXECTE	APLIATRN	4.1.5
IAEXNAME	APLIATRN	4.1.5
IAEXPR	APLIATRS	
IAEXSTCK	APLIATRN	4.1.5
IAFACT	APLIATRS	
IAFACTRL	APLIATRN	4.1.5
IAFCHNAM	APLIANAM	
IAFLCL	APLIATRN	4.1.5
IAGFMT	APLIAGFM	
IAGFMT2	APLIAGFM	
IAGOUT	APLIAGOU	4.1, 4.1.5
IAGRADE	APLIAGRD	4.1.3
IAHTSPEC	APLIASYV	4.1.5
IAIPROD	APLIAPRD	4.1.3
IAIROLL	APLIATRN	4.1.5
IALOG	APLIATRN	4.1.5
IALOGR	APLIATRS	
IAMDOM	APLIADOM	4.1.3
IAMFORM	APLIAFOR	4.1.3
IAMSHARE	APLIATRN	4.1.5
IAMTRAN	APLIATSP	
IAPLFUN	APLIATRN	4.1.5
IAPOW	APLIATRN	4.1.5
IAQCR	APLIAQFN	
IAQDL	APLIAQFN	
IAQDSPEC	APLIATRN	4.1.5
IAQEX	APLIAQFN	
IAQFX	APLIAQFN	
IAQNC	APLIAQFN	

Entry Point or Routine Name	Module Microfiche Name	Method of Operation Diagram
IAQNL	APLIAQFN	
IAQSVC	APLIASHF	
IAQSVO	APLIASHF	
IAQSVQ	APLIASHF	
IAQSVR	APLIASHF	
IAQUADS	APLIATRN	4.1.5
IAQUADSA	APLIATRN	4.1.5
IAREDU	APLIARED	
IARESIDU	APLIATRN	4.1.5
IAREVARY	APLIAROT	
IAROLL	APLIATRN	4.1.5
IAROTA	APLIAROT	
IARTOI	APLIASYV	4.1.5
IARTRACT	APLIASHV	4.1.5
IASCAN	APLIASCN	
IASCOPY	APLIASHV	4.1.5
IASFIND	APLIANAM	
IASHADO	APLIASYV	4.1.5
IASHRPST	APLIATRN	4.1.5
IASHSPEC	APLIASHV	4.1.5
IASQRT	APLIATRS	
IASVOFF	APLIASHV	4.1.5
IASVON	APLIASHV	4.1.5
IASYSPEC	APLIASYV	4.1.5
IASYSPST	APLIASYV	4.1.5
IASYSREF	APLIASYV	4.1.5
IATABREF	APLIASYV	4.1.5
IATIDY	APLIATRN	4.1.5
IATKDP	APLIATAK	
IATOBCD	APLIATBC	
IATOBCD2	APLIATBC	
IAUNSHAD	APLIASYV	4.1.5
IAUNSHR	APLIASHV	4.1.5
IAVALNAM	APLIANAM	
IEABEND	APLIEXAR	4.1
IECHIX	APLIEMND	4.1.3
IECMEX	APLIECMX	
IECOMMA	APLIERHO	
IECONVR	APLIEFCH	
IECOPY	APLIERHO	
IEDATTN	APLIEFXR	
IEDYAD	APLIESCA	4.1, 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.1.5
IEDYB	APLIEFCH	
IEEPSIOT	APLIEPSI	
IEFIND	APLIESPA	3.2
IEFREE	APLIESPA	3.2
IEFUNN	APLIEFNM	4.1.1, 4.1.2
IEGETNI	APLIEFCH	
IEGETNL	APLIEFCH	
IEGETNR	APLIEFCH	
IEGETV	APLIEFCH	
IEGINITI	APLIEFCH	
IEGINITL	APLIEFCH	
IEGINITR	APLIEFCH	
IEGOGOMN	APLIEFNM	4.1.1, 4.1.2
IEGOGOSC	APLIEFNM	4.1.1, 4.1.2
IEGTSPAC	APLIESPA	3.2
IEINDB	APLIEIDX	4.1.4
IEINDD	APLIEIDX	4.1.4
IELDSTK	APLIEXAR	4.1
IEMONAD	APLIEMND	4.1.3
IENAME	APLIESPA	3.2
IERSHP	APLIERHO	
IESCANG	APLIESCA	4.1, 4.1.1, 4.1.2, 4.1.3,
		4.1.4, 4.1.5
IESFIND	APLIESPA	3.2
IESFREE	APLIESPA	3.2
IESGETN	APLIEFCH	
IESGETV	APLIEFCH	
IESGINIT	APLIEFCH	
IESNAME	APLIESPA	3.2
IESPACST	APLIESPA	3.2
IESTOSTK	APLIEXAR	4.1
IESUNFUN	APLIEFNM	4.1.1, 4.1.2
IESYNN	APLIESPA	3.2
IETKDP	APLIETAK	
IEUNFN	APLIEFNM	4.1.1, 4.1.2
IEXARCH	APLIEXAR	4.1
IEXIT	APLIEXAR	4.1
ITBFTYO	APLITSUB	4.2
ITBLDID	APLITIDS	3.2
ITBLDOD	APLITIDS	3.2
ITCKALPN	APLITSUB	4.2
ITCLOSET	APLITFDC	3.0, 3.1, 3.2
ITCMCLEA	APLITCML	
ITCMCONT	APLITCMT	
ITCMCOPO	APLITCPO	
ITCMCOPY	APLITCMC	
ITCMDOST	APLITCMS	
ITCMDROP	APLITCML	
ITCMERAS	APLITCME	
ITCMFNS	APLITCMF	
ITCMFVG	APLITCMF	
ITCMGROU	APLITCMG	
ITCMGRP	APLITCMG	
ITCMGRPS	APLITCMF	
ITCMLIB	APLITCML	
ITCMLOAD	APLITCML	
ITCMMSG	APLITCMT	
ITCMOFF	APLITCMT	
ITCMOPR	APLITCMT	
ITCMPCOP	APLITCMC	
ITCMQUOT	APLITCML	
ITCMSAVE	APLITCML	
ITCMSI	APLITCMI	
ITCMSINL	APLITCMI	
ITCMSTAC	APLITCMS	
ITCMSYMB	APLITCMS	
ITCMVARS	APLITCMF	
ITCMWSID	APLITCML	
ITCMWSSI	APLITCML	
ITCOFIN	APLITCPI	
ITDELETE	APLITCME	
ITEMPFUN	APLITFUN	
ITERRORS	APLITERR	4.2
ITEXECUT	APLITEX	4.0, 4.2
ITFDCLOS	APLITFDC	3.0, 3.1, 3.2
ITFDCVT	APLITNCV	3.2
ITFDEDIT	APLITFDE	3.0, 3.1
ITFDKILL	APLITFDC	3.0, 3.1, 3.2
ITFDNWLN	APLITFDN	
ITFDOPEN	APLITFDO	3.0, 3.1
ITFDTSOF	APLITFDC	3.0, 3.1, 3.2
ITFETCH	APLITFCH	
ITFNLNO	APLITSUB	4.2
ITFORCOF	APLITINP	2.0, 3.0
ITININT	APLITNCV	3.2
ITINPINI	APLITINP	2.0, 3.0
ITINPUT	APLITINP	2.0, 3.0
ITLIBMSG	APLITCML	
ITLINEO	APLITHDR	3.0, 3.1, 3.2

Entry Point or Routine Name	Module Microfiche Name	Method of Operation Diagram
ITLOUT	APLITSUB	4.2
ITNAMINI	APLITCME	
ITNUMCVT	APLITNCV	3.2
ITOKENIZ	APLITLXS	3.2
ITPRFNLN	APLITSUB	4.2
ITPRINTC	APLITSUB	4.2
ITPRLINE	APLITPRL	3.1
ITPRNAME	APLITSUB	4.2
ITPRNUM	APLITSUB	4.2
ITPRWSID	APLITCML	
ITSAVWS	APLITCML	
ITSHV	APLITCML	
ITSQUIRT	APLITSUB	4.2
ITSTSRCH	APLITIDS	3.2
ITSYSCMD	APLITCMD	5.0
ITSYSERR	APLITINI	
ITTIME	APLITSUB	4.2
ITTIMSUB	APLITSUB	4.2
ITTYERR	APLITSUB	4.2
ITTYIZ	APLITINP	2.0, 3.0
ITUSADF	APLITUSG	
ITUSAG	APLITUSG	
ITUSASH	APLITCMI	
ITXBLNL	APLITSUB	4.2
KABEXIT	APLKADSP	1.3
KABOOTS	APLKASTB	1.3
KADEPON	APLKADSP	1.3
KCASE2Q	APLKSSUB	1.3, 1.3.1
KCASE3Q	APLKSSUB	1.3, 1.3.1
KCATOFF	APLKMSCB	1.3
KCDELAY	APLKMSCA	1.3
KCDUMP	APLKMSCA	1.3
KCLEANUP	APLKSSUB	1.3, 1.3.1
KCLEAR	APLKLIBU	1.3
KCMBL	APLKMSCB	1.3
KCOPA	APLKLIBU	1.3
KCOPI	APLKLIBU	1.3
KCOPO	APLKLIBU	1.3
KCOPZ	APLKLIBU	1.3
KCQAI	APLKMSCA	1.3
KCQUOTA	APLKMSCB	1.3
KCQZ	APLKIFIX	1.3
KCSYSER	APLKMSCA	1.3
KCTABS	APLKMSCB	1.3
KCTIME	APLKMSCA	1.3
KCTRAN	APLKMSCB	1.3
KCWIDTH	APLKMSCB	1.3
KDPCREG	APLKDOPS	1.3
	APLKVOPS	1.3
KDPFAB	APLKVOPS	
KDPFAP	APLKDOPS	1.3
KDROP	APLKLIBU	1.3
KFREESP	APLKSSUB	1.3, 1.3.1
KGCFILE	APLKLIBV	1.3
KGCOL	APLKSSUB	1.3, 1.3.1
KGDFILE	APLKLIBV	1.3
KGDROP	APLKLIBG	
KGETSPAC	APLKSSUB	1.3, 1.3.1
KGLOAD	APLKLIBG	1.3
KGSAVE	APLKLIBG	1.3
KGUDIR	APLKLIBG	1.3
KGUFILE	APLKLIBV	1.3
KGWDIR	APLKLIBG	
KGWLIB	APLKLIBV	1.3
KIDSETUP	APLKSSUB	1.3, 1.3.1
KIFONEXT	APLKIFIX	1.3
KINIEX	APLKDOPS	1.3
	APLKVOPS	1.3
KLALLOC	APLKLIBA	1.3
KLCLOS	APLKVOPS	1.3
KLDEALOC	APLKLIBA	1.3
KLGET	APLKDOPS	1.3
	APLKVOPS	1.3
KLIB	APLKLIBU	1.3
KLOAD	APLKLIBU	1.3
KLOPEN	APLKVOPS	1.3
KLPUT	APLKVOPS	1.3
KLRDBITM	APLKLIBV	
KMACRO	APLKADSP	1.3
KPASS	APLKLIBU	1.3
KPGMCHK	APLKIFIX	1.3
KPOSTWAI	APLKSSUB	1.3, 1.3.1
KPPSEARC	APLKSSUB	1.3, 1.3.1
KPROCOFF	APLKSSUB	1.3, 1.3.1
KRETSUB	APLKSSUB	1.3, 1.3.1
KRSTEX	APLKDOPS	1.3
	APLKVOPS	1.3
KSAVE	APLKLIBU	1.3
KSEIZE	APLKSSUB	1.3, 1.3.1
KSETEX	APLKDOPS	1.3
	APLKVOPS	1.3
KSINGAL	APLKSSUB	1.3, 1.3.1
KSPALD	APLKVALD	
KSPAUT	APLKVAUT	7.0
KSPCMD	APLKCVMD	
KSPCPY	APLKVCPY	7.0
KSPDOS	APLKDDOS	
KSPDSI	APLKVDSI	
KSPDSO	APLKVDSO	
KSPEXP	APLKVEXP	7.0
KSPFMT	APLKVFMT	7.0
KSPIMP	APLKVIMP	7.0
KSPINT	APLKVINT	7.0
KSPLBI	APLKVLBI	
KSPLBO	APLKVLBO	
KSPMSG	APLKVMSG	
KSPPIN	APLKVPIN	
KSPSCN	APLKVCMD	7.0
KSPTPO	APLKVTPO	
KSPTRM	APLKVTRM	7.0
KTIMEREX	APLKDOPS	1.3
	APLKVOPS	1.3
KTOINTER	APLKIFIX	1.3
KTRAL	APLKTRQO	1.3
KTRCU	APLKTRQO	1.3
KTRFA	APLKTRQO	1.3
KTRFI	APLKTREQ	1.3
KTRFM	APLKTRQO	1.3
KTRGD	APLKTREQ	1.3
KTRGF	APLKTREQ	1.3
KTRHC	APLKTRQO	1.3
KTRIN	APLKTREQ	1.3
KTRRD	APLKTREQ	1.3
KTRRS	APLKTRQO	1.3
KTRRT	APLKTREQ	1.3
KTRTRAN	APLKTRAN	1.3
KTRWR	APLKTRQO	1.3
KTSCHED	APLKTSRV	1.3
KTSCLEAR	APLKTSRV	1.3
KTSFCHK	APLKTSRV	1.3
KTSFNDF	APLKTSRV	1.3
KTSLINO	APLKTSRV	1.3
KTSLOCID	APLKTSRV	1.3
KTSLOCR	APLKTSRV	1.3

Entry Point or Routine Name	Module Microfiche Name	Method of Operation Diagram
KWSID	APLKLIBU	1.3
KYYTYOI	APLKIFIX	
KYYOFF	APLKMSCB	1.3
OFF121X	APL121	
PCATOFF	APLPMISC	
PCCLEAR	APLPLIBS	
PCCMD	APLPMISC	
PCCOPA	APLPLIBS	
PCCOPI	APLPMISC	
PCCOPO	APLPMISC	
PCCOPZ	APLPMISC	
PCDELAY	APLPTYIO	
PCDROP	APLPLIBS	
PCDTYI	APLPTYIO	
PCDTYO	APLPTYIO	
PCDTYOI	APLPTYIO	
PCDUMP	APLPSERR	
PCLIB	APLPLIBS	
PCLOAD	APLPLIBS	
PCMBL	APLPTYIO	
PCOFF	APLPMISC	
PCPASS	APLPMISC	
PCQAI	APLPMISC	
PCQUOTA	APLPMISC	
PCQZ	APLPMISC	
PCRWAIT	APLPTYIO	
PCSACC	APLPSHVR	1.1.1
PCSAVE	APLPLIBS	
PCSCOPY	APLPSHVR	1.1.1
PCSOFF	APLPSHVR	1.1.1
PCSOFFER	APLPSHVR	1.1.1
PCSON	APLPSHVR	1.1.1
PCSQUERY	APLPSHVR	1.1.1
PCSREF	APLPSHVR	1.1.1
PCSRET	APLPSHVR	1.1.1
PCSSPEC	APLPSHVR	1.1.1
PCSYSER	APLPSERR	
PCTABS	APLPTYIO	
PCTIME	APLPMISC	
PCTRAN	APLPTYIO	
PCTYI	APLPTYIO	
PCTYO	APLPTYIO	
PCTYOI	APLPTYIO	
PCWIDTH	APLPTYIO	
PCWSID	APLPLIBS	
PRDDIR	APLPAPCD	1.1.1
PRDSEQ	APLPAPCD	1.1.1
PWRITE	APLPAPCD	1.1.1
RET121X	APL121	
SCAPL	APLSCFXI	1.2
SCATOFF	APLSCTYP	1.2
	APLYUTYP	
SCATTN	APLSCTYP	1.2
	APLYUTYP	
SCCLEAR	APLSCMSC	
	APLYUMSC	
SCCMD	APLSCMSC	
	APLYUMSC	
SCCOPA	APLSCOPY	
SCCOPI	APLSCOPY	
SCCOPO	APLSCOPY	
SCCOPZ	APLSCOPY	
SCDELAY	APLSCMSC	
	APLYUMSC	
SCDPA2	APLSCDPY	
SCDROP	APLSCLIB	
	APLYULIB	
SCDTYI	APLSCDPY	
	APLYUDPY	
SCDTYIO	APLYUDPY	
SCDTYO	APLSCDPY	
	APLYUDPY	
SCDTYOI	APLSCDPY	
SCDUMP	APLSCERR	1.2
	APLYUERR	1.4
SCENDAPL	APLSCINI	1.2
	APLYUINI	1.4
SCEXTINY	APLSCSVI	
SCFID	APLSCFID	
SCLIB	APLSCLIB	
	APLYULIB	
SCLOAD	APLSCLIB	
	APLYULIB	
SCMBL	APLSCMSG	
	APLYUMSG	
SCMICRO	APLSCINI	1.2
	APLYUINI	1.4
SCOFF	APLSCMSC	
	APLYUINI	
SCPASS	APLSCMSC	
	APLYUMSC	
SCQAI	APLSCMSC	
	APLYUMSC	
SCQUOTA	APLSCMSC	
	APLYUMSC	
SCQZ	APLSCMSC	
SCRWAIT	APLSCMSG	
	APLYUMSC	
SCSAVE	APLSCLIB	
	APLYULIB	
SCSAVOFL	APLSCERR	1.2
	APLYUERR	1.4
SCSPIE	APLSCERR	1.2
	APLYUERR	1.4
SCSRETR	APLYUSHV	
SCSTAE	APLSCERR	1.2
	APLYUERR	1.4
SCSVACC	APLSCSHV	
	APLYUSHV	
SCSVCOPY	APLSCSHV	
	APLYUSHV	
SCSVOFF	APLSCSHV	
	APLYUSHV	
SCSVOFFR	APLSCSHV	
SCSVOFR	APLYUSHV	
SCSVON	APLSCSHV	
	APLYUSHV	
SCSVPINI	APLSCSVI	1.2.1, 8.4.3
SCSVQUER	APLSCSHV	
	APLYUSHV	
SCSVREF	APLSCSHV	
	APLYUSHV	
SCSVRETR	APLSCSHV	
SCSVSPEC	APLSCSHV	
	APLYUSHV	
SCSYSER	APLSCERR	1.2
	APLYUERR	1.4
SCTABS	APLSCTYP	1.2
	APLYUTYP	
SCTIME	APLSCMSC	
	APLYUMSG	
SCTRAN	APLSCMSG	
	APLYUMSC	
SCTYI	APLSCTYP	1.2

Entry Point or Routine Name	Module Microfiche Name	Method of Operation Diagram
	APLYUTYP	
SCTYO	APLSCTYP	1.2
	APLYUTYP	
SCTYOI	APLSCTYP	1.2
	APLYUTYP	
SCWIDTH	APLSCTYP	1.2
	APLYUTYP	
SCWSID	APLSCMSC	
	APLYUMSC	
SSATACH	APLSCTYP	
	APLYUSSH	
SSSROUTR	APLYUSSH	
SSSSVC	APLYUSSH	

ENTRY POINTS AND MODULE NAMES SORTED BY MODULE NAMES

Module Microfiche Name	Entry Point or Routine Name	Method of Operation Diagram
APL	APL	
APLACCBE	APLACCBE	
APLACDSL	APLACDSL	8.2.1
APLACHLP	APLACHLP	8.2.1
APLACMDX	APLACMDX	8.2.1
APLACNDP	APLACNDP	8.2.1
	APLACMDF	8.2.1
	APLAMODE	8.2.1
APLACOPY	APLACOPL	
	APLACOPY	
APLACPRM	APLACPRM	8.2.1
APLACPRO	APLACPRO	8.2.1
APLACQRY	APLACMER	8.2.1
	APLACQRY	8.2.1
APLACQUE	APLACQUE	8.2.1
APLACRCP	APLACRCP	8.2.1
APLACRDA	APLACRDA	8.2.1
	APLADSON	8.2.1
APLACRSA	APLACRSA	8.2.1
APLACSF	APLACSF	8.2.1
APLACXCM	APLACXCM	8.2.1
APLAD	APLAD	
APLADMSG	APLADMSG	8.2.1
APLADTTM	APLADTTM	
APLAESTK	APLAESTK	
APLAK	APLAUALT	
	APLAUCAE	
	APLAUNCO	
	APLAUPRO	
APLAKP	APLAUATN	
APLALINE	APLALINE	
	APLAPAGE	
APLAS	APLAUCAE	
	APLAUNCO	
	APLAUPRO	
APLASA	APLAUALT	
APLASCHD	APLAERRM	8.2
	APLAINIT	8.2
	APLAPASS	8.2
	APLATERM	8.2
	APLATYI	8.2
	APLATYO	8.2
	APLATYOI	8.2
	APLAXCMD	8.2
APLASP	APLAUATN	
APLAUSRX	APLAUSRX	
APLAY	APLAUCAE	
	APLAUNCO	
	APLAUPRO	
APLAYA	APLAUALT	
APLAYP	APLAUATN	
APLCCULL	CVCULL	
APLCDISP	CVDISP	
APLCFUNC	CVFUNC	
APLCGRUP	CVGRUP	
APLCIBNM	CVIBNM	
APLCINIT	CVINIT	6.0
APLCLEAR	CVLEAR	
APLCMISC	CVDATE	
	CVDIRE	
	CVIOER	
	CVPRTR	
	CVTIDY	
APLCOIBM	COIBM	
APLCPARM	CVPARM	
APLCRPRT	CVRPRT	
APLCSAVE	CVSAVE	
APLCSHIP	CVSHIP	
APLCSPIE	CVSPIE	
APLCTBCD	CVTBCD	
APLCVARB	CVVARB	
APLCWKSP	CVWKSP	
APLCWSFN	CVWSFN	
APLFXIIM	APLFXIIM	
APLIACHK	IACHK	
APLIACIR	IACIRCLE	
APLIADEC	IADECODE	4.1.3
APLIADOM	IADDOM	4.1.3
	IAMDOM	4.1.3
APLIAENC	IAENCODE	4.1.3
APLIAFOR	IADFORM	4.1.3
	IAMFORM	4.1.3
APLIAGFM	IAGFMT	
	IAGFMT2	
APLIAGOU	IAGOUT	4.1, 4.1.5
APLIAGRD	IAGRADE	4.1.3
APLIANAM	IAFCHNAM	
	IASFIND	

Module Microfiche Name	Entry Point or Routine Name	Method of Operation Diagram
	IAVALNAM	
APLIAPRD	IAIPROD	4.1.3
APLIAQFN	IAQCR	
	IAQDL	
	IAQEX	
	IAQFX	
	IAQNC	
	IAQNL	
APLIARED	IAREDU	
APLIAROT	IAREVARY	
	IAROTA	
APLIASCN	IASCAN	
APLIASHF	IAQSVC	
	IAQSVO	
	IAQSVQ	
	IAQSVR	
APLIASHV	IARTRACT	4.1.5
	IASCOPY	4.1.5
	IASHSPEC	4.1.5
	IASVOFF	4.1.5
	IASVON	4.1.5
	IAUNSHR	4.1.5
APLIASYV	IAHTSPEC	4.1.5
	IARTOI	4.1.5
	IASHADO	4.1.5
	IASYSPEC	4.1.5
	IASYSPST	4.1.5
	IASYSREF	4.1.5
	IATABREF	4.1.5
	IAUNSHAD	4.1.5
APLIATAK	IATKDP	
APLIATBC	IATOBCD	
	IATOBCD2	
APLIATRN	IABNM	4.1.5
	IADEAL	4.1.5
	IADSHARE	4.1.5
	IAEXECTE	4.1.5
	IAEXNAME	4.1.5
	IAEXSTCK	4.1.5
	IAFACTRL	4.1.5
	IAFLCL	4.1.5
	IAIROLL	4.1.5
	IALOG	4.1.5
	IAMSHARE	4.1.5
	IAPLFUN	4.1.5
	IAPOW	4.1.5
	IAQDSPEC	4.1.5
	IAQUADS	4.1.5
	IAQUADSA	4.1.5
	IARESIDU	4.1.5
	IAROLL	4.1.5
	IASHRPST	4.1.5
	IATIDY	4.1.5
APLIATRS	IAEXPR	
	IAFACT	
	IALOGR	
	IASQRT	
APLIATSP	IADTRAN	
	IAMTRAN	
APLIECMX	IACMX	
	IECMEX	
APLIEFCH	IADYB	
	IECONVR	
	IEDYB	
	IEGETNI	
	IEGETNL	
	IEGETNR	

Module Microfiche Name	Entry Point or Routine Name	Method of Operation Diagram
	IEGETV	
	IEGINITI	
	IEGINITL	
	IEGINITR	
	IESGETN	
	IESGETV	
	IESGINIT	
APLIEFNM	IEFUNN	4.1.1, 4.1.2
	IEGOGOMN	4.1.1, 4.1.2
	IEGOGOSC	4.1.1, 4.1.2
	IESUNFUN	4.1.1, 4.1.2
	IEUNFN	4.1.1, 4.1.2
APLIEFXR	IEDATTN	
APLIEIDX	IEINDB	4.1.4
	IEINDD	4.1.4
APLIEMND	IECHIX	4.1.3
	IEMONAD	4.1.3
APLIEPSI	IEEPSIOT	
APLIERHO	IACOMMA	
	IECOMMA	
	IECOPY	
	IERSHP	
APLIESCA	IEDYAD	4.1, 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.1.5
	IESCANG	4.1, 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.1.5
APLIESPA	IEFIND	3.2
	IEFREE	3.2
	IEGTSPAC	3.2
	IENAME	3.2
	IESFIND	3.2
	IESFREE	3.2
	IESNAME	3.2
	IESPACST	3.2
	IESYNN	3.2
APLIETAK	IETKDP	
APLIEXAR	IEABEND	4.1
	IELDSTK	4.1
	IESTOSTK	4.1
	IEXARCH	4.1
	IEXIT	4.1
APLIEXFR	IACAL370	
APLITCMC	ITCMCOPY	
	ITCMPCOP	
APLITCMD	ITSYSCMD	5.0
APLITCME	ITCMERAS	
	ITDELETE	
	ITNAMINI	
APLITCMF	ITCMFNS	
	ITCMFVG	
	ITCMGRPS	
	ITCMVARS	
APLITCMG	ITCMGROU	
	ITCMGRP	
APLITCMI	ITCMSI	
	ITCMSINL	
	ITUSASH	
APLITCML	ITCMCLEA	
	ITCMDROP	
	ITCMLIB	
	ITCMLOAD	
	ITCMQUOT	
	ITCMSAVE	
	ITCMWSID	
	ITCMWSSI	

Module Microfiche Name	Entry Point or Routine Name	Method of Operation Diagram
	ITLIBMSG	
	ITPRWSID	
	ITSAVWS	
	ITSHV	
APLITCMS	ITCMDOST	
	ITCMSTAC	
	ITCMSYMB	
APLITCMT	ITCMCONT	
	ITCMMSG	
	ITCMOFF	
	ITCMOPR	
APLITCPI	ITCOPIN	
APLITCPO	ITCMCOPO	
APLITERR	ITERRORS	4.2
APLITEX	ITEXECUT	4.0, 4.2
APLITFCH	ITFETCH	
APLITFDC	ITCLOSET	3.0, 3.1, 3.2
	ITFDCLOS	3.0, 3.1, 3.2
	ITFDKILL	3.0, 3.1, 3.2
	ITFDTSOF	3.0, 3.1, 3.2
APLITFDE	ITFDEDIT	3.0, 3.1
APLITFDN	ITFDNWLN	
APLITFDO	ITFDOPEN	3.0, 3.1
APLITFUN	ITEMPFUN	
APLITHDR	ITLINE0	3.0, 3.1, 3.2
APLITIDS	ITBLDID	3.2
	ITBLDQD	3.2
	ITSTSRCH	3.2
APLITINI	APLIINIT	
	ITSYSERR	
APLITINP	ITFORCOF	2.0, 3.0
	ITINPINI	2.0, 3.0
	ITINPUT	2.0, 3.0
	ITTYIZ	2.0, 3.0
APLITLXS	ITOKENIZ	3.2
APLITNCV	ITFDCVT	3.2
	ITININT	3.2
	ITNUMCVT	3.2
APLITPRL	ITPRLINE	3.1
APLITSUB	ITBFTYO	4.2
	ITCKALPN	4.2
	ITFNLNO	4.2
	ITLOUT	4.2
	ITPRFNLN	4.2
	ITPRINTC	4.2
	ITPRNAME	4.2
	ITPRNUM	4.2
	ITSQUIRT	4.2
	ITTIME	4.2
	ITTIMSUB	4.2
	ITTYERR	4.2
	ITXBLNL	4.2
APLITUSG	ITUSADE	
	ITUSAG	
APLKADEF	APLKADEF	1.3
APLKADSP	APLKADSP	1.3
	KABEXIT	1.3
	KADEPON	1.3
	KMACRO	1.3
APLKAGBL	APLKAGBL	1.3
APLKAHST	APLKAHST	
APLKAMIX	APLKAMIX	
APLKASON	APLKASON	1.3
APLKASTB	APLKPFAP	
	APLKPFOH	
	KABOOTS	1.3
APLKCVMD	KSPCMD	

Module Microfiche Name	Entry Point or Routine Name	Method of Operation Diagram
APLKDDOS	KSPDOS	
APLKDOPS	KDPCREG	1.3
	KINIEX	1.3
	KLGET	1.3
	KRSTEX	1.3
	KSETEX	1.3
	KTIMEREX	1.3
APLKEHCP	APLKEHCP	1.3
APLKEMGR	APLKEMGR	1.3
APLKIFIX	APLFXIIM	1.3
	APLKIFON	1.3
	KCQZ	
	KIFONEXT	1.3
	KPGMCHK	1.3
	KTOINTER	1.3
	KYYTOI	
APLKISVI	APLKISVI	1.3
APLKLIBA	KLALLOC	1.3
	KLDEALOC	
APLKLIBB	APLKLIBI	1.3
	APLKLIBT	1.3
APLKLIBC	APLKLUIT	
	APLKLUTM	
APLKLIBF	APLKLIBF	1.3
APLKLIBG	APLKLIBG	1.3
	APLKLIBR	
	KGLOAD	
	KGSAVE	
	KGUDIR	
	KGWDIR	
APLKLIBR	APLKLIBR	???
APLKLIBU	KCLEAR	1.3
	KCOPA	1.3
	KCOPI	1.3
	KCOPO	1.3
	KCOPZ	1.3
	KDROP	1.3
	KLIB	1.3
	KLOAD	1.3
	KPASS	1.3
	KSAVE	1.3
	KWSID	1.3
APLKLIBV	KGCFILE	1.3
	KGDFILE	1.3
	KGUFILE	1.3
	KGWLIB	1.3
	KLRDBITM	
APLKMSCA	KCDELAY	1.3
	KCDUMP	1.3
	KCQAI	1.3
	KCSYSER	1.3
	KCTIME	1.3
APLKMSCB	KCATOFF	1.3
	KCMBL	1.3
	KCQUOTA	1.3
	KCTABS	1.3
	KCTRAN	1.3
	KCWIDTH	1.3
	KYYOFF	
APLKSSUB	APLKSSUB	1.3, 1.3.1
	KCASE2Q	1.3, 1.3.1
	KCASE3Q	1.3, 1.3.1
	KCLEANUP	1.3, 1.3.1
	KFREESP	1.3, 1.3.1
	KGCOL	1.3, 1.3.1
	KGETSPAC	1.3, 1.3.1
	KIDSETUP	1.3, 1.3.1

Module Microfiche Name	Entry Point or Routine Name	Method of Operation Diagram
	KPOSTWAI	1.3, 1.3.1
	KPPSEARC	1.3, 1.3.1
	KPROCOFF	1.3, 1.3.1
	KRETSUB	1.3, 1.3.1
	KSEIZE	1.3, 1.3.1
	KSINGAL	1.3, 1.3.1
APLKSSVP	APLKSSR	1.3, 1.3.1
APLKTCTL	APLKTCTL	1.3
APLKTCWR	APLKTCWR	1.3
APLKTRAN	KTRTRAN	1.3
APLKTREQ	KTRFI	1.3
	KTRGD	1.3
	KTRGF	1.3
	KTRIN	1.3
	KTRRD	1.3
	KTRRT	1.3
APLKTRQO	KTRAL	1.3
	KTRCU	1.3
	KTRFA	1.3
	KTRFM	1.3
	KTRHC	1.3
	KTRRS	1.3
	KTRWR	1.3
APLKTSRV	KTSCHED	1.3
	KTSCLEAR	1.3
	KTSFCHK	1.3
	KTSFNDF	1.3
	KTSLINO	1.3
	KTSLOCID	1.3
	KTSLOCR	1.3
APLKVALD	KSPALD	
APLKVAUT	KSPAUT	7.0
APLKVCMD	KSPSCN	7.0
APLKVCPY	KSPCPY	7.0
APLKVDSI	KSPDSI	
APLKVDSO	KSPDSO	
APLKVEXC	APLKSPRG	7.0
APLKVEXP	KSPEXP	7.0
APLKVFMT	KSPFMT	7.0
APLKVIMP	KSPIMP	7.0
APLKVINT	KSPINT	7.0
APLKVLBI	KSPLBI	
APLKVLBO	KSPLBO	
APLKVMSG	KSPMSG	
APLKVOPS	KDPCREG	
	KDPFAB	
	KINIEX	1.3
	KLCLOS	1.3
	KLGET	1.3
	KLOPEN	1.3
	KLPUT	1.3
	KRSTEX	1.3
	KSETEX	1.3
	KTIMEREX	1.3
APLKVPIN	KSPPIN	
APLKVTPO	KSPTPO	
APLKVTRM	KSPTRM	7.0
APLOCULL	CVCULL	
APLODIRE	CVDIRE	
	CVGDIR	
APLODISP	CVDISP	
APLOFUNC	CVFUNC	
APLOGRUP	CVGRUP	
APLOIBNM	CVIBNM	
APLOINIT	CVINIT	6.0
APLOLEAR	CVLEAR	
APLOMISC	CVDATE	
	CVIOER	
	CVPRTR	
APLOPARM	CVPARM	
APLORPRT	CVRPRT	
APLOSAVE	CVSAVE	
APLOSHIP	CVSHIP	
APLOSLST	CVSLST	
APLOSPIE	CVSPIE	
APLOTBCD	CVTBCD	
APLOTIDY	CVTIDY	
APLOVARB	CVVARB	
APLOWKSP	CVWKSP	
APLOWSFN	CVWSFN	
APLPAPAB	APLPAPAC	1.1.1
	APLPAPOF	1.1.1
	APLPAPPR	1.1.1
	APLPAPRT	1.1.1
	APLPAPSF	1.1.1
APLPAPCD	APCREATE	1.1.1
	APDFN	
	APDROP	1.1.1
	APFILSIZ	1.1.1
	APIO	1.1.1
	APOPEN	1.1.1
	APPASSWD	1.1.1
	APSHARE	1.1.1
	APVIO	1.1.1
	PRDDIR	1.1.1
	PRDSEQ	1.1.1
	PWRITE	1.1.1
APLPAPFS	FSMBUZZ	1.1.1
	FSMFORMT	1.1.1
	FSMGET	1.1.1
	FSMHCOPY	1.1.1
	FSMMINT	1.1.1
	FSMMTYPE	1.1.1
	FSMREAD	1.1.1
	FSMRFORM	1.1.1
	FSMSETC	1.1.1
	FSMSUB1	1.1.1
	FSMSUB2	1.1.1
	FSMSUB3	1.1.1
	FSMWRITE	1.1.1
APLPAPGB	GDDMCRET	1.1.1
	GDDMSCTL	1.1.1
	GDDMSDAT	1.1.1
	GDDMSOFF	1.1.1
APLPAPGC	GDDMRCTL	1.1.1
APLPAPGD	FREESTOR	
	GDDX	1.1.1
	GDDXINIT	1.1.1
	GETSTOR	1.1.1
APLPCOAP	APLPCOAP	
APLPCOEX	APLPCENT	1.1
	ERENDEX	1.1
	ERSAVEAR	1.1
APLPFXIM	APLFXIIM	1.1
APLPLIBS	PCCLEAR	
	PCCOPA	
	PCDROP	
	PCLIB	
	PCLOAD	
	PCSAVE	
	PCWSID	
APLPMISC	PCATOFF	
	PCCMD	
	PCCOPI	

Module Microfiche Name	Entry Point or Routine Name	Method of Operation Diagram
	PCCOPO	
	PCCOPZ	
	PCOFF	
	PCPASS	
	PCQAI	
	PCQUOTA	
	PCQZ	
	PCTIME	
APLPSERR	ERMSGRTN	
	ERTIMDAT	
	PCDUMP	
	PCSYSER	
APLPSHVR	PCSACC	1.1.1
	PCSCOPY	1.1.1
	PCSOFF	1.1.1
	PCSOFFER	1.1.1
	PCSON	1.1.1
	PCSQUERY	1.1.1
	PCSREF	1.1.1
	PCSRET	1.1.1
	PCSSPEC	1.1.1
APLPTYIO	PCDELAY	
	PCDTYI	
	PCDTYO	
	PCDTYOI	
	PCMBL	
	PCRWAIT	
	PCTABS	
	PCTRAN	
	PCTYI	
	PCTYO	
	PCTYOI	
	PCWIDTH	
APLP126T	APLP126T	
APLQDISP	CVDISP	
APLQFUNC	CVFUNC	
APLQGRUP	CVGRUP	
APLQIBNM	CVIBNM	
APLQINIT	CVINIT	6.0
APLQLEAR	CVLEAR	
APLQMISC	CVDATE	
	CVFRTR	
	CVTIDY	
APLQPARM	CVPARM	
APLQRPRT	CVRPRT	
APLQSAVE	CVSAVE	
APLQSPIE	CVSPIE	
APLQVARB	CVVARB	
APLQWKSP	CVWKSP	
APLSCDPY	SCDPA2	
	SCDTYI	
	SCDTYO	
	SCDTYOI	
APLSCERR	SCDUMP	1.2
	SCSAVOFL	1.2
	SCSPIE	1.2
	SCSTAE	1.2
	SCSYSER	1.2
APLSCFID	SCFID	
APLSCFXI	APLFXIIM	1.2
	SCAPL	1.2
APLSCINI	SCENDAPL	1.2
	SCMICRO	1.2
APLSCLIB	SCDROP	
	SCLIB	
	SCLOAD	
	SCSAVE	
APLSCMSC	SCCLEAR	
	SCCMD	
	SCDELAY	
	SCOFF	
	SCPASS	
	SCQAI	
	SCQUOTA	
	SCQZ	
	SCTIME	
	SCWSID	
APLSCMSG	SCMBL	
	SCRWAIT	
	SCTRAN	
APLSCOPY	SCCOPA	
	SCCOPI	
	SCCOPO	
	SCCOPZ	
APLSCSHV	SCSVACC	
	SCSVCOPY	
	SCSVOFF	
	SCSVOFFR	
	SCSVON	
	SCSVQUER	
	SCSVREF	
	SCSVREFR	
	SCSVSPEC	
APLSCSSI	APLSCSSI	
APLSCSVI	APLSHPST	1.2.1
	APLXBSAB	
	APLXBSXT	
	APLXWSWP	
	ASVPSERV	1.2.1, 8.4.3
	SCSVPINI	1.2.1, 8.4.3
	SCEXTINY	
APLSCTYP	SCATOFF	1.2
	SCATTN	1.2
	SCTABS	1.2
	SCTYI	1.2
	SCTYO	1.2
	SCTYOI	1.2
	SCWIDTH	1.2
	SSATACH	
APLSHACC	APLSHACC	1.2.1
APLSHBPB	APLSHBPB	1.2.1
APLSHBVB	APLSHBVB	1.2.1
APLSHCPY	APLSHCPY	1.2.1
APLSHGET	APLSHGET	1.2.1
APLSHOFR	APLSHOFR	1.2.1
APLSHPUT	APLSHPUT	1.2.1
APLSHQRE	APLSHQUE	
APLSHREF	APLSHREF	1.2.1
APLSHRET	APLSHRET	1.2.1
APLSHSOF	APLSHSOF	1.2.1
APLSHSON	APLSHSON	1.2.1
APLSHSPC	APLSHSPC	1.2.1
APLSHSRD	APLSHSRD	1.2.1
APLSHSUB	APLSHSUB	1.2.1
APLXAC	APLXACSO	8.3
	APLXACSV	8.3
APLXAK	APLXAKSO	
	APLXAKSV	
APLXASD	APLXAINP	8.3
	APLXAMSG	8.3
APLXAYD	APLXAINP	8.3
	APLXAMSG	8.3
APLXDKMP	APLXDKMP	
APLXDUMP	APLXDUCL	

Module Micro-fiche Name	Entry Point or Routine Name	Method of Operation Diagram
	APLXDUMP	
	APLXDUOP	
APLXFSFL	APLXFINT	
	APLXFSFL	
	APLXFTRM	
APLXFYFL	APLXFINT	
	APLXFTRM	
	APLXFYFL	
APLXGCAT	APLXGCAT	
APLXGCHC	APLXGCHC	
APLXGCOM	APLXGCOM	
APLXGKON	APLXGKON	1.3
APLXGKR	APLXGKR	
APLXGKRQ	APLXGKRQ	1.3
APLXGKRR	APLXGKRR	1.3
APLXGKT	APLXGKT	1.3
APLXGKU	APLXGKU	1.3
APLXGS	APLXGS	
APLXGY	APLXGY	
	APLXGYON	8.1
	APLXGYRQ	8.1
APLXMKSG	APLXMKSG	
APLXMSSG	APLXMSSG	
APLXMYSG	APLXMYSG	
APLXPK	APLXPK	
APLXPY	APLXPY	
APLXSTAK	APLXBACK	
	APLXCALL	
	APLXSTAK	
APLXTRAN	APLXTRAN	
	APLXTREZ	
	APLXTRZE	
APLXVERS	APLXVERS	
APLXWKWP	APLXWKWP	
APLXWYWP	APLXWYWP	
APLYDAIR	APLYDAIR	
APLYUCMD	APLYUCMD	
APLYUCNV	APLYUCNV	
APLYUDPY	SCDTYI	
	SCDTYIO	
	SCDTYO	
APLYUERR	SCDUMP	1.4
	SCSAVOFL	1.4
	SCSPIE	1.4
	SCSTAE	1.4
	SCSYSER	1.4
APLYUEXC	APLYUEXC	
APLYUFXI	APLFXIIM	1.4
	APLYUFXI	1.4
APLYUHSH	APLYUHSH	
APLYUIIM	APLFXIIM	
APLYUINI	APL	1.4
	SCENDAPL	1.4
	SCMICRO	1.4
	SCOFF	1.4
APLYULIB	SCDROP	
	SCLIB	
	SCLOAD	
	SCSAVE	
APLYULNE	APLYULNE	
APLYUMSC	SCCLEAR	
	SCDELAY	
	SCPASS	
	SCQAI	
	SCQUOTA	
	SCRWAIT	
	SCTRAN	
	SCWSID	
APLYUMSG	SCMBL	
	SCTIME	
APLYURVC	APLYURVC	
	ASVPSRVC	
APLYUSCN	DMSSCND	
	DMSSCNN	
APLYUSHV	SCSRETR	
	SCSVACC	
	SCSVCOPY	
	SCSVOFF	
	SCSVOFR	
	SCSVON	
	SCSVQUER	
	SCSVREF	
	SCSVSPEC	
APLYUSSH	SSATACH	
	SSSROUTR	
	SSSSVC	
APLYUSVI	APLXBYAB	1.2.1, 8.4.1, 8.4.3
	APLXBYXT	1.2.1, 8.4.1, 8.4.3
	APLXWYWP	1.2.1, 8.4.1, 8.4.3
	ASVPSERV	1.2.1, 8.4.1, 8.4.3
APLYUTBL	APLYUTBL	
APLYUTIO	APLYUTIO	
APLYUTYP	SCATOFF	
	SCATTN	
	SCTABS	
	SCTYI	
	SCTYO	
	SCTYOI	
	SCWIDTH	
APLYUUSR	APLYUUSR	
APLYU100	APL100	
APLYU101	APL101	
APLYU102	APL102	1.4.1
APLYU111	APL111	
APLYU210	APL210	1.4.1
APL100	APL100	1.2.2, 1.4.1
APL100K	APL100K	1.3.2
APL100KO	APL100KO	1.3.2
APL101	APL101	1.2.2, 1.4.1
APL102K	APL102K	1.3.2
APL110	APL110	1.2.2
APL111	APL111	1.2.2, 1.4.1
APL120	APL120	8.4.1
APL121	APL121	
	OFF121X	
	RET121X	
APL121K	APL121K	1.3.2, 8.4.3
APL123	APL123	1.2.2, 1.4.1
APL123K	APL123K	1.3.2
APL124K	APL124K	1.3.2
APL125K	APL125K	1.3.2
APL126	APL126	8.4.2, 8.4.3
APL126T	APL126T	
APL132K	APL132K	1.3.2
APL139K	APL139K	1.3.2
ASVPSRVC	ASVPSRVC	

SECTION 5. DATA AREAS

INTERPRETER DATA AREAS

VS APL WORKSPACE

The VS APL processor uses as a data area an area of virtual storage called an active VS APL workspace. An active VS APL workspace contains VS APL functions (user programs), data values developed during function execution, VS APL processor transient data, and a communications area for use of the VS APL components. A saved VS APL workspace is that part of an active workspace that is transferred from virtual storage to a library when a user issues a SAVE or a CONTINUE command or when a line disconnect or force-off occurs.

Only the active workspace is immediately available to a user for program execution and modification. A saved workspace is activated (transferred from a library to virtual storage) when a user issues a LOAD command. A clear workspace (one that contains no functions or data values) is activated when a user issues a CLEAR command.

The minimum and maximum sizes of an active workspace are defined by the host system. Within these limits, the default size of a user's active workspace is defined by the installation. A user may modify the size of the active workspace when issuing a LOAD or CLEAR command. The size specified must be large enough to contain the functions and data values in the workspace to be loaded; it may not exceed the maximum defined by the installation.

An active workspace is functionally divided into eight areas. These areas, in low to high virtual storage address sequence, are shown in Figure 5 and in the sections that follow.

Area	Bytes
Buffer	1024
Executor transient area	268
Translator transient area	756
Interpreter transient area	240
Address table	Variable
Operation stack	Variable
Free space	Variable
R13 stack	1024

Figure 5. VS APL Workspace

Regardless of the size of the workspace, the first four areas and the last one are fixed in size. A user may increase or decrease the size of the address table with the SYMBOLS command and the size of the operation stack with the STACK command. These actions cause a corresponding decrease or increase in the size of free space.

In the following sections, the general function of each area and the format and use of some of the information are described. For a detailed description of the entire workspace, see "WSM" control block format. All symbolic names used in the following sections are as defined in the APLWSM macro.

BUFFER

This area is used to hold data being transmitted to and from the user's terminal. Output strings are built in WSMBUFF until it is full or until terminal input is required, then they are dumped to the terminal with a YYTYO service request. Input is placed in WSMBUFF as a result of a YYTYI service request. Copy data is transferred through WSMBUFF with YYCOPO and YYCOPI requests.

EXECUTOR TRANSIENT AREA

This area is used for communication between the executor and the translator/interpreter. The area extends from WSMSUPSW through WSMRSV03. It includes a save area for use by the executor (WSMSUPSW, WSMSURGS), pointers delimiting the active data in WSMBUFF (WSMCURSR, WSMBFPTR), parameter areas for service requests (WSMPARM1, WSMPARM2, WSMSVLRQ), and a save area for interpreter registers (WSMREGSV). WSMPCPSW contains part of the interpreter PSW when it is given control to handle a program check (see "Program Check On-Vectors" below). WSMNSI contains the restart address after any service request, or the rest of the program check PSW. WSMPTHPT always addresses the PERTERM control block. WSMWORK contains the offset to the R13 stack area. WSMWIDTH has the current terminal width setting.

TRANSLATOR TRANSIENT AREA

This area contains a 156-word scratch area (WSMXXX) and various control words and switches that are used primarily by the translator part of the VS APL processor. This area and the next one are described separately only because the microcoded exarch does not use any part of this area. The area extends from WSMFDTOG through WSMTOGXX.

Included in this area is information which controls the handling of program checks, the writing of the active workspace to a library, and the relocation of a workspace.

Program Check On-Vectors

Four types of program-check interrupts may be intercepted: fixed-point overflow, exponent overflow, fixed-point divide, and floating-point divide. An on-vector is a 4-word list of the addresses of routines to handle those interrupts.

Execution of the APLON macro causes the current on-vector information (two words at WSMON) to be saved at the specified location. Then the address of the specified on-vector is stored in WSMONADR; the offset to the current R13 stack level is stored in WSMONR13; and the program mask is set as specified and stored in WSMONSPM. In the program mask, exponent underflow and significance are always disabled; decimal overflow is always enabled; fixed-point overflow is enabled or disabled as specified.

Execution of the APLOFF macro restores WSMON and the program mask to their prior state (as they were before APLON was executed).

When one of the four interceptible program checks occurs, registers 12 through 15 are reset as they were when the APLON macro was executed, and control is passed to the routine whose address is in the corresponding on-vector element. If the on-vector element is zero (no intercept routine specified), a system error occurs.

While the translator is executing, the program mask and on-vector are set so that all four program checks cause a system error. While the interpreter is executing, the program mask and on-vector are set as a default so that fixed-point overflow causes a system error and the other three program checks cause a DOMAIN ERROR; some interpreter routines use the APLON and APLOFF macros to change and restore this default.

Saved Workspaces

In an active workspace, there is transient information; there may also be unused free space (unallocated block) and data in free space that has been discarded (inactive blocks). There is no need to save any of this information when the active workspace is transferred to a library. Before writing the workspace to disk, all inactive blocks are freed, and all active blocks are collected into the low-address end of free space. The remaining free space (if any) is the unallocated block.

The WSMFREEA control word contains the offset to the low-address end of the unallocated block. After the inactive blocks have been freed, the offset encompasses all of the active data in free space. The part of the active workspace written to disk begins at WSMFREEA and extends through the offset contained in it.

Workspace Relocation

A particular workspace may be transferred into any virtual storage location. Relocation may occur when a saved workspace is activated or when a swappable service request (exit to a VS APL executor routine) has been made. A workspace contains both relative and absolute addresses; the absolute addresses must be adjusted when the workspace is relocated.

Absolute addresses are contained in registers 13 and 14 (pointers to the R13 stack); saved registers 13 and 14 in all levels of the R13 stack; WSMFREEU (address of low end of unallocated block of free space); LADDR, RADDR, and ZADDR (argument and result addresses); and some address table and operation stack entries beginning at WSM1RELO. All of these are located in other areas of the workspace and are described more fully in subsequent sections.

When the VS APL processor receives control, register 11 (MR) contains the virtual storage address of the active workspace. When a clear workspace is activated, MR is simply saved in WSMOLDMR. In all cases, the relocation factor (difference between MR and WSMOLDMR) is computed before MR is saved. If the relocation factor is nonzero, it is applied to all absolute addresses in the workspace.

INTERPRETER TRANSIENT AREA

This area contains a 24-word scratch area (WSMEXTMP) and various control words and switches that are used primarily by the interpreter part of the VS APL processor including the microcoded exarch. The area extends from WSMASYNC through WSMMINUB. The scratch area is reserved for the exclusive use of exarch (whether microcoded or not).

Included in this area is information about the current operation (VS APL primitive function). The statement scan and syntax analysis routine of exarch (IESCANG) passes information about the operator and its arguments to operator routines (both exarch and appendage routines). The operator routines pass information about the result to the result-processing routine of exarch (IESCANG).

Current Operator

During statement scan and syntax analysis, all information about the current operator is collected into one word on the operation stack. Before an operator routine is called, IESCANG places this word in the WSMOPWD field. The operator itself is in the left half (OPBYTE0 and OPBYTE1) of the field. If the operator is neither indexed nor composite, it is duplicated in the right half of the WSMOPWD field. If the operator is indexed (either implicitly or explicitly), the index value is in the fourth byte (OPINDEX); the index bit (OPHASIND) is set; the explicit index bit (OPEXIND) is set if the index was explicitly specified; the fractional index bit (OPFRIND) is set if a nonintegral index was specified. The contents of WSMOPWD for composite functions (reduction, scan, inner product, outer product) is described in "Method of Operation" (Diagram 4.1.3: "Primitive Function Processing"). The format of operator codes is described under "Operators and Separators."

Argument Blocks

During statement scan and syntax analysis, the operator arguments are placed on the operation stack. Before an operator routine is called, IESCANG places the entry for the right argument in the right argument block (WSMRGETV) and calls the IEGETV routine to set up the argument block for fetching of data. If the operation is dyadic, the same thing is done for the left argument using the WSMLGETV block.

Each argument block is three words long. The first word (LVALUE or RVALUE) is used to hold argument elements as they are fetched. The next byte (DL or DR) contains descriptor bits PBITIMME, PBITPERM, DBITSYNO, and DBITAPVE (see "Primary Descriptor" below and "Format of Blocks in Free Space," later in this section). The next byte (DL1 or DR1) contains the argument shape and data type. The next halfword (NL or NR) contains the internal name of the argument if it has a remote value. The third word (LADDR or RADDR) is used to hold addresses of argument elements as they are fetched.

The operator routines use the information in the argument blocks to fetch argument elements. Data fetch routines IEGINITL, IEGINITI, IEGINITR, IEGETNI, IEGETNL, and IEGETNR may be used to do this. Exarch operator routines call the data fetch routines directly; appendage operator routines communicate with them through service routines IESGINIT and IESGETN or the APLGETN macro. See the prologues of IEGETV and the data fetch routines for additional information about the contents and use of the argument blocks.

Result Block

There is a third block (WSMRSULT) that has the same format as the argument blocks. Operator routines place the result in the second word of this block (RESULT) either as an immediate value or the internal name of a remote value. When used for this purpose, the contents of the other two words (ZVALUE and ZADDR) are irrelevant.

The entire block may be used as the result is developed. Operations that have a third argument (for example, subscripted assignment) use it in the same manner as the argument blocks.

Exarch/Appendage Communication

Before appendage routines return to exarch, they place a return code in WSMAFLGS that indicates how the result should be processed; the codes are defined in the APLWSM macro. The return code that indicates no special processing for the result (AFLG2OK) is preset by exarch.

The IASHRPST appendage routine provides several services involving shared or system variables. Before calling IASHRPST, exarch sets WSMAFLGS to indicate which service is required.

Interpreter/Translator Communication

Before the interpreter returns to the translator, it places a reason code in the fourth byte of WSMABTYP; the codes are defined in the APLIERRC macro. If the reason for exit is an error, the address of the point where the abnormal termination routine (IEABEND) was called is placed in WSMABLOC. This is of no interest to the translator, but is useful for diagnostic purposes.

ADDRESS TABLE

For each object (that is, for each function, group, named variable, temporary variable, etc.) in the workspace, the address table contains either the object itself or its address. The address table is a series of fullword entries extending from WSMATAAA through the address contained in WSMBDATS.

The operation stack (the area after the address table) may be considered as part of the address table. The two areas are used for different purposes, but the format of their entries is similar, and they both contain as entries workspace objects themselves or their addresses.

Internal and External Names

Each object in the workspace is known to the VS APL processor by an internal name. An internal name is a 16-bit offset from WSMATAAA to an address table (or operation stack) entry. In other words, the internal name of an object is its location in the address table. An internal name is always a multiple of four; it can be distinguished from other items because its rightmost two bits are zero.

Some objects are also known by an external name—the name given to a function or variable by the user. External names are never used by the interpreter. They are used by the translator in its input routine and when names are to be printed. External names are generally referred to as printnames.

Permanent and Temporary Objects

A permanent variable is one which has a printname. A permanent variable has two address table entries—one for the printname and one for the value assigned to that name. (For a further description of the permanent variable, see "Symbol Table.") A permanent variable is not discarded until the workspace is cleared. When a permanent variable is erased, its value block in free space is discarded and its second address table entry is set to indicate that the printname has no value; the first entry and the printname itself are unchanged.

A temporary variable is one that has no printname. Temporary variables result from user input and from the execution of primitive or defined functions. A temporary variable is discarded as soon as it is no longer needed; both its internal name (its address table entry) and its value block in free space are discarded. For example, when executing the statement

```
A ← 2 3 ρ ι 6
```

four temporary variables occur: **t1**, **t2**, **t3**, and **t4**. **t1** is the scalar 6. **t2** is the vector 2 3. **t3** is the result of the iota function; at its completion, **t1** is discarded. **t4** is the result of the rho function; at its completion, **t2** and **t3** are discarded.

The internal name t4 is discarded when its value is assigned to the permanent variable.

Functions are also either permanent or temporary. A permanent function is one defined by the user. As with permanent variables, a permanent function may be erased; but its printname is not discarded until the workspace is cleared. A temporary function is one that is built by the translator to implement immediate execution, quad input, or the execute primitive. A temporary function has one main statement—one line typed by the user in immediate execution, the response to quad input, or the argument of execute; it also has the branch-to-line-zero statement that is the last statement of every function. When execution of a temporary function is completed, both its internal name and its function block in free space are discarded.

Immediate and Remote Objects

An immediate object is one whose value is contained in (rather than addressed by) an address table or operation stack entry. Immediate entries are used for objects that have no shape and whose value can be represented in 16 bits or less: character, logical, or small integer scalars and one-character printnames. The format of an immediate object is shown in Figure 6.

Byte	Bits	Contents
0		Syntax class and primary descriptor
1	0	Sign bit of an integer value (ATIMSIGN)
	1	ON indicates variable is result of assignment (ABITASGN)
	2	Unused
	3	ON indicates a read-only object—a label (ATIMLBL)
	4	Unused
	5-7	Data type of the object: 100 (DBITCHAR) = character; 001 (DBITINTE) = integer; 000 (DBITLOGI) = logical.
2-3		Value, right-justified

Figure 6. Format of Immediate Object

There are a few immediate address table entries whose format is different than those described above (see "System Variables").

A remote object is one whose value is contained in free space. All functions, all groups, nonimmediate variables, and printnames are remote objects. Byte 0 of the address table entry contains the object's syntax class and primary descriptor; bytes 1 through 3 contain the absolute address of the DN-word (see "Free Space," later in this section) of the object's free space block. It is these entries that must be modified when the workspace is relocated.

When a remote object is placed on the operation stack, its relative rather than its absolute location is stored. Byte 0 of the operation stack entry for a remote object contains its syntax class and primary descriptor; byte 1, bit 1 is as described in Figure 4 (the rest of byte 1 is irrelevant); bytes 2 and 3 contain its internal name.

Syntax Classes

Bits 0 through 3 of byte 0 of all address table and operation stack entries define the syntax class. As noted in Figure 7, some syntax classes occur only on the operation stack.

Class	Symbol	Description
0	SBITNULL	In address table, an unused entry; on the stack, a null value or the beginning of a level
1	SBITOPER	Operator (stack only)
2	SBITVAR	Variable
3	SBITFUN2	Dyadic function
4	SBITRPBR	Right parenthesis or bracket (stack only)
5	SBITLPBR	Left parenthesis or bracket (stack only)
6	SBITSEMI	Semicolon (stack only)
7	SBITLARR	Left arrow (stack only)
8	SBITRBRO	Right operator index bracket (stack only)
9	SBITFUN0	Niladic function
A	SBITEND	End of statement (stack only)
B	SBITFUN1	Monadic function
C	SBITSHAR	Shared object (shared variable, system variable, and (on stack only) quad, quote-quad)
D		Unused
E		Unused
F	SBITSYST	System object (group, printname)

Figure 7. Syntax Classes

Primary Descriptor

For variables, functions, groups, and printnames (syntax classes 2, 3, 9, B, C, F), bits 4 through 7 of byte 0 of an address table or operation stack entry contain the object's primary descriptor as described in Figure 8.

Bit	Symbol	Description
4	PBITVALU	Object has a value
5	PBITIMME	Object is immediate
5	PBITABS	Object is not to be relocated
6	PBITPERM	Object is permanent
7	PBITINUS	In address table, entry is in use
7	PBITNAME	On stack, object is named (has an address table entry)

Figure 8. Primary Descriptors

The valid combination of syntax classes and primary descriptor bits is as described in Figure 9.

Value	Description
27	Object with no value
29	Remote, temporary variable
2B	Remote, permanent variable
2E	Immediate, temporary variable on operation stack (stack immediate)
2F	Immediate, permanent variable in address table
3B	Permanent dyadic function
99	Temporary niladic function
9B	Permanent niladic function
BB	Permanent monadic function
C0	Quad or quote-quad (stack only)
C7	Unused entry (reserved for system variable)
CB	Shared variable (always remote)
CF	System variable (entry is immediate although variable may not be so; see "System Variables")
FB	Group or remote printname
FF	Immediate printname

Figure 9. Combination of Syntax Classes and Primary Descriptor Bits

Address Table Sections

The address table is functionally divided into four sections.

RESERVED ENTRIES: The first 27 entries in the address table (from WSMATAAA up to WSMADTAB) are used for reserved temporary entries, default system variables, constants, and four control words.

The entries for default system variables and some of the constants are remote. The values addressed by these entries are in module APLITMSG rather than in the workspace. These entries precede WSM1RELO, and are not examined during workspace relocation.

The four control words are:

- WSMFUNCT: byte 0 = X'2F' (bit 0 = 1 if the current function is damaged); byte 1 = 0; bytes 2 and 3 contain the internal name of the function currently being executed.
- WSMNXINS: byte 0 = X'2B'; bytes 1 through 3 contain the absolute address of the next token in the function currently being executed.
- WSMTSADR: byte 0 = X'2B'; bytes 1 through 3 contain the absolute address of the top of the operation stack (see "Operation Stack")
- WSMBDATS: byte 0 = X'2B'; bytes 1 through 3 contain the absolute address of the last word of the address table (see "Address Table Management").

SYSTEM VARIABLES: The next 20 entries in the address table (from WSMADTAB to WSM1STNM) are used for system variables (quad-IO, quad-WA, etc.). These entries are all immediate (syntax/descriptor=X'CF'). Byte 1 contains various flag bits including one that indicates whether the value of the system variable is immediate or remote. See the APLWSM macro description in "Data Areas" under WSM control block, at symbol ATIMNOVL for a description of the flag bits.

An entry for a system variable that has an immediate value contains the value in bytes 2 and 3. An entry for a system variable that has a remote value contains the internal name of another address table entry in bytes 2 and 3. The referenced entry may be either the reserved one in the first part of the address table that addresses the default value of the system variable; or a temporary one that addresses the user-specified value in free space.

SYMBOL TABLE: The next part of the address table (beginning at WSM1STNM) is known as the symbol table and is reserved for permanent objects. Its length, in words, is twice the value of the SYMBOLS command. This value is contained in WSMSYMBL, a control word in the translator transient area.

Entries in the symbol table are used in pairs. The first entry of a pair contains or addresses the printname. The second entry contains or addresses the value assigned to the printname; it may contain no value, an immediate variable, or the address of a function block, group definition block, or remote variable block in free space.

Symbol table entries are selected at random by means of a hashing algorithm that uses the printname as input.

TEMPORARY ENTRIES: The remainder of the address table is used for temporary objects, both immediate and remote. The length of this part of the address table varies dynamically.

Address Table Management

Management of the address table is concerned only with the last part—that used for temporary objects. In this section, the term "address table" is used to mean the last part only of the full address table. There are three aspects of address table management: varying the size; getting an internal name; and freeing an internal name.

The address table and the operation stack are contiguous. In a clear workspace, the combined size of the two areas is 515 words—257 address table entries and 258 operation stack entries. However, the boundary between them (its address is maintained in WSMBDATS) is dynamic. The address table grows from low virtual storage up. When more address table space is required, half of the unused operation stack entries are allocated to the address table, and the address in WSMBDATS is incremented. The operation stack grows from high virtual storage down. When more stack space is required, half of the unused address table entries are allocated to the operation stack, and the address in WSMBDATS is decremented. In either case, if no space is available, a STACK FULL error occurs. These functions are performed by routines IAEXNAME and IAEXSTCK.

The size of the operation stack and, indirectly, the address table can also be varied by the user with the STACK command. Execution of the command causes the operation stack space to be increased or decreased; the active part of the stack to be moved down or up; and free space to be decreased or increased. WSMBDATS and the current size of the address table are not affected.

When examining a word between the bottom of the address table and the top of the operation stack, there are two ways of determining the area to which it belongs. The first is to compare the address of the word and the address in WSMBDATS; the second is to examine byte 0 of the word. Byte 0 of unused entries at the bottom of the address table is zero; the last entry is never used. Byte 0 of stack entries (used or not) is nonzero.

In the address table, in addition to entries that have never been used, there may be entries that have been used and freed. The latter are formed into a chain of available names that begins at WSMNXNMW (a control word in the interpreter transient area). The format of entries in the chain (including WSMNXNMW) is: byte 0 is X'04'; byte 1 is unused; bytes 2 and 3 is the internal name of the next entry in the chain.

An example of an available name chain is shown in Figure 10. When an internal name is requested, t5 is given and the name in the t5 entry is placed in WSMNXNMW; thus, t2 becomes the next available name. When another name is requested, t2 is given and t6 becomes the next available name. When a third name is requested, t6 is given; since t6 is not a link in the chain, the next sequential entry (t7) becomes the next available name. If the next sequential entry is not an address table entry, the address table is extended, if possible, as already described. The IENAME routine is called to get an internal name.

An internal name is freed by putting the next available name (the one in WSMNXNMW) in the freed entry, and then putting the freed name in WSMNXNMW.

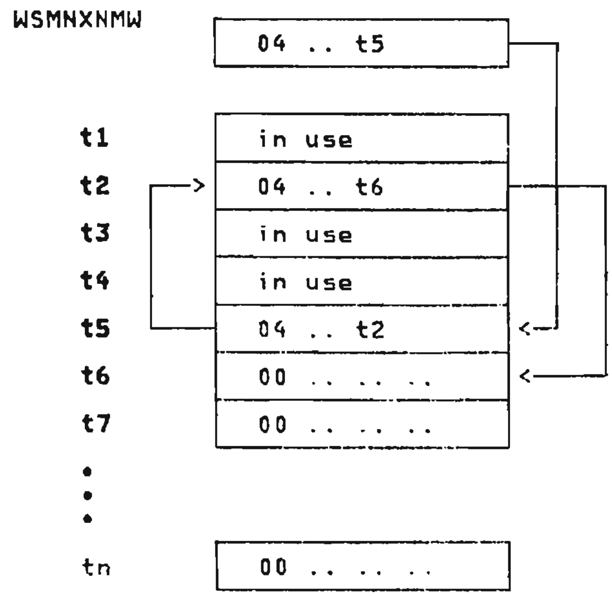


Figure 10. A Chain of Available Names

OPERATION STACK

The operation stack is a pushdown stack that is used to hold input to and output from interpreter routines as VS APL statements are scanned and executed. It is a series of fullword entries extending from the end of the address table to the beginning of free space. It grows from high to low virtual storage as shown in Figure 11. The address of the next available stack entry is maintained in WSMTSADR. The following entry, referred to as the top token on the operation stack, contains the last item put on the stack. The address in WSMTSADR is decremented as items are entered on the stack; it is incremented as items are taken off the stack.

Source of Operation Stack Entries

Tokenized function statements are the primary source of operation stack entries. The process of tokenizing a statement (converting an external statement to its internal form) is described in Diagram 3.2: "Function Definition." In its internal form, a function statement consists of a series of tokens in inverted external sequence. Each token is a halfword in length. There are four general classes of tokens: internal names, operators and separators, descriptors of literals, and special operators. The format of these tokens and how they are put on the stack are described in the following sections.

The input to the interpreter's statement scan and syntax analysis routine is always a function statement. The statement may be part of a permanent function (one defined by the user), a temporary function (immediate execution, quad input, or execute primitive), or an embedded VS APL function. An embedded VS APL function is one that is defined within the interpreter and is used to perform certain VS APL primitive functions (see "Method of Operations" Diagram 4.1.3: "Primitive Function Processing"). The body of an embedded VS APL function is contained in an interpreter module, rather than in a function block in free space. There is no difference in the format of the statements in the various types of functions. During statement scan and execution, the function type is irrelevant.

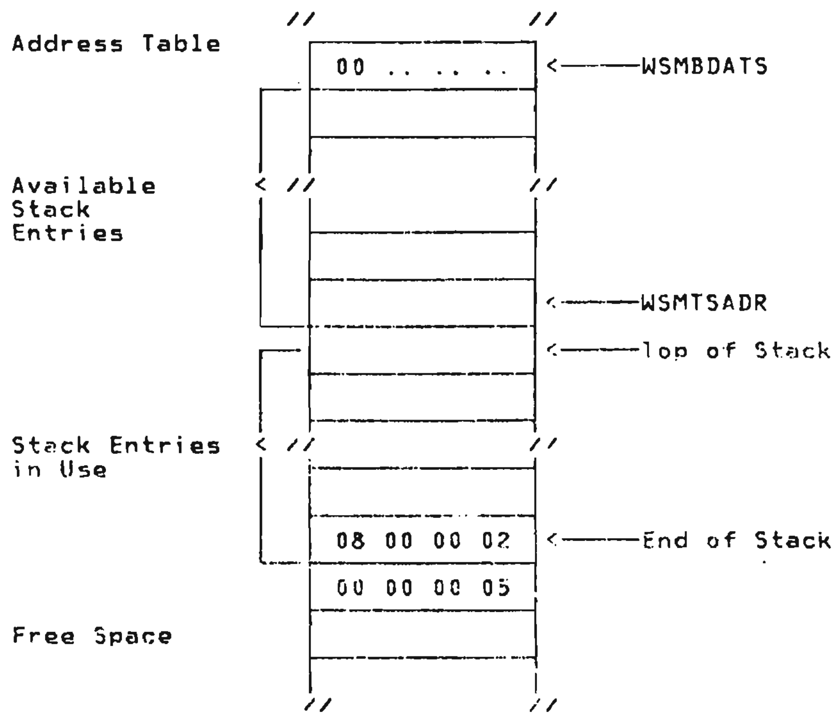


Figure 11. The Operation Stack

A second source of operation stack entries is the execution of primitive and defined functions. The result of execution may be a temporary variable, either remote or immediate; a temporary niladic function resulting from quad input or execute; or an embedded VS APL function.

Finally, there are certain operation stack entries that are generated internally: nulls, stop words, and function call blocks.

Use of the Operation Stack

During statement scan, WSMNXINS contains the address of a token in a function statement. The token is put on the operation stack in the entry whose address is in WSMTSADR. Then WSMNXINS is incremented so that it points to the next token, and WSMTSADR is decremented so that it points to the next available stack entry. The syntax classes of the top two tokens on the stack are analyzed to determine if there is some action to be performed. If not, the next token is fetched and stacked. When there is some action to be performed, the items on the stack are used as input. As a result of the action, items on the stack may be modified; items may be taken off the stack; a result may be put on the stack. At completion, the statement scan is resumed.

For example, in performing a dyadic operation, tokens are fetched and stacked until the operation stack is as described in Part A of Figure 12.

The appropriate dyadic operator routine is called. On return, the top three stack items are discarded, and the result is put on the stack as described in Part B of Figure 12.

A: Just before a dyadic operation

WSMTSADR——>	
	variable (left argument)
	operator
	variable (right argument)
	prior token

B: After the operation

WSMTSADR——>	
	variable (result)
	prior token

Figure 12. Tokens on the Operation Stack

The use of the operation stack is described in "Method of Operations" Diagram 4.1: "Statement Scan, Syntax Analysis, and Execution."

Items on the Operation Stack

The first four bits of every operation stack entry define the syntax class of the item; all syntax classes may appear on the stack. The remainder of the entry varies according to the type of item.

INTERNAL NAMES: The internal name of a function, group, or remote variable (rather than the object itself) is entered on the operation stack. Byte 0 contains the syntax class and primary descriptor; byte 1 is unused; bytes 2 and 3 contain the internal name.

An internal name appears in a function statement as a token whose rightmost two bits are '00'. The name is entered on the stack, and the syntax/descriptor are obtained from the address table.

When the result of executing a primitive or defined function is a remote variable or a function, its syntax, primary descriptor, and internal name are returned in the result block (WSMRSULT). The entry is moved from there to the operation stack.

LITERALS: A literal appears in a function statement as a descriptor token followed by the value of the literal. The rightmost two bits of the descriptor token are '10'. There are four types of literals that are distinguished by bits 12 and 13 of the descriptor token.

A General Literal: Is used for vectors. The format of the descriptor appears in Figure 13.

Bit	Description	
0-3	Shape	
	Value	Meaning
	0101	vector
4-7	Data type	
	Value	Meaning
	0000	logical
	0001	integer
	0011	real
	0100	character
8-11	Unused	
12-15	1010	general literal

Figure 13. General Literal Descriptor Format

The token following the descriptor contains the free-space byte count: length of values plus 12 for count word, DN-word, and element count. The tokens that follow contain the values. The last two tokens contain the element count. The values are padded to a full word, but are not necessarily aligned on a word boundary. The statement scan routine gets a temporary internal name and a block of free space. It enters the shape, data type, name, values, and element count in the block. It enters the internal name on the operation stack with a syntax/descriptor of X'29'.

A Scalar Literal: Is used for large integer and real scalar values. The format of the descriptor appears in Figure 14.

Bit	Description	
0-3	Shape	
	Value	Meaning
	0000	scalar
4-7	Data type	
	Value	Meaning
	0001	integer
	0011	real
8-11	Unused	
12-15	0010	scalar literal

Figure 14. Scalar Literal Descriptor Format

The two or four tokens following the descriptor contain the value. A scalar literal is entered on the operation stack as a temporary remote variable as described for general literals.

A Short Literal: Is used for logical, character, and small integer scalar values. The format of the descriptor appears in Figure 15.

Bit	Description	
0	Sign of integer value; else 0	
1-3	000	
4-7	Data type	
	<u>Value</u>	<u>Meaning</u>
	0000	logical
	0001	integer
	0100	character
8-11	Unused	
12-15	0110	short literal

Figure 15. Short Literal Descriptor Format

The token following the descriptor contains the value. A short literal is entered on the operation stack as an immediate value (described below).

An Invalid Literal: Is used to indicate a value that is too large or too small to be represented. Bits 12 through 15 of the descriptor are 1110. When an invalid literal is encountered, a VALUE error exit is taken.

IMMEDIATE VALUES: The value of a temporary immediate variable is placed directly on the operation stack; it does not appear in the address table. Such an item is referred to as a stack immediate value. Byte 0 contains the syntax class and primary descriptor (X'2E'); byte 1 contains the sign bit and data type; bytes 2 and 3 contain the value. Note that the format is the same as that of an address table immediate entry except that the primary descriptor is 'E' rather than 'F'.

A stack immediate value is built when a short literal is found in a function statement or when the result of executing a primitive function is an immediate variable.

OPERATORS AND SEPARATORS: An operator (VS APL primitive function) or separator appears in a function statement as a token whose rightmost two bits are '01'. The token is duplicated in bytes 0 and 1 and 2 and 3 of an operation stack entry.

The bit patterns of individual operators is such that the operators fall into various functional groups. The meaning of the operator bits appears in Figure 16.

Bit	Symbol	Description
0-3	SBITOPER	Syntax class (0001)
4	OPEQNE	1 = dyadic operator is "equal" or "not equal".
4	OPTEMPGO	1 = monadic operator is "right arrow" entered by user in a temporary function
5	OPRED	1 = operator may be part of a composit operator (reduction, scan, inner product, outer product)
6	OPHASIND	1 = operator has implicit index of 0 (is overstruck with a hyphen). This bit is also set subsequently by the interpreter if the operator is explicitly indexed.
7	OPISMIX	0 = scalar operator (result shape same as argument shapes) 1 = mixed operator
8		No functional significance
9	OPINDBL	1 = operator may be indexed. This bit is also set by the interpreter (using the symblic name OPREAL) when a real floor or ceiling is required.
10-11	OPGRP	Defines class of scalar operators:
	OPCOMPR	00 = comparison
	OPLOGGR	01 = logical
	OPSARTH	10 = simple arithmetic (done as either integer or real according to argument type)
	OPCARTH	11 = complex arithmetic (generally done as real regardless of argument type)
12-13		No functional significance
14-15		Always 01

Figure 16. Operator Bit Meanings

Figure 17 shows the hexadecimal representation of all operators.

1009 <	1031 *	10B1 ●	1181 ⍉	1555 /
100D ≤	1035 ?	10B9 ÷	1185 ↓	1591 .
1011 ∨	1039 ○	1101 ⍴	1189 ∊	15D5 \
1015 ∧	1089 ≥	1105 ↑	118D ⊤	1755 ⌿
1019 ⍱	108D >	1109 ⍳	119D ⍒	17D5 ⍀
101D ⍲	1095 →[1]	110D ⊥	11A1 ⌹	1805 =
1021 +	109D ~	1111 ∘	11A9 ⍕	1885 ≠
1025 ×	10A1 -	111D ⍋	11BD ⌷[1]	1895 →[2]
1059 ⌈	10A5 |	1129 ⍎	11D9 ,	C001 ⎕[3]
102D !	10A9 ⌊	1159 ⌽	1259 ⊖	C005 ⍞[3]

Notes:

1. Entered in a permanent function.
2. Entered in a temporary function.
3. Used to identify a system function; is encoded by the interpreter, not by the user; see "Special Operators."
4. Cannot properly be called operators, since syntax class is shared object. However, the rightmost two bits place them in the class of operators and separators, and they are included here for reference.

Figure 17. Operator Hexadecimal Representations

The hexadecimal representation of separators appears in Figure 18.

Value	Description
4001	Right parenthesis
4005	Right bracket (subscripting) as encoded by the translator
4405	Right bracket as modified by the interpreter to indicate subscripted assignment
4C05	Right bracket as modified by the interpreter to indicate subscripted assignment to a shared or system variable
5001	Left parenthesis
5005	Left bracket (subscripting or operator (index) as encoded by the translator
500D	Left bracket as encoded in an embedded VS APL function to allow a scalar or array to be subscripted as if it had been revelled
6001	Semicolon
6201	Empty subscript marker; generated by the interpreter to indicate an omitted subscript
7101	Left arrow (assignment)
8005	Right operator index bracket
A0x1	End of statement (EOS), generated by the translator as the last token of every function statement. Bit 10 (EOSTPBIT) is 1 if the if the stop vector contains the number of the next statement. Bit 11 (EOSTRBIT) is 1 if the trace vector contains the number of this statement. Bit 11 is always on in the EOS token of the main statement of a quad-input or execute temporary function.

Figure 18. Separator Hexadecimal Representations

SPECIAL OPERATORS: A special operator appears in a function statement as a token whose rightmost two bits are '11'. There are five types of special operators distinguished by bits 11 through 13 of the token.

A Fast Branch Special Operator: Is encoded by the translator when the argument of the branch is input as a positive integer scalar. The fast branch operator is also used for the branch to zero which the translator generates as the last statement of every function. The format of the fast branch token appears in Figure 19. The fast branch is put in bytes 2 and 3 of an operation stack entry; bytes 0 and 1 are set to X'1000'.

Bit	Description
0	OPTEMPGO (see "Operators and Separators")
1-11	Target statement number (argument of branch)
12-15	0011

Figure 19. Fast Branch Special Operator Format

An Escape Special Operator: Is encoded by the translator when it encounters an ill-formed statement or assignment to a stop or trace vector. The escape token for an ill-formed statement is X'0007'. The next token contains the error code (ABSYNT or ABDOMA). The next two tokens contain the byte count of the ill-formed statement. The following tokens contain the statement text as entered. The escape token for assignment to stop and trace vectors is X'ccF7' where cc is the Z-code for S or T. The next token contains the internal name of the function. The escape special operator is put in bytes 2 and 3 of an operation stack entry; byte 0 is set to X'2E'; byte 1 is unused. The following tokens are processed by the translator rather than by the interpreter, and they are not entered on the operation stack.

A Skip Special Operator: Is encoded by the translator when it encounters a comment. The skip token is X'001B'. The next token contains the byte count of the comment plus four. The following tokens contain the comment text. The skip token is not entered on the operation stack. The count token is used to increment WSMNXINS, and the statement scan is resumed with the token following the comment.

An Indirect Special Operator: Only in embedded VS APL function statements. The indirect operator token is X'000B'. The next token is the internal name of a scalar operator. The rightmost operator byte is obtained from the address table entry and catenated to X'10' or to X'18' if the operator is equal or not-equal. The resultant halfword is put in bytes 0 and 1 and 2 and 3 of an operation stack entry.

A Secondary Decode Special Operator: Is encoded by the translator when it encounters the external name of a system function (quad-EX, quad-NL, etc.). The format of the secondary decode token appears in Figure 20.

Bit	**Description**
0-7	Internal code that identifies the system function
8-11	Flag bits that classify the system function
12-15	1111

Figure 20. Secondary Decode Special Operator Format

The secondary decode special operator is put in bytes 2 and 3 of an operation stack entry; bytes 0 and 1 are set to X'11BD' (quad-q operator).

FUNCTION CALL BLOCK (FCB): A function call block is used to save information about the state of the workspace when a function is invoked. At function exit, the information is used to restore the workspace to its prior state. The information that is saved is the current value of the called function locals, the internal name of the calling function (that is, the currently active function), and the location within the calling function of the token following the function call.

An FCB is built on the operation stack when a permanent function, a quad-input or execute temporary function, or an embedded VS APL function is invoked (see Diagram 4.1.1: "Function Call and Function Exit Processing"). There is no way in which immediate execution statements can be nested or invoked, hence an FCB is not built for immediate execution temporary functions.

An FCB is removed from the stack at function exit (see Diagram 4.1.1: "Function Call and Function Exit Processing") or when an error occurs in a temporary function or a locked permanent function or when a branch with no argument is entered (see Diagram 4.2: "Return Code Processing").

The length of an FCB varies according to the number of local variables and labels; its minimum length is ten words. Its format from top to bottom as it appears on the stack appears in Figure 21.

Contents	Meaning
0F..kkkk	k = length of FCB in bytes (40 + 8 * number of locals)
2Fxx0002	Marks end of variable entries. Byte 1 contains translator flags: <u>Equate</u> <u>Meaning</u> X'10' locked function X'30' embedded VS APL function X'01' quad-input temporary function X'02' execute temporary function Three or more pairs of variable entries as follows:
aaaaaaaa	a = copy of variable's address table entry; X'27000000' if a dummy entry
2F..nnnn	n = variable's internal name; rightmost bit is 1 if dummy entry
• • •	
0F..cccc	c = internal name of calling function. If the function is subsequently damaged, bit 0 is set to 1.
0F..iiii	i = offset within calling function of token following function call (displacement from DN-word of function block in free space)

Figure 21. Function Control Block (FCB) Format

Following the first two entries is a pair of entries for (in sequence) each label, each local variable, right argument, left argument, result; that is, for each entry in the called function header, FHEDLOCLn through FHEDZ. A dummy entry in the function header results in a dummy entry in the FCB; these occur when the function has no result, right argument, left argument, or when local names are duplicated.

STACK LEVELS AND STOP WORDS: At any point during execution, the operation stack is subdivided into one or more levels. A level is the set of operation stack entries that define the state of a function whose execution has not been completed. Thus, there is a stack level for the current function, for each pendant function (one which has invoked a function), and for each suspended function (one whose execution has been suspended because an error occurred, because attention was signaled, or because of a stop request).

Going from the top of the stack down, each level except the active current one begins with a stack entry whose first five bits are '00001'. Thus, the top entry in an FCB delimits a level. The other type of entry that delimits a level is a stop word. A stop word itself is a one-entry level. When a function is suspended, any current statement tokens that have been stacked but not yet executed are discarded, and a stop word is put on the stack. The bottom entry on the stack is always a stop word; it delimits the stack itself, rather than a level.

Figure 22 shows the contents of the operation stack, level by level, after the following events have occurred (the term "scan block" is used to identify a series of statement tokens that have been stacked but not yet executed). The user types a statement that is formed into a temporary function T1. T1 calls function AAA; statement 5 of AAA calls function BBB; and an error occurs in statement 7 of BBB. When the keyboard unlocks after the error message, the user types a statement that is formed into a temporary function T2. T2 calls function CCC, and statement 8 of CCC is now being executed.

```
WSMTSADR
 ┌──────────────────────────────────┐
 │ Scan block for CC[8]             │
 └──────────────────────────────────┘

 ┌──────────────────────────────────┐
 │ ICB for call of CCC              │
 │ -------------------------------- │
 │ Scan block for T2[1]             │
 └──────────────────────────────────┘
 ┌──────────────────────────────────┐
 │ Stop word for BBB[7]             │
 └──────────────────────────────────┘
 ┌──────────────────────────────────┐
 │ FCB for call of BBB              │
 │ -------------------------------- │
 │ Scan block for AAA[5]            │
 └──────────────────────────────────┘
 ┌──────────────────────────────────┐
 │ FCB for call of AAA              │
 │ -------------------------------- │
 │ Scan block for T1[1]             │
 └──────────────────────────────────┘
 ┌──────────────────────────────────┐
 │ End of stack stop word           │
 └──────────────────────────────────┘
```

Figure 22. Operation Stack Levels

When a function is pendant, the restart information (the internal name of the function and the address of the next token) is contained in the FCB in its level. When a function is suspended, the restart information is contained in the stop word. The format of a stop word for a suspended function appears in Figure 23.

The end-of-stack stop word is X'08000002'. Note that the top entry in an FCB can be distinguished from a stop word because its last two bits are always '00'.

NULLS: A null is X'07000000'. A null is always put on the operation stack as the first entry in a level. Its purpose is to serve as the prior token when just one statement token has been stacked and the syntax classes of the top two tokens are analyzed.

Operation Stack Management

When a token is put on the stack, the address in WSMTSADR is decremented by four. If the new address is less than or equal to that contained in WSMBDATS, or if byte 0 of the entry pointed to is 0, an attempt is made to extend the stack as described in "Address Table Management" earlier in this section.

Bit	Description
0-4	Level identifier (00001)
5-15	Statement number at which execution is suspended
16-29	Bits 0-13 of internal name of suspended function
30-31	Status of suspended function:

Code	Meaning
01	Damaged
11	Good

Figure 23. Suspended Function Stop Word Format

Items are taken off the stack by incrementing the address in WSMTSADR by a multiple of four. The stack itself is not modified.

FREE SPACE

Free space extends from the bottom of the operation stack to the beginning of the R13 stack. It contains the values of the remote objects in the workspace—variables, functions, printnames, groups. Free space is divided into blocks of words. There are four types of blocks—dummy, unallocated, inactive, and active.

Format of Blocks in Free Space

Each block of free space begins and ends with a count word. The interior of a block varies depending on the type of block and type of object. A count word contains the length of the block in bytes including the length of one of its count words. The rightmost two bits of a count word are used as a block type flag:

Bits	Meaning
00	Inactive
01	Active or dummy
10	Unallocated

Thus a 100-word block contains 98 interior words; the value of each count word is:

Value	Meaning
396	For an inactive block
397	For an active block
398	For the unallocated block

The format and purpose of dummy, unallocated, and inactive blocks are described under "Free Space Management." All active blocks have a common second word known as the "DN-word." Bytes 0 and 1 of a DN-word contain the object's descriptor. The bit meanings appear in Figure 24.

Bit	Symbol	Description
0-3		Not used; always 0
4	DBITAPVE	1 = object is an arithmetic progression vector; bit 8 also = 1
5-6		Not used; always 0
7	DBITSYNO	1 = object is a synonym; bit 8 also equals 1
8	DBITESCA	1 = object is a special case: bit 4 or bit 7 also equals 1
9-11		Shape descriptor:
	DBITSCAL	000 = scalar
	DBITVL1	001 = vector containing one element
	DBITARRY	x1x = array of any length; bit 9 and/or bit 11 also equals 1
	DBITARL1	011 = array containing one element
	DBITVECT	101 = vector, length not 1; either an empty or a multi-element vector
	DBITARRA	111 = array, length not 1; either an empty or a multi-element array
	DBITNSCA	1xx = object is neither a scalar nor a pseudoscalar (a one-element vector or array)
12		Not used; always 0
13-15		Data type descriptor; corresponds to bits 13 through 15 of an immediate address table entry:
	DBITLOGI	000 = logical
	DBITINTE	001 = integer
	DBITREAL	011 = real
	DBITNUM	0x1 = numeric of some sort
	DBITCHAR	100 = character

Figure 24. DN-Word Bit Meanings

Bytes 2 and 3 of a DN-word contain the object's internal name. The address of the object's address table or operation stack entry is obtained by adding the internal name and the value of WSMATAAA.

The address contained in a remote address table or operation stack entry is that of the DN-word of the object's active block.

The remaining words in an active block for various types of objects are described in the following sections. In the descriptions, the conventions appear in Figure 25.

The reason for the padding is that free space blocks are aligned on a doubleword boundary to speed up the fetching and storing of real values. The padding occurs, when necessary, to fill out a block to a multiple of eight bytes.

Code	Meaning
CCCC	Denotes a count word
DDNN	Denotes a DN-word
x... ...x	Denotes an item (x) that occurs any number of times, always as some number of fullwords
UU	Denotes an unused halfword
XXXX	Denotes an unused word of padding that may or may not occur

Figure 25. Active Block Descriptor Conventions

ORDINARY VARIABLES: An ordinary variable is one that is neither an arithmetic progression vector nor a synonym. The format of an active block for various shapes of ordinary variables appears in Figure 26.

Variable	Format
integer scalar	CCCC DDNN VVVV CCCC
real scalar	CCCC DDNN VVVV VVVV XXXX CCCC
vector	CCCC DDNN V... ...V XXXX NELM CCCC
array	CCCC DDNN V... ...V XXXX R... ...R RANK NELM CCCC

Figure 26. Active Block Format of Variables

The element values (V) are stored in the word(s) following the DN-word. Integers are stored in raveled sequence, one word per element. Real values are stored in raveled sequence, two words per element. Characters are stored in raveled sequence, one byte per element, and may be padded on the right with undefined bytes to complete a word. Logical values are stored one bit per element; the bytes are in raveled sequence, but the bits within a byte are reversed; the byte containing the last element may be padded on the left with undefined bits. Thus, the elements of a 19-element logical vector are stored in one word in this sequence with an undefined fourth byte:

7 6 5 4 3 2 1 0 15 14 13 12 11 10 9 8 x x x x x 18 17 16

For vectors and arrays, the element count (NELM) is stored as a fullword. For arrays, the rank (RANK) is stored as a fullword; the dimension vector (R...) is stored one fullword per dimension. An empty vector or array has an element count of zero and no value words.

The elements of an ordinary variable are accessed by stepping forward from the DN-word or beginning count word. The shape and size information is accessed by stepping backward from the ending count word.

ARITHMETIC PROGRESSION VECTOR: An arithmetic progression (AP) vector is a vector of integers that form an arithmetic progression. For example,

1 2 3 4 5

10 13 16 19 22 25 28

17 3 -11 -25

Any AP vector may be represented in a compressed form: initial element, step between elements, number of elements. An AP vector is represented in free space by a six-word block:

CCCC DDNN INIT STEP NELM CCCC

The translator does not examine vectors to determine if they are arithmetic progressions. AP vectors are generated by the monadic iota operator routine when the argument is greater than one. They are preserved across many operations such as addition or subtraction of a scalar, multiplication by a scalar, take, and drop.

Storing AP vectors in their compressed form saves space. It also saves processing time for operations with AP vector arguments.

SYNONYMS: Synonyms are variables whose value and size are the same; their shape is usually, but not necessarily, the same. To save space, the value, shape, and size are stored just once in a value block whose format is as described for ordinary variables; a temporary internal name is obtained for this value block. The address table entry for each synonymous variable contains the address of a synonym block. The synonym blocks are formed into a chain, and each one points to the value block. The format of a synonym block is:

CCCC DDNN UUVV PPSS XXXX CCCC

VV is the internal name of the value block; PP is the internal name of the predecessor in the chain; SS is the internal name of the successor in the chain. Since the rightmost two bits of an internal name are always 0, the rightmost bit of PP is set to 1 in the first synonym of a chain, and the rightmost bit of SS is set to 1 in the last synonym of a chain. In practice, the start of chain and end of chain are indicated by a value of -1 (X'FFFF').

Synonyms may be set up in the following cases: assignment, ravel of an array, invocation of a monadic or dyadic function (the arguments are copied), during statement scan and syntax analysis (see Diagram 4.1: "Statement Scan, Syntax Analysis, and Execution"). A synonym is made only if the argument is permanent and its value block is large (in practice, more than 40 bytes long). If the argument is temporary, its value block is simply used for the result. If the argument is permanent and small, a copy of it rather than a synonym is made for the result. The making of synonyms and copies is done by the IESYNN routine.

Figure 27 shows the setting up and extending of a synonym chain that occurs when the following statement is executed.

```
A←,B←C←3 4⍴⍳12
```

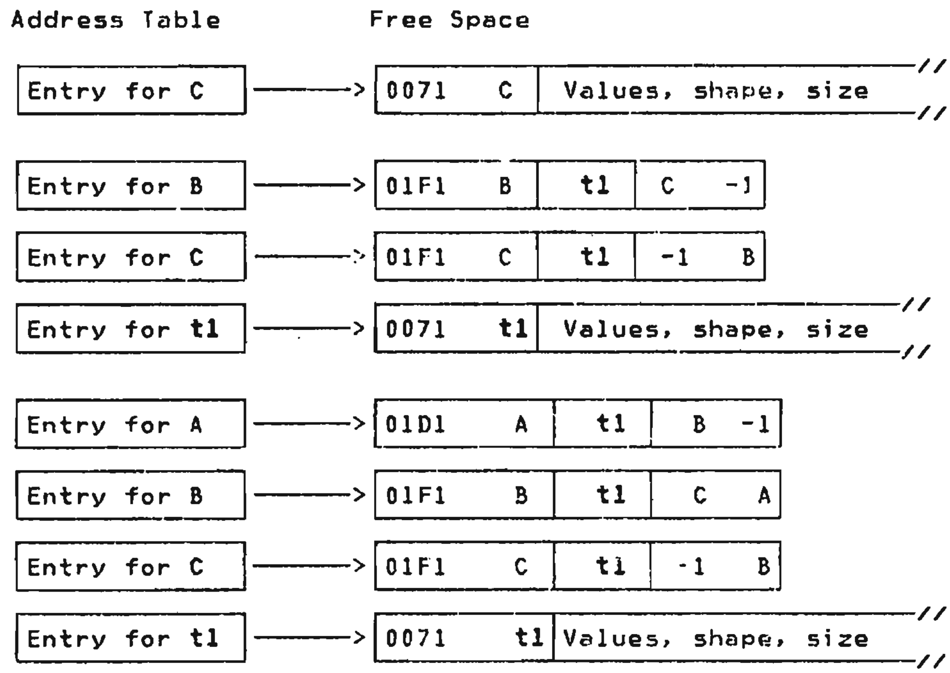


Figure 27. A Synonym Chain

Note that the descriptor in each free space block is shown in hexadecimal; the remaining information is shown symbolically and the count words are omitted for simplicity.

The top part shows the address table and free space following assignment to C; at this point C is an ordinary variable.

The middle part shows the new synonym chain following assignment to B.

The bottom part shows the extended synonym chain following the ravel of B and assignment to A and illustrates that the shape descriptors in the synonym blocks and value block are different if the ravel of an array has occurred. A is correctly described as a vector in its synonym block; the dimension and rank information in the value block is ignored when A is accessed. B and C are correctly described as arrays in their synonym blocks. When synonyms are accessed, the shape descriptor in the value block is ignored.

When a synonym is freed, it is removed from the chain by modifying the synonym blocks of its predecessor and successor. When a synonym chain is reduced to one link, the remaining variable is made an ordinary variable by discarding its synonym block and the internal name of the value block; putting the address of the value block in the variable's address table entry; and putting the variable's descriptor and internal name in the value block's DN word.

Although a value block has an entry in the temporary part of the address table, the entry is flagged as permanent. This is done to prevent the internal name and the value block from being discarded during recovery from a user error.

SHARED VARIABLES: The address table entry for a shared variable contains the address of a share ID block. The share ID block points to a value block. The format of the value block is as described for an ordinary variable. As with synonyms, the value block has a temporary internal name, but its address table entry is flagged as permanent.

The format of a share ID block is:

```
CCCC DDNN UUVV IIII XXXX CCCC
```

VV is the internal name of the value block. IIII is the offer sequence number (also known as the share ID number); this is an integer that uniquely identifies the variable to the shared storage manager.

GROUPS: The format of the free space block for a group is:

```
CCCC DDNN CTMM MM.. ..MM XXXX CCCC
```

The block is described as a character vector (X'0054'). The first halfword following the DN-word (CT) is a count of the number of members. The following halfwords (MM..) contain the internal names of the members.

PRINTNAMES: The format of the free space block for a printname is:

```
CCCC DDDD CV.. V XXXX CCCC
```

The block is described as a character vector (X'0054'). The first byte following the DN-word (C) is a count of the number of characters in the printname less one. The following bytes are the printname as a character string; there may be undefined bytes on the right to complete a word.

FUNCTIONS: The interior of a function block in free space is divided into three sections—head, body, tail. In the following description, the term "offset" means the displacement from the DN word of the function block.

The function header is a series of halfwords, as described in Figure 28. The symbols are as defined in the FHED macro (see Data Areas "FHED"). Since the rightmost two bits of an internal name are always 00, bit settings of 01, 10, 11 are used to indicate something other than a name.

The body of the function block contains the internal text of the function statements. Where Sn indicates the text of statement n and EOSn indicated the end-of-statement token for statement n, the body contains:

```
EOS0 S1  EOS1 S2  EOS2 Sn  EOSn branch-to-zero EOSx
```

The last two tokens of the body are those generated by the translator to enable function exit when the last statement (Sn) has been executed.

The first part of the tail of the function block contains the offset to each end-of-statement token in the body—EOS0 through EOSx. Each offset is a halfword. Following the offsets, there may be a word of padding. The next word (RANK if this were an ordinary variable) contains a value that is equal to or greater than the number of bytes used when displaying the longest statement in the function. The next and last word (NELM if this were an ordinary variable) contains:

Bytes	Contents
0-1	A count of the number of labels
2-3	Offset to the first label in the head of the function block

Halfword	Symbol	Description
0-1	FHEDDN	DN-word; descriptor is X'0054'
2	FHEDM	Number of last statement defined by user
3	FHEDT	Offset to tail
4	FHEDS	Translator flags:
	FHEDLOCK	X'10' = locked function
	FHEDMAG	X'30' = embedded VS APL function
	FHEDQUAD	X'01' = quad-input temporary function
	FHEDEXEC	X'02' = execute temporary function
		X'00' = immediate execution temporary function or unlocked permanent function; distinguished by primary descriptor in address table entry
5	FHEDK	Byte count for function call block (FCB); equal to 40 plus 8 times the number of locals including labels.
6	FHEDZ	Internal name of result; rightmost bits = 01 if none
7	FHEDL	Internal name of left argument; rightmost bits = 01 if none
8	FHEDR	Internal name of right argument; rightmost bits = 01 if none
9-n	FHEDLOCL	First, the internal name of each local variable in the sequence defined by the user in the function header; if name is a duplicate of a prior name, rightmost bits = 01. Then, a pair of halfwords for each label in statement number sequence. The first halfword contains the statement number in bits 0 through 13; rightmost bits = 11. The second halfword contains the internal name of the label.
m		Value of X'0002' (rightmost bits = 10) indicates end of locals.

Figure 28. Function Header

Free Space Management

In a clear workspace, free space contains a dummy block at each end and an unallocated block in the middle. A dummy block looks like an active block with no interior; it consists of contiguous count words whose value is 5. The dummy blocks delimit free space; their use is described later. The boundaries of free space are contained in a pair of control words in the interpreter transient area: WSMFREES contains the offset from WSMATAAA to the word following the beginning dummy block; WSMFREET contains the offset to the word following the ending dummy block.

The unallocated block is the free space that is unused but available for storing of objects. There is always just one unallocated block in the workspace. It has a count word at each end with a block type flag of 2; the interior is undefined. Whenever space for an active block is allocated, the space is taken alternately from the bottom and top end of the unallocated block. Thus, the used part of free space grows from each end toward the middle.

There are three control words in the translator and interpreter transient areas that point to the unallocated block. WSMFREEU contains the absolute address of the beginning count word. The rightmost bit of WSMFREEU is an allocation flag. It is flipped each time a block is allocated; if 0, space is allocated from the bottom (low address end) of the unallocated block; if 1, space is allocated from the top. WSMFREEA and WSMFREEZ contain, respectively, the offset from the beginning of the workspace to the word following the beginning count word and to the ending count word. These two control words are used primarily by certain translator and appendage routines that use the unallocated block as a work area. WSMFREEA and its contents also serve to delimit that part of the workspace that is written to disk.

There are three routines that may be called by exarch routines to get a block of free space. IEFIND takes a byte count as input; IEGTSPAC takes a data type descriptor, rank, and element count as input; IESPACST takes a model variable as input. All three routines allocate a block of the requested size and update the unallocated block count words and pointers; they also call IENAME to get a temporary internal name for the block. Translator and appendage routines communicate with IEFIND via the IESFIND service routine or the APLSFND macro. These allow the option of getting an internal name or providing one as input. The macro also provides the option of requesting that space be allocated from the top or bottom end of the unallocated block.

The IEFREE routine is called by exarch routines to free an active block and its internal name (if temporary). An active block is freed by setting the block type flag in each of its count words to 0 (inactive block). The preceding and following blocks are then checked. If either is unused (inactive or unallocated) the newly freed block is merged with it. Thus, if possible, the freed space is immediately recovered; if not, the presence of a few large inactive blocks rather than many small ones speeds up the collection of discarded material. The presence of the dummy blocks, which are flagged as active, obviates special handling when the first or last block is freed. Translator and appendage routines communicate with IEFREE via the IESFREE service routine or the APLSFREE macro.

The IATIDY routine is called to collect the discarded material when the amount of space requested for a block is not available, when the workspace is saved, and when system variable quad-WA is referenced. IATIDY goes through free space, block by block, deleting all inactive blocks and collecting all active blocks into the low address end of free space. The area to be examined is delimited by the dummy blocks and by WSMFREES and WSMFREET. During this process, IATIDY updates all the items that contain absolute free space addresses: the address table entry for each relocated active block, WSMFREEU, WSMNXINS, ZADDR, LADDR, and RADDR. It also updates the unallocated block count words and relative pointers (WSMFREEA and WSMFREEZ). Note that active blocks are simply moved; there is no need to examine or modify their contents since they contain no absolute addresses.

R13 STACK

The R13 stack extends from the end of free space to the end of the workspace. It is 1024 bytes in length; the relative location of its low address end is contained in WSMWORK (a control word in the executor transient area).

This area is used as a pushdown stack of variable-sized save areas. A save area (or level) is added to the stack whenever a routine is entered via macro APLENTRY. The level is removed when the routine returns to its caller via macro APLEXIT. It is known as the R13 stack because register 13 is used to point to the beginning of the current level; register 14 is used to point to the current level's end.

Each level is used as a save area for the calling routine's registers and, optionally, as a work area for the called routine. Execution of the APLENTRY macro always causes register 12 through register 15 of the calling routine to be saved in the new current level. Optional parameters cause other registers to be saved and a work area of the requested size to be appended to the current level.

All translator and appendage routines and all exarch service routines (those which are called by translator or appendage routines) begin with an APLENTRY macro and terminate with an APLEXIT macro. Each call to one of these routines, therefore, adds a level to the R13 stack. Potentially the R13 stack can overflow. However, the interpreter is designed so that calls are never nested to such a depth.

The remaining exarch routines execute as one routine in the sense that they share a single level of the R13 stack. This level is created when exarch is called by the translator at entry point IEXARCH; it is removed when exarch returns to the translator at IEXIT. Most exarch routines do not use the R13 stack as a work area; those that do use macros APLGET13 and APLDRP13 to extend and restore exarch's R13 stack level.

VSPC WORKSPACE

When VS APL is running under VSPC, it has a VSPC workspace as its data area. The first 2048 bytes of the VSPC workspace contain the control blocks and work areas described below. The remainder of the VSPC workspace contains the VS APL workspace described in the preceding section. If data has been placed in the alternate input stack, the stack follows the VS APL workspace, and if the user has a shared-variable connection with the FSM auxiliary processor or the GDDM auxiliary processor, an auxiliary-processor work area follows the alternate input stack.

At the beginning of the VSPC workspace is a 256-byte VSPC control block called the workspace header (WSH). Following the WSH is a 24-byte VSPC control block called the standard file name (SFN). The VS APL executor uses these control blocks in its communications with VSPC. They are described in <u>VS Personal Computing (VSPC): Writing Processors</u>, SH20-9074, and in <u>VSPC Version 2: Writing Processors</u>, SH20-9203.

Following the SFN is a 72-byte APL control block called the PERTERM header (PTH) that can be referenced, but not modified, by the translator and interpreter. The address of the PTH is contained in control work WSMPTHPT in the executor transient area of the VS APL workspace. Following the PTH is a 1556-byte executor work area that is for the exclusive use of the executor. The last 140 bytes preceding the VS APL workspace are used to contain a 132-byte buffer which must immediately precede WSMBUFF. This buffer is used to hold ⍞ output which may later be written to the terminal input area (display terminals only). See "Control Block Formats" for a detailed description of the PTH and the executor work area (ECA).

EXECUTOR DATA AREAS

CMS EXECUTOR GLOBAL TABLE

When VS APL is running under CMS, the executor has a 4096-byte work area called the global table. The address of the global table is contained in absolute storage location X'440' (symbolic location GLBLTABL in CMS macro NUCON).

At the beginning of the global table is a 72-byte APL control block called the PERTERM header (PTH) which can be referenced, but not modified, by the translator and the interpreter. The address of the PTH is contained in control word WSMPTHPT in the executor transient area of the VS APL workspace. The remainder of the global table is for the exclusive use of the executor. See "Control Block Formats" for a detailed description of the PTH and the global table (CMSGL).

PTH is immediately followed by a PTX which is available for use by only the CMS executor.

TSO EXECUTOR GLOBAL TABLE

When VS APL is running under TSO, the executor uses a work area called the global table. The address of the global table is contained in the first four bytes of the PRB save area, which is addressed by the TCBFSA field for all of the VS APL tasks.

At the beginning of the global table is a 72-byte APL control block called the PERTERM header (PTH) which can be referenced, but not modified, by the translator and interpreter. The address of the PTH is contained in control word WSMPTHPT in the executor transient area of the VS APL workspace. The remainder of the global table is for the exclusive use of the executor. See "Control Block Formats" for a detailed description of the PTH and the global table TSOGL.

VS APL EXECUTOR STACK FOR CICS

Most CICS executor routines use a special set of entry and exit codes that saves registers and provides working storage from a processing stack. The following list shows which modules create stacks, which routines use stacks, and where the stacks are located:

Module Creating Stack	Stack Use	Location of Stack
APLKASON	Used by user signon and library services routines	In the transaction work area (TWA) provided by CICS for the signon (APL) transaction
APLKADSP	Used by the process dispatcher routines	In the TWA for the user (APLU) transaction

Module Creating Stack	Stack Use	Location of Stack
APLKIFIX	Used by all YY code service routines—it is the primary executor stack for the user session	In transaction storage for the user (APLU) transaction
APLKLIBG	Used by the library service routines	In the TWA for the library (APLL) transaction
APLKTCTL	Used by terminal I/O routines	In the TWA for the terminal (APLT) transaction or (on attention) the sign on (APL) transaction
APL121K	Used by library services routines	In transaction storage for the user (APLU) transaction
APL124K	Used by terminal manager routines	In transaction storage for the user (APLU) transaction
APL132K	Used by destination manager routines	In transaction storage for the user (APLU) transaction
APLKSSUB	Used by shared storage manager routines	In shared memory

Stacks are made up of a series of variable-length entries, with register 11 always pointing to the current entry. Stacks are used beginning at the high address end and filling toward the low address end. Space within the stack is allocated in fullwords, and the stack may be of any size.

The format and content of a stack entry are shown in Figure 29.

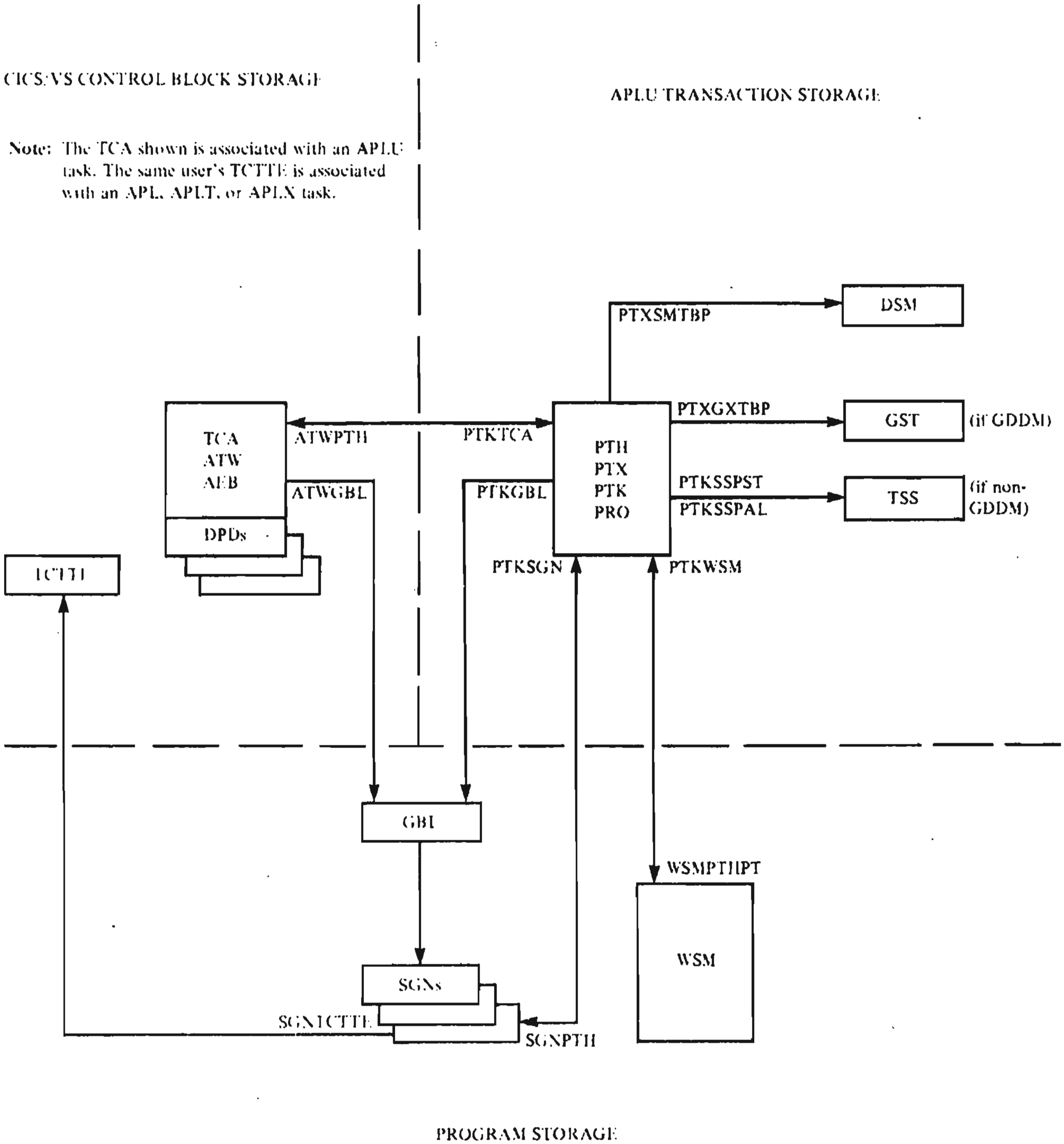
Dec	Hex	Length	Name	Description
0	0	4	STKP	Pointer to a word at the low address end of the stack area. The word contains the address of the stack overflow routine provided by the module that owns the stack. stack.
4	4	4	STKI	Four characters identifying the routine for which this stack entry is provided. When exit is made from this routine, the second bit in this field is turned off, providing easy identification of recent control flow when analyzing dumps.
8	8	4	STKR	The contents of register 14 when this routine was entered.
12	C	4Xn	STKS	The contents of any other registers saved when this routine was entered. Contains registers 2 through 10 for stack entries representing module entry points. For internal subroutines, STKS may be omitted or may contain a subset of registers 2 through 10.
Varies				Routines, when entered, may request additional stack space formatted in any way they require. For the format of this area and the total length of the stack entry, consult the individual routine.

Figure 29. Format and Content of a Stack Entry

CICS/VS EXECUTOR DATA AREA INTERRELATIONSHIPS

Figure 30 shows the relationships between major data areas used by the CICS/VS executor.

Note: The TCA shown is associated with an APLU task. The same user's TCTTE is associated with an APL, APLT, or APLX task.



Note: There is one GBL in the CICS/VS system. It points to the head of the SGN table. There is one active SGN entry for each user.

Figure 30. CICS/VS Executor Data Area Interrelationships

VS APL COMMON EXECUTOR STACK

Executor modules which are common to multiple environments, including most APLA... and APLX... modules, use a special set of entry and exit code that saves registers and provides working storage from a processing stack. Some of the modules which are unique to a single subsystem have also adopted this convention. All of these modules are referred to as "SP-modules", and can be identified by their use of an APLXPROC macro at the beginning of the executable code.

The APLXPROC macro generates an ID string at the module entry point that not only names the module and provides a compile data, but also contains a field indicating the amount of stack storage required by the module.

SP-modules are never called directly. Instead, a stack linkage routine is used which saves registers and provides the required working storage. The APLCALLS macro is normally used for this purpose. A single register (normally R13) is used to point to a stack entry which includes:

- a pointer to the linkage routine,
- a standard 18-word save area, and
- the required working storage.

The stack entry also has a prefix which identifies the module currently using the stack entry. In addition, stack services provides for abend exits at each level in the stack. These are also recorded in the stack entry prefix.

Typically, then, a stack entry will appear as follows:

```
             -----------------------------------
   -10      |  -> abend exit routine            |
             -----------------------------------
   - C      |        caller's R13               |
             -----------------------------------
   - 8      |        module name                |
            |                                   |
             -----------------------------------
R13--->     |  branch to linkage routine        |
             -----------------------------------
     4      |  -> previous stack entry          |
             -----------------------------------
     8      |  -> next stack entry              |
             -----------------------------------
     C      |  save R14-R12 when this           |
            /  module issues a call.            /
            /  (First byte is X'FF' if          /
            |  control has returned.)           |
             -----------------------------------
    48      |  working area for this module.    |
            /  (variable size)                  /
             -----------------------------------
```

CONTROL BLOCK FORMATS

Control blocks are given, in this subsection, in alphabetic order. Each control block heading shows the name of the control block followed (within parenthesis) by the components which use the control block. Acronyms are employed in place of the actual component names and have the following meanings:

ALL	All the principal components
AP	Auxiliary Processors
CICS	CICS/VS Executor
CMS	CMS Executor
CONV	Conversion Programs
NTRP	Interpreter including the translator exarch and appendage routines
SERV	CICS/VS Service Programs
TSO	TSO Executor
VSPC	VSPC Executor
XSYS	Cross-system components including the session manager, common auxiliary processor services, and common service support routines)

APC (XSYS, AP)

This is the common AP services interface request block. It contains the type of the shared storage manager request and the return code from the request. (The format of this layout is the one used in publications titled "Data Areas and Symbolic Names Cross-Reference Table," usually distributed on microfiche.) This control block is mapped by the APLXAPC macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	92	APC	
0	(0)	SIGNED	2	APCRQTYP	REQUEST CODE SET BY AP
2	(2)	SIGNED	2	APCRQVAR	WHICH SH VAR IN SET THIS
4	(4)	CHARACTER	6	APCRC	
4	(4)	BITSTRING	1	APCRET	AP DESIGNATED RETURN CODE
5	(5)	BITSTRING	1		UNUSED
6	(6)	SIGNED	2	APCSSMRC	RETURN CODE FROM SSM
8	(8)	SIGNED	2	APCSSMRS	REASON CODE FROM SSM
10	(A)	CHARACTER	2	APCFLAGS	MISCELLANEOUS FLAGS
10	(A)	BITSTRING	1	APCFLAG1	FLAG BYTE 1
		1...		APCEBCD	EBCDIC TRANSLATION FLAG
		.1..		APCIGNOR	IGNORE UNREF'D WAITING VALUE FLAG
		..11 1111			RESERVED
11	(B)	BITSTRING	1		RESERVED
12	(C)	A-ADDRESS	4	APCEAT	ERROR TABLE ADDRESS SET BY AP
16	(10)	SIGNED	4	APCPARM1	GENERAL INPUT/OUTPUT VALUE 1
20	(14)	SIGNED	4	APCPARM2	GENERAL INPUT/OUTPUT VALUE 2
24	(18)	SIGNED	4	APCPARM3	GENERAL INPUT/OUTPUT VALUE 3
28	(1C)	SIGNED	4	APCPARM4	GENERAL INPUT/OUTPUT VALUE 4
28	(1C)	SIGNED	2	APCNAMEL	LENGTH OF RESOURCE NAME
30	(1E)	BITSTRING	1	APCATYPE	TYPE OF RESOURCE FOR AUTHCHECK
31	(1F)	UNSIGNED	1	APCALEVL	ACCESS LEVEL FOR AUTHCHECK
32	(20)	A-ADDRESS	4	APCANCH	ANCHOR BLOCK ADDRESS
36	(24)	A-ADDRESS	4	APCPSCV	PRIMARY SHARED VARIABLE SCV
40	(28)	CHARACTER	52	APCWORKA	WORK/REG SAVE AREA FOR AP COMMON
40	(28)	SIGNED	52	APCREGS	REGISTER SAVE AREA
92	(5C)	CHARACTER	0	APCEND	END OF PLS APC MAPPING

CROSS REFERENCE

APC	0	(0)
APCALEVL	31	(1F)
APCANCH	32	(20)
APCATYPE	30	(1E)
APCEAT	12	(C)
APCEBCD	10	X'80'
APCEND	92	(5C)
APCFLAGS	10	(A)
APCFLAG1	10	(A)
APCIGNOR	10	X'40'
APCNAMEL	28	(1C)
APCPARM1	16	(10)
APCPARM2	20	(14)
APCPARM3	24	(18)
APCPARM4	28	(1C)
APCPSCV	36	(24)
APCRC	4	(4)
APCREGS	40	(28)
APCRET	4	(4)
APCRQTYP	0	(0)
APCRQVAR	2	(2)
APCSSMRC	6	(6)
APCSSMRS	8	(8)
APCWORKA	40	(28)

APFT (VSPC)

This is the information table for auxiliary processors distributed with VS APL for VSPC. There are 15 of these blocks defined within the VSPC executor work area (ECA). This control block is mapped by the APLPFT macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	0	APFT	
0	(0)	SIGNED	4	APCTL	CTL SHARE ID
4	(4)	SIGNED	4	APDAT	DATA SHARE ID
8	(8)	SIGNED	2	APCIN	CTL INTERNAL NAME
10	(A)	SIGNED	2	APDIN	DATA INTERNAL NAME
12	(C)	SIGNED	4	APAPRCOD	AP RETURN,REASON CODES
12	(C)	SIGNED	2	APAPRET	AP RETURN CODE
14	(E)	SIGNED	2	APAPREA	AP REASON CODE
16	(10)	HEX	1	APLACT	SAVE LAST AP ACTION.
APLACT DEFINITIONS=APFIFO DEFINITIONS					
17	(11)	HEX	1	APSACT	SAVE CURRENT USER/AP ACTION
APSACT FLAGS DEFINED BY APACVS DEFINITIONS					
18	(12)	SIGNED	2	APSVRCOD	SHRVAR RETURN,REASON CODES
18	(12)	HEX	1	APSVRET	SV RETURN CODE
19	(13)	HEX	1	APSVREA	SV REASON CODE
20	(14)	A-ADDRESS	4	APGDDXCO	OFFSET OF GDDX COMMON AREA AP126
20	(14)	SIGNED	4	APSFN	SFT/SET ADDRESS, ID
20	(14)	HEX	1	APSFNID	SFT/SET ID
21	(15)	A-ADDRESS	3	APSFNAD	SFT/SET ADDRESS (NON-RELOCATABLE)
GENERAL FLAGS					
24	(18)	HEX	1	APDSMS	DUMMY SHARE MEMORY(DSM) STATUS
APDSMS FLAG DEFINITIONS					
		11..		APDDSM	"X'C0'". DATA STATUS: 01=USER SPECIFIED 11=AP SPECIFIED 00=DATA REFERENCED
		.1..		APDUSPEC	"X'40'". USER SPECIFIED
		..11		APCDSM	"X'30'". CTL STATUS: 01=USER SPECIFIED 11=AP SPECIFIED 00=CTL REFERENCED
		...1		APCUSPEC	"X'10'". USER SPECIFIED
		1		APSVCMP	"X'01'". USER SV REQUEST COMPLETED
25	(19)	HEX	1	APACVS	ACCESS CONTROL FLAGS
APACVS FLAG DEFINITIONS					
		1...		APADSP	"X'80'". AP SPEC DATA
		.1..		APACSP	"X'40'". AP SPEC CTL
		..1.		APADRF	"X'20'". AP REF DATA
		...1		APACRF	"X'10'". AP REF CTL
		.... 1...		APUDSP	"X'08'". USER SPEC DATA
		.... .1..		APUCSP	"X'04'". USER SPEC CTL
		.... ..1.		APUDRF	"X'02'". USER REF DATA
		1		APUCRF	"X'01'". USER REF CTL
26	(1A)	HEX	1	APAPID	PSEUDO-AP IDENTIFICATION

APFT (VSPC) continued

```
 OFFSETS    TYPE     LENGTH  NAME              DESCRIPTION

==============================================================================
APAPID DEFINITIONS
            .... ...1        APVSAM            "X'01'". VSAM AP
            .... ..1.        APAPL             "X'02'". VSPC APL AP
            .... .1..        APBCD             "X'04'". VSPC EBCDIC AP
            .... 1...        APWSA             "X'08'". WORKSPACE ACCESS AP
            ...1 ....        APFSM             "X'10'". FSM AP PCM2045P
            ..1. ....        APGDDM            "X'20'". GDDM AP AP 126
            .1.. ....        APALT             "X'40'". ALTERNATE INPUT AP
            1... ....        APCMD             "X'80'". VSPC COMMAND AP
  27    (1B) HEX           1 APFLG             APFT ENTRY FLAGS
==============================================================================
APFLG DEFINITIONS
            1... ....        APCACT            "X'80'". CTL PARTNER ACTIVE
            .1.. ....        APDACT            "X'40'". DATA PARTNER ACTIVE
            11.. ....        APFTACT           "APCACT+APDACT".APFT ENTRY ACTIVE
                                               MASK
==============================================================================
AP FIFO ACTION STACK (APFIFO)
------------------------------------------------------------------------------
  28    (1C) SIGNED        4 APFIFO
==============================================================================
APFIFO ENTRY DEFINITIONS
 BYTE ENTRY PER AP ACTION
 ACTION ENTRY FLAGS:APACSP,APADSP,APADRF,APACRF
            .... 1...        APAPDT            "X'08'". DATA TYPE FLAG:
                                               0=CODE(RETURN,COMMAND) 1=DATA
------------------------------------------------------------------------------
  32    (20) CHARACTER     8 APSUFFIX          CTL/DAT SUFFIX
------------------------------------------------------------------------------
  40    (28) HEX           1 APSUFLTH          CTL/DAT SUFFIX LENGTH
  41    (29) HEX           1 APINFO
==============================================================================
VSPC COMMAND AP AND ALTERNATE INPUT AP INFORMATION
  41    (29) HEX           1 APCFLG            COMMAND, ALTERNATE INPUT AP FLAGS
==============================================================================
APCFLG FLAG DEFINITIONS
            1... ....        APCIBAD           "X'80'". INITIAL VALUE NOT YET
                                               ACCEPTED
            .11. ....        APCALL            "X'60'". COMMAND AP 'ALL' OPTION
            ..1. ....        APCERR            "X'20'". COMMAND AP 'ERROR' OPTION
            ...1 ....        APCLIFO           "X'10'". ALTERNATE INPUT AP 'LIFO'
                                               OPTION
==============================================================================
WORKSPACE ACCESS AP INFORMATION
  41    (29) HEX           1 APWFLG            WORKSPACE ACCESS AP FLAGS
==============================================================================
APWFLG FLAG DEFINITIONS
            11.. ....        APWVWS            "X'C0'". DISPLAY VSPC WORKSPACE
            .1.. ....        APWPTC            "X'40'". DISPLAY VSPC PTC
  42    (2A) SIGNED        2
------------------------------------------------------------------------------
  44    (2C) SIGNED        4 APWOFSET          OFFSET OF AREA TO BE EXAMINED
------------------------------------------------------------------------------
  48    (30) SIGNED        4 APWLENG           LENGTH OF AREA TO BE EXAMINED
==============================================================================
VSPC FILE AP INFORMATION
  41    (29) HEX           1 APPCOF
  41    (29) HEX           1 APPFLG            VSPC AP FLAGS
```

APFT (VSPC) continued

```
 OFFSETS     TYPE     LENGTH  NAME               DESCRIPTION

================================================================================
APPFLG FLAG DEFINITIONS
             1... ....        APPSI              "X'80'". OPEN MODES:SEQUENTIAL
                                                 INPUT
             .1.. ....        APPSO              "X'40'". SEQ.OUTPUT
             ..1. ....        APPDI              "X'20'". DIRECT INPUT
             ...1 ....        APPDIO             "X'10'". DIRECT INPUT/OUTPUT VSPC
                                                 FILE OPEN MASK
             1111 ....        APPOPEN            "APPSI+APPSO+APPDI+APPDIO"
             .... 1...        APPMODE            "X'08'". 1-VARIABLE PROTOCOL MODE:
                                                 0=COMMAND MODE 1=DATA TRANSFER
                                                 MODE VSPC FILE TYPE:
             .... .1..        APPFILUN           "X'04'". 0=DEFINED, 1=UNDEFINED
             .... ..1.        APPFILDR           "X'02'". 0=SEQUENTIAL, 1=DIRECT
                                                 SEQUENTIAL I/O SUBMODE:
             .... ...1        APPLINE            "X'01'". 0=DATA ONLY,1=LINE# AND
                                                 DATA
  42    (2A) SIGNED        2  APPREC             I/O BUFFER DISPLMT TO CURRENT ITEM
--------------------------------------------------------------------------------
  44    (2C) SIGNED        4  APPRECNO           LOGICAL RECORD NUMBER
--------------------------------------------------------------------------------
  48    (30) SIGNED        2  APPLRECL           DIRECT FILE RECORD LTH SAVE
             .... 1..1        APPLTH             "*-APPCOF"
================================================================================
VSAM FILE AP INFORMATION
  50    (32) HEX           1  APVSAMF
  50    (32) HEX           1  APVFLG1            VSAM AP FILE OPEN MODE FLAGS
================================================================================
APVFLG1 DEFINITIONS
             .... ...1        APVIN              "X'01'". OPEN INPUT
             .... ..1.        APVOUT             "X'02'". OPEN OUTPUT
             .... .1..        APVUP              "X'04'". OPEN UPDATE
             .... .111        APVOPEN            "APVIN+APVOUT+APVUP"VSAM FILE OPEN
                                                 MASK
  51    (33) HEX           1  APVFLG2            VSAM AP ACTION FLAGS
================================================================================
APVFLG2 DEFINITIONS=CURRENT I/O ACTION
             .... ...1        APVR               "X'01'". READ
             .... ..1.        APVRU              "X'02'". READ UPDATE
             .... .1..        APVWRT             "X'04'". WRITE
             .... 1...        APVERA             "X'08'". ERASE
             ...1 ....        APVPOS             "X'10'". POSITION
             ...1 ...1        APVKF              "X'11'". KEY FEEDBACK
             ..1. ....        APVOPN             "X'20'". OPEN
             .1.. ....        APVCLS             "X'40'". CLOSE
             1... ....        APVKEY             "X'80'". KEYED REQUEST
             1... ....        APVWRTI            "X'80'". WRITE NEW RECORD (OPEN
                                                 UPDATE)
             .... ..1.        APVLTH             "*-APVSAMF"
================================================================================
FULL SCREEN MANAGER (FSM) AP INFORMATION
  41    (29) HEX           1  APFSMF
  41    (29) HEX           1  APFFLAG            WORK FLAG
================================================================================
APFFLAG DEFINITIONS
             .... ...1        APFIDXF            "X'01'". INDEX INTO FSMFLD FOR FLD
                                                 NO.
             .... ..1.        APFPENDF           "X'02'". PENDING FORMAT
             .... .1..        APFBUZZR           "X'04'". SET ALARM ON NEXT
                                                 READ,WRITE
             .... 1...        APFCURSR           "X'08'". SET CURSOR ON WRITE
             ..1. ....        APFFINIT           "X'20'". FSMWORK INITIALIZED
             ...1 ....        APFMEMP            "X'10'". DAT VARIABLE EMPTY
  42    (2A) HEX           1  APFCMDR            LAST READ-TYPE COMMAND
```

```
 OFFSETS     TYPE      LENGTH  NAME            DESCRIPTION

================================================================================
APFCMDR VALUES = APFREAD,APFGET,APFRFORM,APFWRITE
  43    (2B) HEX            1  APFSMCMD        CURRENT FSM CMD
================================================================================
APFSMCMD VALUES
             .... ...1         APFFORMT        "1". FORMAT
             .... ..1.         APFWRITE        "2". WRITE
             .... ..11         APFREAD         "3". READ
             .... .1.1         APFGET          "5". GET
             .... .11.         APFMTYPE        "6". MODIFY-TYPE
             .... .111         APFMMINT        "7". MODIFY-INTENSITY
             .... 1..1         APFRFORM        "9". READ-FORMAT
             .... 1.1.         APFHCOPY        "10". HARDCOPY
             .... 1.11         APFBUZZ         "11". SOUND ALARM
             .... 11..         APFSETC         "12". SET CURSOR
             .... 11..         APFCMDH         "12". HIGH COMMAND VALUE
--------------------------------------------------------------------------------
  44    (2C) SIGNED         4  APDATVAL        DISPLMT FROM WS TO DAT VALUE BLK

--------------------------------------------------------------------------------
  48    (30) SIGNED         4  APFSMD          DISPLMT FROM WSH TO FSMWORK
             .... 1.11         APFLTH          "*-APFSMF"
================================================================================
GDDM AP INFORMATION
  41    (29) HEX            1  APGDDMF
  41    (29) HEX            1  APGCTYPE        DATA TYPE OF CTL VARIABLE
================================================================================
APGCTYPE VALUES
             .... ...1         APGFIXED        "1". INTEGER
             .... ..1.         APGFLOAT        "2". FLOATING POINT
  42    (2A) SIGNED         2  APGINDEX        INDEX OF GDDX PATH UNIQUE BLOCK
--------------------------------------------------------------------------------
  44    (2C) A-ADDRESS      4  APGCTLBO        OFFSET OF OUTPUT CTL BUFFER
--------------------------------------------------------------------------------
  48    (30) A-ADDRESS      4  APGDATBO        OFFSET OF OUTPUT DAT BUFFER AP126
             .... 1.11         APGLTH          "*-APGDDMF" AP126
             ..11 .1..         APFTLTH         "*-APFT"
```

CROSS REFERENCE

APACRF	25	X'10'	APFRFORM	43	X'09'	APUDSP	25	X'08'
APACSP	25	X'40'	APFSETC	43	X'0C'	APVCLS	51	X'40'
APACVS	25	(19)	APFSM	26	X'10'	APVERA	51	X'08'
APADRF	25	X'20'	APFSMCMD	43	(2B)	APVFLG1	50	(32)
APADSP	25	X'80'	APFSMD	48	(30)	APVFLG2	51	(33)
APALT	26	X'40'	APFSMF	41	(29)	APVIN	50	X'01'
APAPDT	28	X'08'	APFT	0	(0)	APVKEY	51	X'80'
APAPID	26	(1A)	APFTACT	27	X'C0'	APVKF	51	X'11'
APAPL	26	X'02'	APFTLTH	48	X'34'	APVLTH	51	X'02'
APAPRCOD	12	(C)	APFWRITE	43	X'02'	APVOPEN	50	X'07'
APAPREA	14	(E)	APGCTLBO	44	(2C)	APVOPN	51	X'20'
APAPRET	12	(C)	APGCTYPE	41	(29)	APVOUT	50	X'02'
APBCD	26	X'04'	APGDATBO	48	(30)	APVPOS	51	X'10'
APCACT	27	X'80'	APGDDM	26	X'20'	APVR	51	X'01'
APCALL	41	X'60'	APGDDMF	41	(29)	APVRU	51	X'02'
APCDSM	24	X'30'	APGDDXCO	20	(14)	APVSAM	26	X'01'
APCERR	41	X'20'	APGFIXED	41	X'01'	APVSAMF	50	(32)
APCFLG	41	(29)	APGFLOAT	41	X'02'	APVUP	50	X'04'
APCIBAD	41	X'80'	APGINDEX	42	(2A)	APVWRT	51	X'04'
APCIN	8	(8)	APGLTH	48	X'0B'	APVWRTI	51	X'80'
APCLIFO	41	X'10'	APINFO	41	(29)	APWFLG	41	(29)
APCMD	26	X'80'	APLACT	16	(10)	APWLENG	48	(30)
APCTL	0	(0)	APPCOF	41	(29)	APWOFSET	44	(2C)
APCUSPEC	24	X'10'	APPDI	41	X'20'	APWPTC	41	X'40'
APDACT	27	X'40'	APPDIO	41	X'10'	APWSA	26	X'08'
APDAT	4	(4)	APPFILDR	41	X'02'	APWVWS	41	X'C0'
APDATVAL	44	(2C)	APPFILUN	41	X'04'			
APDDSM	24	X'C0'	APPFLG	41	(29)			
APDIN	10	(A)	APPLINE	41	X'01'			
APDSMS	24	(18)	APPLRECL	48	(30)			
APDUSPEC	24	X'40'	APPLTH	48	X'09'			
APFBUZZ	43	X'0B'	APPMODE	41	X'08'			
APFBUZZR	41	X'04'	APPOPEN	41	X'F0'			
APFCMDH	43	X'0C'	APPREC	42	(2A)			
APFCMDR	42	(2A)	APPRECNO	44	(2C)			
APFCURSR	41	X'08'	APPSI	41	X'80'			
APFFINIT	41	X'20'	APPSO	41	X'40'			
APFFLAG	41	(29)	APSACT	17	(11)			
APFFORMT	43	X'01'	APSFN	20	(14)			
APFGET	43	X'05'	APSFNAD	21	(15)			
APFHCOPY	43	X'0A'	APSFNID	20	(14)			
APFIDXF	41	X'01'	APSUFFIX	32	(20)			
APFIFO	28	(1C)	APSUFLTH	40	(28)			
APFLG	27	(1B)	APSVCMP	24	X'01'			
APFLTH	48	X'0B'	APSVRCOD	18	(12)			
APFMEMP	41	X'10'	APSVREA	19	(13)			
APFMMINT	43	X'07'	APSVRET	18	(12)			
APFMTYPE	43	X'06'	APUCRF	25	X'01'			
APFPENDF	41	X'02'	APUCSP	25	X'04'			
APFREAD	43	X'03'	APUDRF	25	X'02'			

APM (CICS, XSYS)

This is the request control block for authorization check in CICS/VS. It is mapped by the APLKAPM macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	13	APM	ACHK PARAMETER LIST
0	(0)	BITSTRING	3	APMAMASK	USER AUTHORIZATION MASK
3	(3)	BITSTRING	1	APMRELBY	WHICH TABLE BYTE TO CHECK
4	(4)	BITSTRING	1	APMRESOR	THE RESOURCE TYPE TO CHECK
5	(5)	CHARACTER	8	APMNAME	THE RESOURCE NAME TO CHECK

CROSS REFERENCE

APM	0	(0)
APMAMASK	0	(0)
APMNAME	5	(5)
APMRELBY	3	(3)
APMRESOR	4	(4)

ATW (CICS, AP)

This is the CICS/VS executor user task TWA mapping area which contains register save areas and dispatch blocks for VS APL processes. It is mapped by the APLKATW macro.

ABOVE FIELDS MUST REMAIN AT THE BEGINNING OF ATW, AS THEY ARE USED BY APLKWAIT, APLKEXIT, AND THE SSM REQUEST MACROS.

PROGRAM CHECK SAVE AREA
APLXBND DSECT=NO,P=ATB GENERATE XBEND AREA

THE FOLLOWING FIELDS ARE SET ONLY FOR PROGRAM CHECK TYPE ABENDS. THE AREAS ARE DEFINED FOR SYSTEM ABENDS BUT ARE NOT SET OR USED BY ABEND RECOVERY SERVICES.

THE FOLLOWING 16 WORDS ARE THE REGISTERS AT THE POINT OF THE PROGRAM CHECK, STORED FROM REGISTER 0 TO 15

DEPENDENT PROCESS CONTROL

STACK FOR APLKADSP

BEGINNING OF DPDS

DPD - DEPENDENT PROCESS DISPATCH ENTRY
THE DPD'S ARE LOCATED IN THE TWA FOLLOWING THE ATW HEADER.
ONE ENTRY IS USED PER PROCESS. THIS INCLUDES THE INTERPRETER INTERFACE PROCESS AND THE VARIOUS DEPENDENT AP PROCESSES.
EACH ENTRY CONTAINS ALL INFORMATION REQUIRED TO RESUME EXECUTION OF THE PROCESS, AS WELL AS ABEND EXIT AND AP NUMBER DATA.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	0	DPD	
0	(0)	HEX	1	DPDFLAG	USAGE FLAGS
		1...		DPDFWAIT	"X'80'" THIS PROCESS IS WAITING
		.1..		DPDFSING	"X'40'" WAITING ON A SINGLE ECB
		..1.		DPDFAPL	"X'20'" WAITING ON APL ECB(S) ONLY
		...1		DPDFSYS	"X'10'" WAITING ON SYS (NON CICS) ECB(S)
		1111		DPDFWALL	"DPDFWAIT+DPDFSING+DPDFAPL+DPDFSYS" ALL WAIT FLGS
		.... 1...		DPDFXBX	"X'08'" ABND EXIT NEEDS FULL BND
		.... ..1.		DPDFSTRT	"X'02'" PROCESS BEING STARTED
		1		DPDFACT	"X'01'" THIS ENTRY IS ACTIVE
1	(1)	HEX	1	DPDNR	RELATIVE DPD NUMBER
2	(2)	SIGNED	2	DPDOFFST	OFFSET IN DAP TABLE OF AP NR
4	(4)	CHARACTER	8	DPDXIT	ABEND EXIT INFORMATION
4	(4)	A-ADDRESS	4	DPDXADR	EP TO EXIT ROUTINE
8	(8)	SIGNED	4	DPDXPRM	PARM TO PASS TO EXIT ROUTINE
12	(C)	A-ADDRESS	4	DPDWKA	ADDR OF WORK AREA PASSED TO AP
16	(10)	CHARACTER	16	DPDECBL	PTRS TO UP TO 4 ECBS, T USED FOR DEBUGGING
32	(20)	CHARACTER	64	DPDSAVE	SAVE ALL PROCESS REGS

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
32	(20)	SIGNED	4	DPDPR0	REG 0
36	(24)	SIGNED	4	DPDPREG(13)	REGS 1 THRU 13
88	(58)	SIGNED	4	DPDPR14	REG 14 T
92	(5C)	SIGNED	4	DPDPR15	REG 15 T

NOTE THAT REG 1 IS A POINTER TO THE ECBLIST.
IF IT IS ZERO, THE PROCESS IS AUTOMATICALLY DISPATCHABLE.

.11.	DPDLEN	"*-DPD" LENGTH OF ONE DPD ENTRY

CROSS REFERENCE

DPD	0	(0)
DPDECBL	16	(10)
DPDFACT	0	X'01'
DPDFAPL	0	X'20'
DPDFLAG	0	(0)
DPDFSING	0	X'40'
DPDFSTRT	0	X'02'
DPDFSYS	0	X'10'
DPDFWAIT	0	X'80'
DPDFWALL	0	X'F0'
DPDFXBX	0	X'08'
DPDLEN	92	X'60'
DPDNR	1	(1)
DPDOFFST	2	(2)
DPDPREG	36	(24)
DPDPR0	32	(20)
DPDPR14	88	(58)
DPDPR15	92	(5C)
DPDSAVE	32	(20)
DPDWKA	12	(C)
DPDXADR	4	(4)
DPDXIT	4	(4)
DPDXPRM	8	(8)

BND (XSYS, AP)

This is the common executor services abend interface block for abend exits. It is mapped by the APLXBND macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	80	BND	
0	(0)	SIGNED	4		ABEND TYPE WORD
0	(0)	BITSTRING	1	BNDTYPE	ABEND TYPE BYTE
		1...		BNDPROG	PROGRAM CHECK INDICATOR
		.1..		BNDSYS	SYSTEM ABEND INDICATOR
1	(1)	BITSTRING	3		RESERVED
4	(4)	SIGNED	4	BNDCODE	SYSTEM PROVIDED ABEND CODE,
8	(8)	BITSTRING	8	BNDPSW	PROGRAM CHECK PSW
8	(8)	SIGNED	4	BNDPSWD1	WORD ONE OF PSW
8	(8)	BITSTRING	2	BNDBCMKC	BC MODE MASKS, KEY AND CMWP
10	(A)	SIGNED	2	BNDBCINT	BC MODE INTERRUPT CODE
12	(C)	SIGNED	4	BNDPSWD2	WORD TWO OF PSW
12	(C)	A-ADDRESS	4	BNDECADR	EC MODE INTERRPUT ADDRESS
12	(C)	BITSTRING	1	BNDBCICP	BC MODE FIRST BYTE OF WORD 2
		11..		BNDBCILC	BC MODE INSTURCTION LENGTH
		..11		BNDBCCC	BC MODE CONDITION CODE
		 1111		BNDBCMSK	BC MODE PROGRAM MASK
13	(D)	A-ADDRESS	3	BNDBCADR	BC MODE INTERRUPT ADDRESS
16	(10)	CHARACTER	64	BNDREGS	BEGINNING OF REGISTER SAVE AREA
16	(10)	SIGNED	4	BNDREG0	REGISTER 0 AT PROGRAM CHECK
20	(14)	SIGNED	4	BNDREG1	REGISTER 1 AT PROGRAM CHECK
24	(18)	SIGNED	4	BNDREG2	REGISTER 2 AT PROGRAM CHECK
28	(1C)	SIGNED	4	BNDREG3	REGISTER 3 AT PROGRAM CHECK
32	(20)	SIGNED	4	BNDREG4	REGISTER 4 AT PROGRAM CHECK
36	(24)	SIGNED	4	BNDREG5	REGISTER 5 AT PROGRAM CHECK
40	(28)	SIGNED	4	BNDREG6	REGISTER 6 AT PROGRAM CHECK
44	(2C)	SIGNED	4	BNDREG7	REGISTER 7 AT PROGRAM CHECK
48	(30)	SIGNED	4	BNDREG8	REGISTER 8 AT PROGRAM CHECK
52	(34)	SIGNED	4	BNDREG9	REGISTER 9 AT PROGRAM CHECK
56	(38)	SIGNED	4	BNDREGA	REGISTER 10 AT PROGRAM CHECK
60	(3C)	SIGNED	4	BNDREGB	REGISTER 11 AT PROGRAM CHECK
64	(40)	SIGNED	4	BNDREGC	REGISTER 12 AT PROGRAM CHECK
68	(44)	SIGNED	4	BNDREGD	REGISTER 13 AT PROGRAM CHECK

BND (XSYS, AP) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
72	(48)	SIGNED	4	BNDREGE	REGISTER 14 AT PROGRAM CHECK
76	(4C)	SIGNED	4	BNDREGF	REGISTER 15 AT PROGRAM CHECK

CROSS REFERENCE

BND	0	(0)
BNDBCADR	13	(D)
BNDBCCC	12	X'30'
BNDBCICP	12	(C)
BNDBCILC	12	X'C0'
BNDBCINT	10	(A)
BNDBCMKC	8	(8)
BNDBCMSK	12	X'0F'
BNDCODE	4	(4)
BNDECADR	12	(C)
BNDPROG	0	X'80'
BNDPSW	8	(8)
BNDPSWD1	8	(8)
BNDPSWD2	12	(C)
BNDREGA	56	(38)
BNDREGB	60	(3C)
BNDREGC	64	(40)
BNDREGD	68	(44)
BNDREGE	72	(48)
BNDREGF	76	(4C)
BNDREGS	16	(10)
BNDREG0	16	(10)
BNDREG1	20	(14)
BNDREG2	24	(18)
BNDREG3	28	(1C)
BNDREG4	32	(20)
BNDREG5	36	(24)
BNDREG6	40	(28)
BNDREG7	44	(2C)
BNDREG8	48	(30)
BNDREG9	52	(34)
BNDSYS	0	X'40'
BNDTYPE	0	(0)

CIT (CICS, SERV)

This is the VSAM control interval trailer written by the CICS/VS executor library services and the CICS/VS service program. It is mapped by the APLKCIT macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	10	CITS	
0	(0)	SIGNED	4	CITSBLOK	SCROLL BLOCK SEQUENCE, NR OF
4	(4)	SIGNED	2	CITSLINE	NR OF LOGICAL LINES IN CI
6	(6)	SIGNED	2	CITSSEGN	NR OF LINE SEGMENTS IN CI
8	(8)	SIGNED	2	CITSSEGL	LENGTH USED FOR SEGN CALC
0	(0)	STRUCTURE	4	CITW	SCROLL LINE DESCRIPTOR
0	(0)	SIGNED	2	CITWSEGN	NR SCREEN SEGMENTS IN LINE
2	(2)	SIGNED	2	CITWOFFS	OFFSET IN CI TO LINE DATA
0	(0)	STRUCTURE	32	CIT	
0	(0)	CHARACTER	14	CITKEY	FILE LOCATOR KEY
0	(0)	UNSIGNED	3	CITLIBNO	LIBRARY NUMBER
3	(3)	CHARACTER	11	CITMBRA	
3	(3)	CHARACTER	8	CITMEMBR	MEMBER NAME
3	(3)	CHARACTER	8	CITCLNAM	CLIST NAME
11	(B)	UNSIGNED	3	CITHILN	HIGHEST LINE NUMBER
14	(E)	BITSTRING	1	CITFTYPE	FILE TYPE
		111.			RESERVED
		...1		CITCLNRD	CLIST IS NOREAD
		.... 1...		CITOBJ	OBJECT PROGRAM
		.... .1..			RESERVED
		.... ..1.		CITRO	RECORD ORIENTED
		1		CITDIR	DIRECT
15	(F)	CHARACTER	1	CITCATTR	CONTENT ATTRIBUTE
16	(10)	UNSIGNED	4	CITNRBA	NEXT CONTROL INTERVAL RBA
16	(10)	A-ADDRESS	4	CITNEXT	NEXT 4K BLOCK ADDRESS
20	(14)	SIGNED	2	CITNLR	NUMBER OF LOGICAL RECORDS
22	(16)	SIGNED	2	CITDLEN	DATA LENGTH
22	(16)	A-ADDRESS	2	CITCLDSP	DISP IN BLOCK TO NEXT LINE
24	(18)	CHARACTER	1		RESERVED
25	(19)	CHARACTER	7	CITVSAM	VSAM CONTROL INFORMATION
25	(19)	UNSIGNED	3	CITVRDF	VSAM RDF
25	(19)	BITSTRING	1	CITRDFF	VSAM RDF FLAG BYTE
26	(1A)	SIGNED	2	CITRECL	VSAM RECORD LENGTH
28	(1C)	UNSIGNED	4	CITVCIDF	VSAM CIDF
28	(1C)	A-ADDRESS	2	CITCIFSD	CONTROL INTERVAL FREE SPACE DISPLACEMENT
30	(1E)	SIGNED	2	CITCIFSL	CONTROL INTERVAL FREE SPACE LENGTH

CROSS REFERENCE

CIT	0	(0)
CITCATTR	15	(F)
CITCIFSD	28	(1C)
CITCIFSL	30	(1E)
CITCLDSP	22	(16)
CITCLNAM	3	(3)
CITCLNRD	14	X'10'
CITDIR	14	X'01'
CITDLEN	22	(16)
CITFTYPE	14	(E)
CITHILN	11	(B)
CITKEY	0	(0)
CITLIBNO	0	(0)
CITMBRA	3	(3)
CITMEMBR	3	(3)
CITNEXT	16	(10)
CITNLR	20	(14)
CITNRBA	16	(10)
CITOBJ	14	X'08'
CITRDFF	25	(19)
CITRECL	26	(1A)
CITRO	14	X'02'
CITS	0	(0)
CITSBLOK	0	(0)
CITSLINE	4	(4)
CITSSEGL	8	(8)
CITSSEGN	6	(6)
CITVCIDF	28	(1C)
CITVRDF	25	(19)
CITVSAM	25	(19)
CITW	0	(0)
CITWOFFS	2	(2)
CITWSEGN	0	(0)

CMSGL (CMS, XSYS, AP)

This is the CMS executor global table mapping. For a more detailed description, see "Executor Data Areas." It is mapped by the APLCMSGL macro.

```
 OFFSETS    TYPE     LENGTH  NAME            DESCRIPTION

-------------------------------------------------------------------------------
   0    (0) STRUCTURE     0  CMSGL
-------------------------------------------------------------------------------
   0    (0) FLOATING      8  PTH
===============================================================================
 THE PERTERM HEADER PROVIDES INFORMATION ABOUT THE ACTIVE
 USER WITH REGARD TO THE SYSTEM ENVIRONMENT, AND COMPLETES
   0    (0) SIGNED        4  PTHWORD1
-------------------------------------------------------------------------------
   0    (0) HEX           1  PTHASYNC
            1... ....        PTHDATTN        "X'80'". DOUBLE-ATTENTION
                                             SIGNALLED
            .1.. ....        PTHQEND         "X'40'". QUANTUM-END REQUESTED
            ..1. ....        PTHCPULM        "X'20'" CPU LIMIT EXCEEDED.
            .... .1..        PTHNOOUT        "X'04'" 'CANCEL OUTPUT' SIGNAL
                                             RECEIVED.
            .... ..1.        PTHFOFF         "X'02'". LINE-DROP OR BOUNCE
            .... ...1        PTHATTN         "X'01'". SINGLE ATTENTION
                                             SIGNALLED
   1    (1) HEX           1  (2)             RESERVED
   3    (3) HEX           1  PTHSUSP1        SUPERVISOR SUSPENSION BITS
            1... ....        PTHCWBIT        "X'80'". CLOCK WAIT BIT
            .1.. ....        PTHWABIT        "X'40'". YYWATE BIT
            ..1. ....        PTHSVBIT        "X'20'". SH. VAR. WAIT BIT
===============================================================================
PTHWSTAT HOLDS THE PROCESSING STATE OF THIS WS
-------------------------------------------------------------------------------
   4    (4) HEX           1  PTHWSTAT
            1... ....        PTHSVON         "X'80'". THIS USER SIGNED ON TO
                                             SVP
            .... ..1.        PTHSINK         "X'02'". THIS IS A COPY SINK
            .... ...1        PTHSORS         "X'01'". THIS IS A COPY SOURCE
===============================================================================
PTHUSTAT RECALLS THINGS WE'RE DOING FOR OR TO THIS USER
   5    (5) HEX           1  PTHUSTAT
            1... ....        PTHLOCKB        "X'80'". WE KEEP HIS KBD LOCKED
            .1.. ....        PTHMDY          "X'40'". DATE FORMAT FLAG
===============================================================================
PTHMDY=1='MM/DD/YY'
PTHMDY=0='DD-MM-YY'
            ..1. ....        PTHMSBLK        "X'20'". WE BLOCK HIS MESSAGES
            ...1 ....        PTHMICRO        "X'10'". APL MICROCODE WILL BE
                                             USED.
            .... 1...        PTHFSAVL        "X'08'" RESERVED FOR FULLSCREEN
                                             EDIT
            .... .1..        PTHUEXTN        "X'04'" PTH EXTENSION (PTX) EXISTS
===============================================================================
PTHQVAR IS THE MAXIMUM NUMBER OF VARIABLES HE MAY SHARE
   6    (6) SIGNED        2  PTHQVAR
===============================================================================
PTHYYCOD CONTAINS THE YYCODE OF THE LAST SVCC ISSUED
PTHSRCOD CONTAINS THE RETURN CODE THAT RESULTED.
   8    (8) SIGNED        4  PTHYYRC
-------------------------------------------------------------------------------
   8    (8) SIGNED        2  PTHYYCOD
            1... ....        PTHSPCLY        "X'80'"HI-ORDER BIT ON IF
                                             'SPECIAL' YYCODE
  10    (A) SIGNED        2  PTHSRCOD
```

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
PTHWIDTH IS THIS TERMINAL'S CURRENT LINE-WIDTH SETTING					
12	(C)	SIGNED	2		RESERVED
14	(E)	SIGNED	2	PTHWIDTH	
PTHCURSR IS THE TYPEBALL POSITION RESULTING FROM THE LAST TYO OR TYI. PTHCURSR=0='AT THE LEFT MARGIN'.					
16	(10)	SIGNED	2		RESERVED
18	(12)	SIGNED	2	PTHCURSR	
PTHQSIZE IS THE MAXIMUM SIZE A SHARED VARIABLE MAY OBTAIN					
20	(14)	SIGNED	4	PTHQSIZE	
PTHPARM1, PTHPARM2 ARE RETURN PARAMETER FIELDS FOR SOME SVCC FUNCTIONS.					
24	(18)	FLOATING	8		
24	(18)	SIGNED	4	PTHPARM1	
28	(1C)	SIGNED	4	PTHPARM2	
PTHWSLEN CONTAINS THE SIZE OF THE WS ADDRESS SPACE					
32	(20)	SIGNED	4	PTHWSLEN	
PTHACCNO CONTAINS THE BINARY ACCOUNT NUMBER OF THIS USER					
36	(24)	SIGNED	4	PTHACCNO	
TIME FIELDS: ALL ARE IN APL-STANDARD TIME FORMAT IE A FLOATING POINT NUMBER OF MICROSECONDS, POSSIBLY FRACTIONAL. TIME-OF-DAY VALUES ARE FROM THE BEGINNING OF THE APL EPOCH. INTERVALS ARE SIMPLY MICROSECOND COUNTS. PTHLOCAL IS THE OFFSET OF THIS USER FROM GMT. PTHCPUTM IS THE CPU TIME THIS SESSION. PTHKEYTM IS THE UNLOCKED-KBD TIME THIS SESSION. PTHCNCTM IS THE DATE/TIME HE SIGNED ON.					
40	(28)	FLOATING	8		
40	(28)	FLOATING	8	PTHLOCAL	
48	(30)	FLOATING	8	PTHCPUTM	
56	(38)	FLOATING	8	PTHKEYTM	
64	(40)	FLOATING	8	PTHCNCTM	
		.1.. 1...		PTHSIZE	"*-PTH" SIZE OF PERTERM HEADER.
APLXXPTX DSECT=NO					
72	(48)	FLOATING	8	PTX	PERTERM EXTENSION FOR EXECUTOR COMMON SERVICES
72	(48)	A-ADDRESS	4	PTXWSM	ADDR OF ACTIVE WORKSPACE
76	(4C)	A-ADDRESS	4	PTXVCT	ADDR OF VECTOR TABLE
80	(50)	A-ADDRESS	4	PTXSTACK	ADDR OF SP STACK
84	(54)	A-ADDRESS	4	PTXSMTBP	ADDR OF SESSION TABLE

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
88	(58)	A-ADDRESS	4	PTXGXTBP	ADDR OF GDDX CONTROL TABLE
92	(5C)	A-ADDRESS	4	PTXGXGDM	ADDR OF CURRENT GDM
96	(60)	A-ADDRESS	4	PTXPRTBP	ADDR OF PRINT SERVICES TABLE
100	(64)	A-ADDRESS	4	PTXFSTBP	ADDR OF FILE SERVICES TABLE
104	(68)	A-ADDRESS	4	PTXATTN	ADDR OF ACTIVE ATTENTION ROUTINE
108	(6C)	SIGNED	4	PTXFLAG	DEFINE WORD OF FLAGS
108	(6C)	HEX	1	PTXSUBSY	SUBSYSTEM FLAGS
		1...		PTXTSO	"X'80'" THIS IS A TSO USER
		.1..		PTXCMS	"X'40'" THIS IS A CMS USER
		..1.		PTXCICS	"X'20'" THIS IS A CICS USER
		...1		PTXVSPC	"X'10'" THIS IS A VSPC USER
109	(6D)	HEX	1	PTXDEBUG	VARIOUS DEBUG OPTIONS
		1...		DBGMICRO	"X'80'" DEBUG CANCEL MICROCODE TEST DEBUG
		.1..		DBGNSTAE	"X'40'" DEBUG CANCEL ESTAE EXITS DEBUG
		.... ..1.		DBGECHO	"X'02'" DEBUG ECHO STACK (CMD PARM) DEBUG
		1		DBGMSG	"X'01'" DEBUG ERROR MESSAGES DEBUG
110	(6E)	HEX	1	PTXFLAGS	GENERAL USE FLAGS
		1...		PTXAIPUR	"X'80'" PURGE THE ALTERNATE INPUT STACK
		.1..		PTXFSRST	"X'40'" FULLSCREEN RESTORE REQUIRED
		.... 1...		PTXADSM	"X'08'" ADSM OWNS THE SESSION
111	(6F)	HEX	1		RESERVED
112	(70)	A-ADDRESS	4	PTXDXTBP	DUMP SERVICES TABLE POINTER
116	(74)	SIGNED	4	PTXLEVEL	VS APL RELEASE LEVEL
120	(78)	SIGNED	4	PTXCODE	TERMINAL TYPE (GDDM) CODE
COMMON WORK AREA, USED FOR/BY ADSM AND IS IS AVAILABLE FOR OTHER USERS AS A SCRATCH					
124	(7C)	CHARACTER	28	PTXSCRTH	7 WORD SCRATCH PAD AREA
124	(7C)	CHARACTER	8	PTXSMPSD	ADSM PASSWORD RETURN AREA
124	(7C)	CHARACTER	8	PTXSMPRO	ADSM PROFILE OPTION (OR BLANKS)
124	(7C)	A-ADDRESS	4	PTXSMP1	ADSM PARM1 FIELD
128	(80)	A-ADDRESS	4	PTXSMP2	ADSM PARM2 FIELD
132	(84)	A-ADDRESS	4	PTXSMP3	ADSM PARM3 FIELD
136	(88)	A-ADDRESS	4	PTXSMP4	ADSM PARM4 FIELD
140	(8C)	A-ADDRESS	4	PTXSMP5	ADSM PARM5 FIELD
144	(90)	A-ADDRESS	4	PTXSMP6	ADSM PARM6 FIELD
148	(94)	A-ADDRESS	4	PTXSMP7	ADSM PARM7 FIELD
152	(98)	SIGNED	4	PTXHILIT	1D,II,OO,FO SF,OUT-ATTR,IN-ATTR,FLAGS HILITE

CMSGL (CMS, XSYS, AP) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
152	(98)	HEX	1	PTXHISF	START FIELD 3270 ORDER HILITE
153	(99)	HEX	1	PTXHIAOT	OUTPUT ATTRIBUTE BYTE HILITE
154	(9A)	HEX	1	PTXHIIOT	INPUT ATTRIBUTE BYTE HILITE
155	(9B)	HEX	1	PTXHIFLG	FLAGS (OUTPUT,INPUT HILITE) HILITE
		1...		PTXHIOHI	"X'80'" OUTPUT HILITING REQUESTED HILITE
		.1..		PTXHIIHI	"X'40'" INPUT HILITING REQUESTED HILITE
156	(9C)	A-ADDRESS	4	PTXHELPQ	ADDRESS OF MESSAGE QUEING RTN
160	(A0)	A-ADDRESS	4	PTXUSRWA	ADDR OF INST EXIT WORK AREA
164	(A4)	SIGNED	4	PTXRSV01	RESERVED
168	(A8)	SIGNED	4	PTXRSV02	RESERVED
172	(AC)	SIGNED	4	PTXRSV03	RESERVED
176	(B0)	SIGNED	4	PTXRSV04	RESERVED
		1.11 .1..		PTXEND	"*" END OF THE PTX
		.11. 11..		PTXLEN	"*-PTX" SET THE LENGTH OF THE PTX
THE CMS GLOBAL TABLE DEFINES THE STATE OFVS APL IN THE CURRENT MACHINE					
180	(B4)	CHARACTER	8	CMSGLID	LITERAL TO VERIFY CONTROL BLOCK
188	(BC)	A-ADDRESS	4	CMSGRING	ROUND AND ROUND...
NEXT 4 WORDS WILL NEVER BE USED BY VS APL HOWEVER, OFFSET MAY CHANGE DUE TO PTH,PTX ETC. ABOVE					
192	(C0)	SIGNED	4	CMSUSER0	AND NOW,
196	(C4)	SIGNED	4	CMSUSER1	A FEW WORDS
200	(C8)	SIGNED	4	CMSUSER2	TO OUR SPONSOR...
204	(CC)	SIGNED	4	CMSUSER3	
NEXT 8 WORDS RESERVED FOR IBM USE WITH OPTIONAL FEATURES					
208	(D0)	SIGNED	4	IBMOPT1	
212	(D4)	SIGNED	4	IBMOPT2	
216	(D8)	SIGNED	4	IBMOPT3	
220	(DC)	SIGNED	4	IBMOPT4	
224	(E0)	SIGNED	4	IBMOPT5	
228	(E4)	SIGNED	4	IBMOPT6	
232	(E8)	SIGNED	4	IBMOPT7	
236	(EC)	SIGNED	4	IBMOPT8	
240	(F0)	FLOATING	8	CMSDIAG0	CP EXTENDED IDENTIFICATION
240	(F0)	CHARACTER	8	CMSDSYS	SYSTEM NAME ('VM/370')
248	(F8)	HEX	3	CMSDVERS	VERSION, LEVEL, PLC
251	(FB)	HEX	1	CMSDCODE	STIDP
252	(FC)	HEX	2		MCEL
254	(FE)	HEX	2		STAP

CMSGL (CMS, XSYS, AP) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
256	(100)	CHARACTER	8	CMSDUSER	USERID
264	(108)	HEX	8	CMSDPP	PP BIT MAP
272	(110)	A-ADDRESS	4	CMSSOPT	A(OPTIONS) MAPPED BY APLSOPT
ACTIVE WORKSPACE STATUS.					
276	(114)	A-ADDRESS	4	CMSWSADR	ADDRESS OF INCORE WORKSPACE
280	(118)	SIGNED	4	CMSMAXWS	MAXIMUM ALLOWED WS SIZE. (SEE PTHWSLEN FOR CURR SIZE)
ACTIVE WORKSPACE ID.					
284	(11C)	HEX	24	CMSAWSID	ACTIVE WSID.
284	(11C)	SIGNED	4	CMSALIB	LIB NUMBER.
288	(120)	CHARACTER	12	CMSANAM	WS NAME (Z-CODES).
300	(12C)	CHARACTER	8	CMSAPAS	PASSWORD FROM LAST)SAVE OR)LOAD.
308	(134)	HEX	24	CMSAVACT	SAVE ACTIVE WSID THRU)COPY OR)WSID.
332	(14C)	SIGNED	4	CMSRS01F(3)	RESERVED
LIBRARY MANAGEMENT PARAMETERS AND CONTROL FIELDS.					
344	(158)	HEX	1	CMSLIBFL	LIBRARY MGMT FLAGS.
		1...		CMSNOLIB	"BIT0" LIBRARY TABLE IS EMPTY-- NO PUBLIC OR PROJECT LIBS.
		.1..		CMSPRIVT	"BIT1" =1 IF CURRENT LIB OPERATION ACCESSES PRIVATE LIB. =0 IF PUBLIC OR PROJECT.
		..1.		CMSPUBLC	"BIT2" (USED ONLY IF LIBPRIVT=0.) =1 IF CURR LIB OP IS FOR PUBLIC LIB. =0 IF CURR LIB IS PROJECT.
		...1		CMSOLDWS	"BIT3" DURING)SAVE, WS EXISTS IN LIBRARY BEFORE)SAVE.
		1		CMSLTPTH	"BIT7" =1 IF DEFAULT LIB (PTHACCNO) IS IN LIB TABLE
345	(159)	HEX	1	CMSYYLFL	FLAGS FOR YYLIB HANDLING.
		1...		CMSYYLNT	"BIT0" STORAGE GOTTEN FOR FILENAME NAME TABLE. (ALSO MEANS, IF = 1, THAT YYLIB HAS BEEN REISSUED BECAUSE OF OVERFLOW OF WSMBUFF OR NAME TABLE ON PREVIOUS YYLIB.)
		.1..		CMSYYLNO	"BIT1" NAME TABLE HAS OVERFLOWED. (THERE ARE MORE QUALIFIED WS NAMES FOUND THAN THERE ARE TABLE ENTRIES.)
		..1.		CMSYYLBO	"BIT2" WSMBUFF OVERFLOWED. (WE COULDN'T FIT ALL FILENAMES IN NAME TABLE INTO WSMBUFF.)
346	(15A)	HEX	1	CMSRS02X(2)	RESERVED
LIBRARY TABLE POINTERS.					
348	(15C)	SIGNED	4	CMSLIBXL	NEXT 3 WORDS MUST BE CONTIGUOUS.
		..1. 11..		CMSLTL	"44" SIZE OF LIBRARY TABLE ENTRY.
348	(15C)	A-ADDRESS	4	CMSLTADR	ADDRESS OF LIBRARY TABLE. DEFAULTS TO A(CMSLIBTB).

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
352	(160)	A-ADDRESS	4	CMSLTLF	WORD CONTAINING CMSLTL.
356	(164)	SIGNED	4	CMSLTSIZ	NUMBER OF LIB TABLE ENTRIES IN USE, TIMES CMSLTL.

FOLLOWING EQUATES DEFINE FORMAT OF ENTRY IN INCORE LIBRARY TABLE.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
		....		CMSLTFLG	"0" OFFSET TO FLAG FIELD (1 BYTE)
		1...		CMSLTPRV	"BIT0" =0 IF LIB IS PUBLIC OR PROJECT =1 IF LIB IS PRIVATE
		.1..		CMSLTPRJ	"BIT1" =0 IF LIB IS PUBLIC. =1 IF LIB IS PROJECT.
		..1.		CMSLTRNG	"BIT2" =0 IF SINGLE LIB NUMBER. =1 IF RANGE OF LIB NUMBERS.
		...1		CMSLTACC	"BIT3" =0 IF DYNAMICALLY LINKED DISK =1 IF PERMANENT ACCESS
		1		CMSLTDSK	"1,3" OWNER DISK ADDRESS, IN EBCDIC. BLANKS IF NOT USED. THREE BYTES.
		11		CMSLTDSL	"3" LENGTH OF DISK ADDR.
		.... .1..		CMSLTOWN	"4,8" OWNER USERID, IN EBCDIC. BLANKS IF NOT USED. 8 BYTES
		.... 1...		CMSLTOWL	"8" LENGTH OF OWNER'S USERID.
		.... 11..		CMSLTLMO	"12,2" WRITE LINK MODE FOR DISK
		.... 111.		CMSLTAMO	"14,2" ACCESS MODE FOR DISK
		...1		CMSLTLB1	"16,4" LIB NUMBER, OR LOWER LIMIT OF LIB NUMBER RANGE. FULLWRD INTEGER.
		...1 .1..		CMSLTLB2	"20,4" UPPER LIMIT OF RANGE OF LIB NUMBERS. FULLWORD INTEGER. USED ONLY IF CMSLTFLG.CMSLTRNG=1.
		...1 1...		CMSLTCUU	"24,4" LINKED ADDRESS 'CUU '
		...1 11..		CMSLTLC	"28,4" TOTAL LINK COUNT
		..1.		CMSLTWC	"32,4" WRITE LINK COUNT
		..1. .1..		CMSLTRPW	"36,8" READ PASSWORD

DATE/TIME WORKSPACE WAS SAVED, TO BE RETURNED TO INTERP. VIA PDSPASS.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
360	(168)	FLOATING	8	CMSAVDAT	TIME/DATE IN FLOATING PT.

WORK AREA FOR BUILDING PARMLISTS
USED FOR CP + CMS COMMANDS, FID
MAY BE USED BY ANY MODULE BETWEEN APLCALL'S
APLSCFID WILL NOT USE THIS AREA

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
368	(170)	FLOATING	8	CMSWORK(16)	128 BYTES OF ALIGNED STORAGE

WORK AREAS FOR BUILDING FSCB'S.
CMSFSCB1 FSCB 'FILENAME FILETYPE FM',RECFM=F,NOREC=1,BSIZE=4096

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
496	(1F0)	SIGNED	4	CMSFSCB1	
496	(1F0)	CHARACTER	8		
504	(1F8)	CHARACTER	8		
512	(200)	CHARACTER	8		
520	(208)	CHARACTER	2		
522	(20A)	A-ADDRESS	2		
524	(20C)	A-ADDRESS	4		

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
528	(210)	SIGNED	4		
532	(214)	CHARACTER	2		
534	(216)	A-ADDRESS	2		
536	(218)	A-ADDRESS	4		
540	(21C)	SIGNED	4	(4)	ADDITIONAL SPACE FOR FORM=EFSCBS
		..11 11..		CMSFSCBL	"*-CMSFSCB1" LENGTH OF FSCB.
SECOND FSCB USED ONLY BY SCOPY FOR WORK FILE CMSFSCB2 FSCB 'FILENAME FILETYPE FM',RECFM=F,NOREC=1,BSIZE=1026					
556	(22C)	SIGNED	4	CMSFSCB2	
556	(22C)	CHARACTER	8		
564	(234)	CHARACTER	8		
572	(23C)	CHARACTER	8		
580	(244)	CHARACTER	2		
582	(246)	A-ADDRESS	2		
584	(248)	A-ADDRESS	4		
588	(24C)	SIGNED	4		
592	(250)	CHARACTER	2		
594	(252)	A-ADDRESS	2		
596	(254)	A-ADDRESS	4		
600	(258)	SIGNED	4	(4)	(AS ABOVE FOR FORM=E)
THE FOLLOWING GROUP OF FIELDS IS USED BY YYLIB.					
616	(268)	A-ADDRESS	4	CMSNTPTR	ADDR OF NAME TABLE FOR YYLIB
620	(26C)	SIGNED	4	CMSNTSIZ	SIZE OF NAME TABLE (DBLWORDS
624	(270)	A-ADDRESS	4	CMSNTZ	ADDR OF END OF NAME TABLE.
628	(274)	SIGNED	4	CMSYYLNP	NUMBER OF ENTRIES IN NAME TABLE THAT HAVE NOT YET BEEN PUT INTO WSMBUFF TO BE PRINTED.
632	(278)	A-ADDRESS	4	CMSNTHI	ADDR OF NEXT AVAILABLE ENTRY IN NAME TABLE.
636	(27C)	CHARACTER	11	CMSYYLNL	CURRENT NAME, INCLUDING SUFFIX OF 3 BLANKS, FOR COMPARISON TO PDSNAME.
636	(27C)	CHARACTER	8	CMSYYLNN	NEW NAME, AS OBTAINED FROM FST AND TRANSLATED TO Z-CODES.
644	(284)	CHARACTER	3	CMSYYLNS	NAME SUFFIX FOR CMSYYLNN (THREE ZBLANKS).
647	(287)	CHARACTER	1	CMSLIBMD	ACCESS MODE FOR ANY LIB REQUEST
648	(288)	CHARACTER	8	CMSNTONM	NAME TABLE OVERFLOW NAME. THIS IS THE LOWEST ALPHABETIC NAME WHICH WAS NOT PUT INTO THE NAME TABLE, DUE TO OVERFLOW. ALL NAMES IN THE TABLE ARE LOWER THAN THIS.

CMSGL (CMS, XSYS, AP) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
WORK AREAS FOR COPY.					
656	(290)	SIGNED	4	CMSCOPSA	NUMBER OF RECORDS IN SINKWS FILE USED TO SAVE AREA BETWEEN (1) BEGINNING OF WS AND (2) SPOT POINTED TO BY WSMFREEA. (PART A.)
660	(294)	SIGNED	4	CMSCOPSB	NUMBER OF RECORDS IN SINKWS FILE USED TO SAVE AREA BETWEEN (1) SPOT POINTED TO BY WSMFREEZ AND (2) END OF WORKSPACE. (PART B.)
664	(298)	A-ADDRESS	4	CMSCOPSZ	ADDRESS OF SPOT IN WORKSPACE AREA WHERE PART B OF SINKWS FILE STARTS.
668	(29C)	SIGNED	4	CMSAVSIZ	SAVE SIZE OF ACTIVE (SINK) WS HERE WHILE SOURCE IS LOADED.
672	(2A0)	SIGNED	4	CMSICTR	COUNT OF RECORDS READ FROM COPYDATA FILE.
676	(2A4)	SIGNED	4	CMSOCTR	COUNT OF RECORDS WRITTEN TO COPYDATA FILE.
680	(2A8)	SIGNED	4	CMSIREC	RECNO PARAMETER FOR NEXT FSREAD OF COPYDATA FILE
684	(2AC)	SIGNED	4	CMSOREC	RECNO PARAMETER FOR NEXT FSWRITE OF COPYDATA FILE.
688	(2B0)	SIGNED	2	CMSCOPLL	SAVE LENGTH OF COPIED- OBJECTS LIST HERE.
690	(2B2)	SIGNED	2	CMSNKMOD	MODE OF SINKWS FILE IS SAVED HERE.
FIELDS USED DURING)SAVE FOR MANIPULATION OF TEMPORARY WS FILE.					
616	(268)	CHARACTER	8	CMSNTYPE	SAVE FILETYPE OF SAVED WS.
624	(270)	FLOATING	8		PARMS FOR RENAME OF SAVED WS.
624	(270)	CHARACTER	8	CMSRENAM	'RENAME'.
632	(278)	CHARACTER	8	CMSOLDN	OLD FILENAME.
640	(280)	CHARACTER	8	CMSOLDT	OLD FILETYPE.
648	(288)	CHARACTER	2	CMSOLDM1	OLD FILEMODE (FIRST 2 CHARS)
650	(28A)	CHARACTER	6	CMSOLDM2	OLD FILEMODE (LAST 6 CHARS)
656	(290)	CHARACTER	8	CMSNEWN	NEW FILENAME.
664	(298)	CHARACTER	8	CMSNEWT	NEW FILETYPE.
672	(2A0)	CHARACTER	2	CMSNEWM1	NEW FILEMODE (FIRST 2 CHARS)
674	(2A2)	CHARACTER	6	CMSNEWM2	NEW FILEMODE (LAST 6 CHARS)
680	(2A8)	HEX	8	CMSRENZ	END OF PARMS (HEX F'S)
692	(2B4)	SIGNED	4	CMSRS03F(4)	RESERVED
PROGRAM MANAGEMENT. ADDRESS OF APLMAIN MODULE.					
708	(2C4)	A-ADDRESS	4	CMSAMAIN	

CMSGL (CMS, XSYS, AP) continued

```
 OFFSETS     TYPE      LENGTH  NAME            DESCRIPTION

==============================================================================
REGISTER SAVE AREAS FOR SUPERVISOR.
------------------------------------------------------------------------------
 712   (2C8) SIGNED          4  CMSAVE          THREE SAVE AREAS
                                CMSAVEZ         "*" MARKS END OF SAVE AREAS.
------------------------------------------------------------------------------
 928   (3A0) A-ADDRESS       4  CMSAVEZP        ADDR OF END OF SAVE AREAS.
             .1.. 1...          CMSBMPSV        "18*4" TO BUMP TO NEXT SAVE AREA.
------------------------------------------------------------------------------
 932   (3A4) HEX             1  CMSPGMFL        PROGRAM MANAGEMENT FLAGS.
             1... ....          CMSHRSYS        "BIT0" =1 IF THIS IS SHARED APL
                                                SYS
             .1.. ....          CMSINSVP        "BIT1" =1 IF PROGRAM CONTROL HAS
                                                BEEN GIVEN TO THE SVP (WHICH IN
                                                TURN GIVES CONTROL TO AN AP).
             ..1. ....          CMSCOPER        "BIT2" SYSTEM ERROR OCCURRED WHILE
                                                IN COPY STATUS. SET BY YYSYSER,
                                                CHECKED BY YYCOPZ.
             .... 1...          CMSABEX         "BIT4" =1 IF COMMON STAE EXIT
                                                EXISTS
             .... .1..          CMSCSUB         "BIT5" =1 IF CMS CMDS MUST BE
                                                SUBSET
             .... ..1.          CMSBSEPP        "BIT6" =1 IF SYSTEM IS BSEPP
             .... ...1          CMSVMSP         "BIT7" =1 IF SYSTEM IS VM/SP
 933   (3A5) HEX             1  CMSRS04X        RESERVED
==============================================================================
CMSASTOP WILL CONTAIN A 'BCR' INSTRUCTION WHEN IT IS
NECESSARY TO STOP THE VIRTUAL MACHINE (VIA ADSTOP) ON
INTERPRETER SYSTEM ERRORS OR SUPERVISOR ABENDS.
 934   (3A6) SIGNED          2  CMSASTOP        SEE COMMENT ABOVE.
==============================================================================
 LIST FORM OF STAE MACRO GOES HERE.
CMSTAE  STAE  ,MF=L,PURGE=QUIESCE,ASYNCH=NO
------------------------------------------------------------------------------
 936   (3A8) SIGNED          4
------------------------------------------------------------------------------
 936   (3A8) A-ADDRESS       1  CMSTAE          FLAGS FOR TCB, PURGE AND ASYNCH
 937   (3A9) A-ADDRESS       3                  EXIT ADDR. NOT SPECIFIED
------------------------------------------------------------------------------
 940   (3AC) A-ADDRESS       4                  PARM. LIST ADDR. NOT SPECIFIED
------------------------------------------------------------------------------
 944   (3B0) A-ADDRESS       4                  TCB NOT SPECIFIED
             .... 11..          CMSTAEL         "*-CMSTAE" LENGTH OF STAE LIST
                                                FORM.
==============================================================================
LIST FORM OF SPIE (PICA) GOES HERE.
------------------------------------------------------------------------------
 948   (3B4) SIGNED          4
             .... .11.          CMSPICAL        "6" LENGTH OF PICA.
------------------------------------------------------------------------------
 948   (3B4) HEX             1  CMSPICA         OUR PICA.
==============================================================================
THIS IS THE ECB AND ASSOCIATED FLAGS USED WHEN
PUTTING THE APL PROGRAM TO SLEEP ON YYDELAY AND YYRWAIT.
 956   (3BC) SIGNED          4  CMSECB          THE WAIT ECB.
```

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
960	(3C0)	HEX	1	CMSWAITF	SHOWS WHY WE ARE WAITING ON CMSECB.
		1...		WAITRPLY	"BIT0" WAITING FOR ATTENTION TO UNLOCK KEYBOARD AFTER SENDING MESSAGE.
		.1..		WAITIMER	"BIT1" WAITING FOR ATTENTION OR TIMER POP FOR YYDELAY.
		..1.		CMSTIMEP	"BIT2" INSPECTED BY YYDELAY AFTER FALLING OUT OF WAIT MACRO. =1 IF TIMER EXIT POSTED ECB. =0 IF ATTN EXIT POSTED ECB.
		...1		CMSVWAIT	"BIT3" WAITING FOR DOUBLE ATTN TO BREAK SHARED VARIABLE DEADLOCK.
				CMSMINDL	"1000000" MINIMUM WAIT TIME FOR YYDELAY, IN MICROSECONDS.
CMSENDRT CONTAINS THE ADDRESS OF THE APL TERMINATION ROUTINE IN MODULE APLSCINI. IT IS CALLED BY YYOFF.					
964	(3C4)	A-ADDRESS	4	CMSENDRT	SEE COMMENT ABOVE.
CMSPSTIM IS WHERE CP PUTS THE THE RESULT OF THE PSEUDO TIMER DIAGNOSE INSTRUCTION. THIS FOR THE YYDUMP SERVICE					
968	(3C8)	FLOATING	8	CMSPSTIM(4)	DIAG 00C NEEDS 4 DOUBLE
968	(3C8)	CHARACTER	16	CMSPSDT	MM/DD/YYHH:MM:SS (EXACTLY).
968	(3C8)	CHARACTER	8	CMSPDATE	PSEUDO DATE.
976	(3D0)	CHARACTER	8	CMSPTIME	PSEUDO TIME.
984	(3D8)	FLOATING	8	CMSPVIRT	VIRTUAL CPU TIME USED SINCE LOGON, IN MICROSECONDS, UNSIGNED 64-BIT INTEGER.
992	(3E0)	FLOATING	8		IGNORED. SEE VM MANUAL.
CMSDMPNO CONTAINS THE DUMP NUMBER THAT IS PUT INTO SYSTEM ERROR MESSAGES BY THE INTERPRETER. IT IS RETURNED BY					
1000	(3E8)	SIGNED	4	CMSDMPNO	SEE ABOVE.
1004	(3EC)	SIGNED	4	CMSRS05F(4)	RESERVED
TERMINAL MANAGEMENT.					
1020	(3FC)	HEX	1	CMSIDLSW	=ENL IF IDLES REQ'D, ELSE 0.
1021	(3FD)	HEX	1	CMSNLSW	=ENL IF NEW-LINE SEEN, ELSE X'00'. USED FOR WSMPARM2 CHECK ON YYTYO.
1022	(3FE)	HEX	1	CMSFLAGS	TERMINAL MANAGEMENT FLAGS
		1...		CMSSEGZ	"BIT0" SEGMENT IS LAST ONE IN CMSBF
		.1..		CMSRFLAG	"BIT1" READING FROM TERMINAL
		..1.		CMSTYOI	"BIT2" TYO CALLED FROM TYOI
		...1		CMSLAST	"BIT3" FINAL TYO OUTPUT
		.... 1...		CMSOUT	"BIT4" O-U-T SIGNALLED ON DISPLAY TERMINAL.
		.... .1..		CMSNLREQ	"BIT5" ON YYTYO FOR DISPLAY TERM, INPUT PARM SAYS NEW-LINE CHAR MUST BE AT END OF OUTPUT.
		.... ..1.		CMSQUIET	"BIT6" RUNNING WITHOUT A TERMINAL
		1		CMSDSMAV	"BIT7" DISPLAY SESSION MG AVAILABLE
		.1.. 1111		CMS3270W	"79" DEFAULT WIDTH IF 3270.
		.111 1...		CMS2741W	"120" DEFAULT WIDTH IF 2741 OR OTHER TYPEWRITER TERMINAL.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
CMSBUFF POINTERS. USED DURING TYO TO KEEP TRACK OF WHAT HAS BEEN PUT IN CMSBUFF.					
1024	(400)	A-ADDRESS	4	CMSBFLIN	BEGINNING OF CURRENT LINE IN CMSBUFF (TYPEWRITER ONLY)
				CMSHELD	"CMSBFLIN" FOR DISPLAY TERMINALS, CONTAINS LENGTH OF DATA HELD FROM PREVIOUS YYTYO.
1028	(404)	A-ADDRESS	4	CMSBFSEG	BEGINNING OF CURRENT SEGMENT IN CMSBUFF. (FOR TYPEWRITER TERMINAL ONLY.)
				CMSINBUF	"CMSBFSEG" LENGTH OF INPUT BUFFER FOR DISPLAY TERMINAL (=135 IF USING 3270).
PARMLISTS FOR RDTERM AND WRTERM MACROS.					
1032	(408)	SIGNED	4	CMSXPLST	WRTERM PLIST USED BY ATTENTION (STAX) EXIT.
1032	(408)	CHARACTER	8		
1040	(410)	HEX	1		UNUSED HISTORIC BIT.
1041	(411)	A-ADDRESS	3	CMSXADDR	OUTPUT ADDRESS.
1044	(414)	CHARACTER	1		BLACK RIBBON.
1045	(415)	HEX	1		LONG WRITE, EDIT=NO.
1046	(416)	SIGNED	2	CMSXLGTH	OUTPUT LENGTH.
		...1		CMSXPLL	"*-CMSXPLST" LENGTH OF PLIST.
1048	(418)	SIGNED	4	CMSWPLST	WRTERM PLIST.
1048	(418)	CHARACTER	8		
1056	(420)	HEX	1		UNUSED HISTORIC BIT.
1057	(421)	A-ADDRESS	3	CMSWADDR	OUTPUT ADDRESS.
1060	(424)	CHARACTER	1		BLACK RIBBON.
1061	(425)	HEX	1		LONG WRITE, EDIT=NO.
1062	(426)	SIGNED	2	CMSWLGTH	OUTPUT LENGTH.
		...1		CMSWPLL	"*-CMSWPLST" LENGTH WRTERM PLIST.
1064	(428)	SIGNED	4	CMSRPLST	RDTERM PLIST.
1064	(428)	CHARACTER	8		
1072	(430)	HEX	1		UNUSED HISTORIC BIT.
1073	(431)	A-ADDRESS	3	CMSRADDR	INPUT BUFFER ADDRESS.
1076	(434)	CHARACTER	1		ATTREST=NO OPTION.
1077	(435)	HEX	1		UNUSED.
1078	(436)	SIGNED	2	CMSRLGTH	INPUT LENGTH.
		...1		CMSRPLL	"*-CMSRPLST" LENGTH OF RDTERM PLIST.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
'CMSDCCW' DEFINES A SPECIAL PSEUDO-CCW USED WITH DIAGNOSE X'58' TO DISPLAY PROMPTS IN THE INPUT AREA OF A SCREEN. ALL FIELDS EXCEPT THE LENGTH FIELD ARE INITIALIZED BY APLSCINI; THE CCW IS USED IN MODULE APLSCDPY.					
1032	(408)	FLOATING	8	CMSDCCW	
1032	(408)	HEX	1		OP-CODE.
1033	(409)	A-ADDRESS	3	CMSDCCWA	ADDR OF DATA.
1036	(40C)	HEX	1		STANDARD CCW FLAGS.
1037	(40D)	HEX	1		LINE NUMBER ON SCREEN.
1038	(40E)	SIGNED	2	CMSDCCWL	DATA LENGTH
TERMINAL DEVICE INFORMATION. DESCRIPTOR BITS FOR REAL CONSOLE DEVICE, AS RETURNED BY DIAGNOSE 24.					
1080	(438)	SIGNED	4	CMSTYCON	DESCRIP BITS ARE PUT HERE.
		....		RDEVTYPC	"0" OFFSET OF BYTE IN TYCON GIVING DEVICE CLASS.
		1...		CLASTERM	"BIT0" CLASS IS TYPEWRITER DEVICE. (CLASTERM BIT SET IF REMOTE 3270. APLSCINI WILL RESET BITS TO LOOK LIKE LOCAL 3270).
		.1..		CLASDPY	"BIT1" CLASS IS DISPLAY (GRAPHICS) DEVICE.
		1		RDEVTYPE	"1" OFFSET OF BYTE IN TYCON GIVING DEVICE TYPE.
		11		RDEVLLEN	"3" OFFSET TO BYTE CONTAINING LINE LENGTH.
		...1 1...		CMS2741	"BIT3+BIT4" TYPE IS 2741.
		...1 .1..		CMS1050	"BIT3+BIT5" TYPE IS 1050.
		.... .1..		CMS3270	"BIT5" TYPE IS 3270.
		1...		CMSR3270	"BIT0" TYPE IS REMOTE 3270 (BIT 'CLASTERM' IS ALSO SET).
PARMLIST FOR STAX MACRO. CMSTAXPL STAX 0,MF=LIST FORM OF STAX. MOVED TO THIS SPOT BY INITIALIZATION.					
1084	(43C)	SIGNED	4	CMSTAXPL	
1084	(43C)	A-ADDRESS	4		ADDRESS OF EXIT ROUTINE
1088	(440)	A-ADDRESS	2		LENGTH OF INPUT BUFFERS
1090	(442)	A-ADDRESS	2		LENGTH OF OUTPUT BUFFERS
1092	(444)	A-ADDRESS	4		ADDRESS OF OUTPUT BUFFERS
1096	(448)	A-ADDRESS	4		ADDRESS OF INPUT BUFFERS
1100	(44C)	A-ADDRESS	1		REPLACE/NO REPLACE, DEFERRAL IND
1101	(44D)	A-ADDRESS	3		ADDRESS OF USER PARAMETERS
		...1 .1..		CMSTAXL	"*-CMSTAXPL" LENGTHE OF STAX PARMLIST.
		....		STXEXIT	"0" OFFSET TO FIELD IN CMSTAXPL CONTAINING ADDR OF STAX EXIT ROUTINE.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
PLIST FOR 'ASVPHINT' MACRO. THE SSM 'ASVPHINT' MACRO IS EXECUTED IF THIS IS A DISPLAY (3270) TERMINAL. IF 'TERM APL ON' IS SET AND THIS IS A DISPLAY TERMINAL, VM/370 WILL GIVE AN EXTERNAL INTERRUPT WHENEVER THE PA2 KEY IS STRUCK. APL USES THE PA2 TO SIGNAL CANCEL-OUTPUT.					
1084	(43C)	HEX	12	CMSEXTPL	PLIST FOR 'ASVPHINT' MACRO.
1084	(43C)	A-ADDRESS	4	CMSEXTID	ACCOUNT NUMBER FROM PTHACCNO.
1088	(440)	V-ADDRESS	4	CMSEXTAD	"V(SCDPA2)" ADDR OF EXTERNAL INTERRUPT EXIT HANDLER. HANDLER SETS CANCEL-OUTPUT BIT.
1092	(444)	A-ADDRESS	1	(4)	SEE ASVICV FOR FIRST TWO BYTES ASYN LAST TWO BYTES ARE EXT INT CODE.
		.... 11..		CMSEXPLL	"*-CMSEXTPL" LENGTH OF PLIST.
1104	(450)	HEX	256	CMSTABS	CURRENT TAB SETTING. (ALL 0 IF NO TABS.)
ADDRESSES OF DEVICE-DEPENDENT SERVICE REQUEST HANDLERS. THE ADDRESSES IN THESE FIELDS DEPEND ON WHETHER THE TERMINAL IS A TYPEWRITER OR A DISPLAY (3270). APLSCINI STORES THE ADDRESSES HERE, APLSCFXI USES THEM.					
1360	(550)	SIGNED	4	CMSDDADR	
1360	(550)	A-ADDRESS	4	CMSXTYI	ADDRESS OF YYTYI HANDLER.
1364	(554)	A-ADDRESS	4	CMSXTYO	ADDRESS OF YYTYO HANDLER.
1368	(558)	A-ADDRESS	4	CMSXTYOI	ADDRESS OF YYTYOI HANDLER.
CONSOLE ADDRESS. NEEDED FOR DIAGNOSE 58 IF THIS IS A 3270. SET BY APLSCINI, USED BY APLSCDPY.					
1372	(55C)	A-ADDRESS	4	CMSCONAD	SEE COMMENT ABOVE. 2143
1376	(560)	SIGNED	4	CMSRS06F(4)	RESERVED
STORAGE MANAGEMENT. THE ENTRY-POINT AND WORKAREA ADDRESSES FOR THE FIRST TEN AUXILIARY PROCESSORS ARE KEPT HERE. THE FIRST WORD OF EACH WORD-PAIR HAS THE ENTRY POINT ADDRESS, THE SECOND WORD HAS THE WORKAREA ADDRESS.					
1392	(570)	A-ADDRESS	4	CMSAPAL(20)	SEE COMMENT ABOVE.
		.... 1.1.		CMSAPALL	"(*-CMSAPAL)/8" NUMBER OF ENTRIES.
KEEP ADDRESS AND LENGTH OF AREA WE GOT IN USER PROGRAM AREA FOR WORKSPACE, SHARED MEM AND AP WORK AREAS. WE USE THESE TO FREE THE SPACE AT YYOFF.					
1472	(5C0)	SIGNED	4	CMSFRADR	ADDRESS OF WS, ETC. AREA.
1476	(5C4)	SIGNED	4	CMSFRSIZ	LENGTH OF AREA, IN DOUBLEWORDS.
				CMSAPWKL	"512" WORK AREA FOR EACH AP
1480	(5C8)	SIGNED	4	CMSRS07F(4)	RESERVED

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
SHARED VARIABLES.					
1496	(5D8)	HEX	1	CMSSHVFL	SHARED VARIABLE FLAGS.
		1...		SHVAVAIL	"BIT0" =1 IF SHARED VARIABLES CAN BE USED DURING THIS SESSION.
		.... 1...		SHVRPEAT	"BIT4" IF =1, WE REPEAT A REFERENCE OR OFFER REQUEST ONCE TO PREVENT FALSE RESULTS WITH CERTAIN DISTRIBUTED AUX. PROCESSORS. IF =0, REQUEST HAS NOT BEEN REPEATED AND MAY HAVE TO BE FOR CERTAIN RETURN/REASON CODES.
		.... .1..		SHVNOAP	"BIT5" A.P.'S NOT LOADABLE 2018
1497	(5D9)	HEX	1	CMSRS08X(3)	RESERVED
THIS IS THE ECB LIST INFORMATION THAT WE PASS TO THE SVP WHEN WE DO A SHARED VARIABLE WAIT.					
1500	(5DC)	A-ADDRESS	4	CMSECBLA	ADDR OF ECB LIST.
1504	(5E0)	A-ADDRESS	4	CMSVECBA	ADDR OF ECB AREA.
1508	(5E4)	SIGNED	4	CMSVPECB	PCV ECB FOR SH VAR WAIT.
		.... .1..		CMSECBSP	"4" SIZE OF ECB OR ECB LIST ELMT
1512	(5E8)	A-ADDRESS	4	CMSSSMAD	ADDR OF SHARED STORAGE MANAGER.
1516	(5EC)	SIGNED	4	CMSIOE14	I/O INTERRUPT RETURN REG
1520	(5F0)	SIGNED	4	CMSEIR13	EXT INT. R13 POINTER
1524	(5F4)	A-ADDRESS	4	CMSEIOLD	ADDRESS OF OLD EXT INT EXIT
1528	(5F8)	A-ADDRESS	4	CMSTSKBL	ADDRESS OF INT TSK BLOCK
1532	(5FC)	SIGNED	4	CMSRS09F(2)	RESERVED
SHARED VARIABLE INFORMATION THAT IS PASSED TO THE SVP AT APL STARTUP TO INITIALIZE THE SHARED VARIABLE FACILITY.					
1540	(604)	SIGNED	4	CMSSVPIN	THE FOLLOWING 3 WORDS MUST BE CONTIGUOUS.
1540	(604)	SIGNED	4	CMSNUMAP	NUMBER OF AP'S LOADED.
1544	(608)	SIGNED	4	CMSSMSIZ	SIZE OF SHARED MEMORY.
1548	(60C)	A-ADDRESS	4	CMSSMADR	THE ADDRESS OF SHARED MEMORY
1552	(610)	SIGNED	4	CMSRS10F(4)	RESERVED
1568	(620)	A-ADDRESS	4	CMSAPADA(300)	60 5-WORDS FOR SVP PARMS. ONE SET FOR EACH AP. SEE MODULE APLSCINI FOR DEFINITION OF SETS. (CMSBUFF IS AT SAME LOCATION AS CMSAPADA.)
2768	(AD0)	FLOATING	8	CMSINIBF	LABEL USED FOR BUFFER SPACE DURING INITIALIZATION
THE TERMINAL I/O BUFFER.					
1568	(620)	CHARACTER	2048	CMSBUFF	THE BUFFER.
				CMSBUFFZ	"*" MARKS END OF CMSBUFF.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
MISCELLANEOUS.					
3616	(E20)	FLOATING	8	CMSPACK	CONVERTS LIB NUMBERS TO EBCD
3624	(E28)	SIGNED	4	CMSRECNO	RECORD NO. FOR FSWRITE OF FILE FOR YYCOPO.
3628	(E2C)	SIGNED	2	CMSCLISL	HOLD COPY LIST LENGTH.
WORK AREA USED FOR EDITING SUPERVISOR MESSAGES WITH LINEDIT MACRO.					
CMSLINED LINEDIT MF=L,MAXSUBS=5					
3630	(E2E)	HEX	1	CMSLINED(47)	
CMSINITF IS USED ONLY DURING INITIALIZATION (SEE APLSCINI) AFTER INITIALIZATION, CMSWORKF IS AVAILABLE TO TO ANY ROUTINE WHICH DOES NOT GIVE UP CONTROL VOLUNTARILY. (I.E. HANDS OFF, ASYNCH ROUTINES)					
3677	(E5D)	HEX	1	CMSINITF	
3677	(E5D)	HEX	1	CMSWORKF	
THESE FOUR DOUBLEWORDS CONTAIN TIMES, FOR QUAD-AI'S USE.					
3680	(E60)	FLOATING	8	CMSTRTUP	TIME OF DAY THAT APL WAS STARTED, IN APL STANDARD TIME FORMAT.
3688	(E68)	FLOATING	8	CMSKEYTM	PTHKEYTM IS NOW USED INSTEAD OF THIS FIELD.
3696	(E70)	FLOATING	8	CMSCPUAC	ACCUMULATED VIRTUAL CPU TIME FOR INTERPRETER, IN MILLISECONDS.
3704	(E78)	FLOATING	8	CMSHOLDT	HOLD AREA FOR SAVING CPU TIME WHEN INTERP IS DISPATCHED OR TIME OF DAY WHEN KEYBOARD IS UNLOCKED.
THE GLOBAL-TABLE-RESIDENT LIBRARY TABLE. LIB TABLE IS HERE IF THERE ARE NO MORE THAN CMSLLTMX/CMSLTL LOGICAL RECORDS IN THE LIBRARY TABLE					
3712	(E80)	SIGNED	4	CMSLLTMX	LIB TABLE MUST BE ON FULLWRD "((CMSGL+CMSML-*)/CMSLTL-1)*CMSLTL" AMOUNT OF SPACE AVAILABLE IN GLOBAL TABLE FOR LIB TABLE ENTRIES.
3712	(E80)	HEX	1	CMSLIBTB	THE LIBRARY TABLE.
GLOBAL EQUATES.					
		.... 1.1.		CMSR	"R10" GLOBAL TABLE BASE REGISTER.

CLASDPY	1080 X'40'
CLASTERM	1080 X'80'
CMSABEX	932 X'08'
CMSALIB	284(11C)
CMSAMAIN	708(2C4)
CMSANAM	288(120)
CMSAPADA	1568(620)
CMSAPAL	1392(570)
CMSAPALL	1392 X'0A'
CMSAPAS	300(12C)
CMSAPWKL	512
CMSASTOP	934(3A6)
CMSAVACT	308(134)
CMSAVDAT	360(168)
CMSAVE	712(2C8)
CMSAVEZ	928
CMSAVEZP	928(3A0)
CMSAVSIZ	668(29C)
CMSAWSID	284(11C)
CMSBFLIN	1024(400)
CMSBFSEG	1028(404)
CMSBMPSV	928 X'48'
CMSBSEPP	932 X'02'
CMSBUFF	1568(620)
CMSBUFFZ	3616
CMSCLISL	3628(E2C)
CMSCONAD	1372(55C)
CMSCOPER	932 X'20'
CMSCOPLL	688(2B0)
CMSCOPSA	656(290)
CMSCOPSB	660(294)
CMSCOPSZ	664(298)
CMSCPUAC	3696(E70)
CMSCSUB	932 X'04'
CMSDCCW	1032(408)
CMSDCCWA	1033(409)
CMSDCCWL	1038(40E)
CMSDCODE	251 (FB)
CMSDDADR	1360(550)
CMSDIAG0	240 (F0)
CMSDMPNO	1000(3E8)
CMSDPP	264(108)
CMSDSMAV	1022 X'01'
CMSDSYS	240 (F0)
CMSDUSER	256(100)
CMSDVERS	248 (F8)
CMSECB	956(3BC)
CMSECBLA	1500(5DC)
CMSECBSP	1508 X'04'
CMSEIOLD	1524(5F4)
CMSEIR13	1520(5F0)
CMSENDRT	964(3C4)
CMSEXPLL	1092 X'0C'
CMSEXTAD	1088(440)
CMSEXTID	1084(43C)
CMSEXTPL	1084(43C)
CMSFLAGS	1022(3FE)
CMSFRADR	1472(5C0)
CMSFRSIZ	1476(5C4)
CMSFSCBL	540 X'3C'
CMSFSCB1	496(1F0)
CMSFSCB2	556(22C)
CMSGL	0 (0)
CMSGLID	180 (B4)
CMSGRING	188 (BC)
CMSHELD	1024
CMSHOLDT	3704(E78)
CMSHRSYS	932 X'80'
CMSICTR	672(2A0)
CMSIDLSW	1020(3FC)
CMSINBUF	1028
CMSINIBF	2768(AD0)
CMSINITF	3677(E5D)
CMSINSVP	932 X'40'
CMSIOE14	1516(5EC)
CMSIREC	680(2A8)
CMSKEYTM	3688(E68)
CMSLAST	1022 X'10'
CMSLIBFL	344(158)
CMSLIBMD	647(287)
CMSLIBTB	3712(E80)
CMSLIBXL	348(15C)
CMSLINED	3630(E2E)
CMSLLTMX =	308
CMSLTACC	356 X'10'
CMSLTADR	348(15C)
CMSLTAMO	356 X'0E'
CMSLTCUU	356 X'18'
CMSLTDSK	356 X'01'
CMSLTDSL	356 X'03'
CMSLTFLG	356 X'00'
CMSLTL	348 X'2C'
CMSLTLB1	356 X'10'
CMSLTLB2	356 X'14'
CMSLTLC	356 X'1C'
CMSLTLF	352(160)
CMSLTLMO	356 X'0C'
CMSLTOWL	356 X'08'
CMSLTOWN	356 X'04'
CMSLTPRJ	356 X'40'
CMSLTPRV	356 X'80'
CMSLTPTH	344 X'01'
CMSLTRNG	356 X'20'
CMSLTRPW	356 X'24'
CMSLTSIZ	356(164)
CMSLTWC	356 X'20'
CMSMAXWS	280(118)
CMSMINDL	1000000
CMSML	4096
CMSNEWM1	672(2A0)
CMSNEWM2	674(2A2)
CMSNEWN	656(290)
CMSNEWT	664(298)
CMSNKMOD	690(2B2)
CMSNLREQ	1022 X'04'
CMSNLSW	1021(3FD)
CMSNOLIB	344 X'80'
CMSNTHI	632(278)
CMSNTONM	648(288)
CMSNTPTR	616(268)
CMSNTSIZ	620(26C)
CMSNTYPE	616(268)
CMSNTZ	624(270)
CMSNUMAP	1540(604)
CMSOCTR	676(2A4)
CMSOLDM1	648(288)
CMSOLDM2	650(28A)
CMSOLDN	632(278)
CMSOLDT	640(280)
CMSOLDWS	344 X'10'
CMSOREC	684(2AC)
CMSOUT	1022 X'08'
CMSPACK	3616(E20)
CMSPDATE	968(3C8)
CMSPGMFL	932(3A4)
CMSPICA	948(3B4)
CMSPICAL	948 X'06'
CMSPRIVT	344 X'40'
CMSPSDT	968(3C8)
CMSPSTIM	968(3C8)
CMSPTIME	976(3D0)
CMSPUBLC	344 X'20'
CMSPVIRT	984(3D8)
CMSQUIET	1022 X'02'
CMSR	3712 X'0A'
CMSRADDR	1073(431)
CMSRECNO	3624(E28)
CMSRENAM	624(270)
CMSRENZ	680(2A8)
CMSRFLAG	1022 X'40'
CMSRLGTH	1078(436)
CMSRPLL	1078 X'10'
CMSRPLST	1064(428)
CMSRS01F	332(14C)
CMSRS02X	346(15A)
CMSRS03F	692(2B4)
CMSRS04X	933(3A5)
CMSRS05F	1004(3EC)
CMSRS06F	1376(560)
CMSRS07F	1480(5C8)
CMSRS08X	1497(5D9)
CMSRS09F	1532(5FC)
CMSRS10F	1552(610)
CMSR3270	1080 X'80'
CMSSEGZ	1022 X'80'
CMSSHVFL	1496(5D8)
CMSSMADR	1548(60C)
CMSSMSIZ	1544(608)
CMSSOPT	272(110)
CMSSSMAD	1512(5E8)
CMSSVPIN	1540(604)
CMSTABS	1104(450)
CMSTAE	936(3A8)
CMSTAEL	944 X'0C'
CMSTAXL	1101 X'14'
CMSTAXPL	1084(43C)
CMSTIMEP	960 X'20'
CMSTRTUP	3680(E60)
CMSTSKBL	1528(5F8)
CMSTYCON	1080(438)
CMSTYOI	1022 X'20'
CMSUSER0	192 (C0)
CMSUSER1	196 (C4)
CMSUSER2	200 (C8)
CMSUSER3	204 (CC)
CMSVECBA	1504(5E0)
CMSVMSP	932 X'01'
CMSVPECB	1508(5E4)
CMSVWAIT	960 X'10'
CMSWADDR	1057(421)
CMSWAITF	960(3C0)
CMSWLGTH	1062(426)
CMSWORK	368(170)
CMSWORKF	3677(E5D)
CMSWPLL	1062 X'10'
CMSWPLST	1048(418)
CMSWSADR	276(114)
CMSXADDR	1041(411)
CMSXLGTH	1046(416)
CMSXPLL	1046 X'10'
CMSXPLST	1032(408)
CMSXTYI	1360(550)
CMSXTYO	1364(554)
CMSXTYOI	1368(558)
CMSYYLBO	345 X'20'
CMSYYLFL	345(159)
CMSYYLNL	636(27C)
CMSYYLNN	636(27C)
CMSYYLNO	345 X'40'
CMSYYLNP	628(274)
CMSYYLNS	644(284)
CMSYYLNT	345 X'80'
CMS1050	1080 X'14'

CROSS REFERENCE

CMS2741	1080	X'18'
CMS2741W	1022	X'78'
CMS3270	1080	X'04'
CMS3270W	1022	X'4F'
DBGECHO	109	X'02'
DBGMICRO	109	X'80'
DBGMSG	109	X'01'
DBGNSTAE	109	X'40'
IBMOPT1	208	(D0)
IBMOPT2	212	(D4)
IBMOPT3	216	(D8)
IBMOPT4	220	(DC)
IBMOPT5	224	(E0)
IBMOPT6	228	(E4)
IBMOPT7	232	(E8)
IBMOPT8	236	(EC)
PTH	0	(0)
PTHACCNO	36	(24)
PTHASYNC	0	(0)
PTHATTN	0	X'01'
PTHCNCTM	64	(40)
PTHCPULM	0	X'20'
PTHCPUTM	48	(30)
PTHCURSR	18	(12)
PTHCWBIT	3	X'80'
PTHDATTN	0	X'80'
PTHFOFF	0	X'02'
PTHFSAVL	5	X'08'
PTHKEYTM	56	(38)
PTHLOCAL	40	(28)
PTHLOCKB	5	X'80'
PTHMDY	5	X'40'
PTHMICRO	5	X'10'
PTHMSBLK	5	X'20'
PTHNOOUT	0	X'04'
PTHPARM1	24	(18)
PTHPARM2	28	(1C)
PTHQEND	0	X'40'
PTHQSIZE	20	(14)
PTHQVAR	6	(6)
PTHSINK	4	X'02'
PTHSIZE	64	X'48'
PTHSORS	4	X'01'
PTHSPCLY	8	X'80'
PTHSRCOD	10	(A)
PTHSUSP1	3	(3)
PTHSVBIT	3	X'20'
PTHSVON	4	X'80'
PTHUEXTN	5	X'04'
PTHUSTAT	5	(5)
PTHWABIT	3	X'40'
PTHWIDTH	14	(E)
PTHWORD1	0	(0)
PTHWSLEN	32	(20)
PTHWSTAT	4	(4)
PTHYYCOD	8	(8)
PTHYYRC	8	(8)
PTX	72	(48)
PTXADSM	110	X'08'
PTXAIPUR	110	X'80'
PTXATTN	104	(68)
PTXCICS	108	X'20'
PTXCMS	108	X'40'
PTXCODE	120	(78)
PTXDEBUG	109	(6D)
PTXDXTBP	112	(70)
PTXEND	176	X'B4'
PTXFLAG	108	(6C)
PTXFLAGS	110	(6E)
PTXFSRST	110	X'40'
PTXFSTBP	100	(64)
PTXGXGDM	92	(5C)
PTXGXTBP	88	(58)
PTXHELPQ	156	(9C)
PTXHIAOT	153	(99)
PTXHIFLG	155	(9B)
PTXHIIHI	155	X'40'
PTXHIIOT	154	(9A)
PTXHILIT	152	(98)
PTXHIOHI	155	X'80'
PTXHISF	152	(98)
PTXLEN	176	X'6C'
PTXLEVEL	116	(74)
PTXPRTBP	96	(60)
PTXRSV01	164	(A4)
PTXRSV02	168	(A8)
PTXRSV03	172	(AC)
PTXRSV04	176	(B0)
PTXSCRTH	124	(7C)
PTXSMPRO	124	(7C)
PTXSMPSD	124	(7C)
PTXSMP1	124	(7C)
PTXSMP2	128	(80)
PTXSMP3	132	(84)
PTXSMP4	136	(88)
PTXSMP5	140	(8C)
PTXSMP6	144	(90)
PTXSMP7	148	(94)
PTXSMTBP	84	(54)
PTXSTACK	80	(50)
PTXSUBSY	108	(6C)
PTXTSO	108	X'80'
PTXUSRWA	160	(A0)
PTXVCT	76	(4C)
PTXVSPC	108	X'10'
PTXWSM	72	(48)
RDEVLLEN	1080	X'03'
RDEVTYPC	1080	X'00'
RDEVTYPE	1080	X'01'
SHVAVAIL	1496	X'80'
SHVNOAP	1496	X'04'
SHVRPEAT	1496	X'08'
STXEXIT	1101	X'00'
WAITIMER	960	X'40'
WAITRPLY	960	X'80'

DESC (CICS, XSYS, AP)

This is the VS APL variable mapping descriptor used to describe object types as a numeric, scalar, vector, etc. It is mapped by the APLDESC macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	4	APLDESC	APL DESCRIPTOR WORD
0	(0)	BITSTRING	1	APLDESC0	FIRST DESCRIPTOR BYTE
		1111			RESERVED
		.... 1...		APL0APV	ARITH PROGRESSION VECTOR
1	(1)	BITSTRING	1	APLDESC1	SECOND DESCRIPTOR BYTE
		1...		APL1EXTN	TYPE IS DEFINED BY DESC0
		.1..		APL1NOT1	OFF=1 ELEM, ON=0 OR >1
		..1.		APL1ARRY	ON=ARRAY, OFF=SCALAR/VECTOR
		...1		APL1NOTS	OFF=SCALAR, ON=VECTOR/ARRAY
		.... .1..		APL1CHAR	CHARACTER DATA
		11		APL1REAL	REAL (FLOATING) NUMERIC
		.... ..1.			
		1		APL1INTE	INTEGER (BINARY) NUMERIC
2	(2)	SIGNED	2	APLDNN	FOR APL INTERPRETER USE ONLY
4	(4)	CHARACTER	0	APLDATA	V..V BEGINS HERE

CROSS REFERENCE

APLDATA	4	(4)
APLDESC	0	(0)
APLDESC0	0	(0)
APLDESC1	1	(1)
APLDNN	2	(2)
APL0APV	0	X'08'
APL1ARRY	1	X'20'
APL1CHAR	1	X'04'
APL1EXTN	1	X'80'
APL1INTE	1	X'01'
APL1NOTS	1	X'10'
APL1NOT1	1	X'40'
APL1REAL	1	X'03'

DIB (CICS, XSYS)

This is the destination interface block. It controls a CICS/VS transient data destination or a 3270 printer that has been opened by the destination manager. The DIB is passed to the destination manager by the terminal manager, the screen format manager, and auxiliary processor 132. The user perterm points to a chain of DIBs. This control block is mapped by the APLKDIB macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	36	DIB	DESTINATION INTERFACE BLOCK
0	(0)	A-ADDRESS	4	DIBCHAIN	CHAIN OF DIBS FOR USER
4	(4)	CHARACTER	1	DIBREQ	REQUEST TYPE
		1...		DIBOPEN	OPEN THE DIB
		.1..		DIBCLOSE	CLOSE THE DIB
		..1.		DIBREAD	READ A RECORD
		...1		DIBWRITE	WRITE A RECORD
5	(5)	CHARACTER	1	DIBOPTN	OPTIONS KEPT WHILE OPEN
		11..			RESERVED
		..1.		DIBOREAD	OPEN FOR READ
		...1		DIBOWRIT	OPEN FOR WRITE
		.... 1...		DIBENQOP	ENQ FROM OPEN TILL CLOSE
		.... .1..		DIBENQRQ	ENQ FOR I/O ONLY
		.... ..1.		DIBZCODE	CONVERT FROM/TO ZCODE
		1		DIBNFORM	BYPASS FORMATTING
6	(6)	CHARACTER	1	DIBFLGS	PROCESSING STATE
		1...		DIBPRINT	OUTPUT TO PRINT TERMINAL
		.1..		DIBFXLEN	FIXED LENGTH RECORDS
		..1.		DIBCTLA	ANSI CONTROL CHARACTERS
		...1		DIBCTLM	MACHINE CONTROL CHARACTERS
		.... 1...		DIBSYNCP	SYNCPOINT HAS FORCED DEQ
		.... .1..		DIBPLIM	PRINT LIMIT IS IN EFFECT
		.... ..1.		DIBINTRA	INTRAPARTITION DESTIN.
7	(7)	BITSTRING	1	DIBFLG2	RESERVED
8	(8)	CHARACTER	4	DIBDEST	DESTINATION/TERMINAL NAME
12	(C)	CHARACTER	4	DIBXLATE	TRANSLATE TABLE SUFFIX
16	(10)	A-ADDRESS	4	DIBAREA	ADDR OF DATA AREA
20	(14)	SIGNED	2	DIBRLEN	LENGTH OF RECORD
22	(16)	SIGNED	2	DIBMAXLN	MAXIMUM RECORD LENGTH
24	(18)	A-ADDRESS	4	DIBRESRC	ADDR OF TCTTE OR DCT ENTRY
28	(1C)	A-ADDRESS	4	DIBTDOA	ADDR OF A TDOA FOR OUTPUT
32	(20)	SIGNED	4	DIBCNT	NR OF I/O S SINCE OPEN
36	(24)	CHARACTER	0		END OF DIB

CROSS REFERENCE

DIB	0	(0)
DIBAREA	16	(10)
DIBCHAIN	0	(0)
DIBCLOSE	4	X'40'
DIBCNT	32	(20)
DIBCTLA	6	X'20'
DIBCTLM	6	X'10'
DIBDEST	8	(8)
DIBENQOP	5	X'08'
DIBENQRQ	5	X'04'
DIBFLGS	6	(6)
DIBFLG2	7	(7)
DIBFXLEN	6	X'40'
DIBINTRA	6	X'02'
DIBMAXLN	22	(16)
DIBNFORM	5	X'01'
DIBOPEN	4	X'80'
DIBOPTN	5	(5)
DIBOREAD	5	X'20'
DIBOWRIT	5	X'10'
DIBPLIM	6	X'04'
DIBPRINT	6	X'80'
DIBREAD	4	X'20'
DIBREQ	4	(4)
DIBRESRC	24	(18)
DIBRLEN	20	(14)
DIBSYNCP	6	X'08'
DIBTDOA	28	(1C)
DIBWRITE	4	X'10'
DIBXLATE	12	(C)
DIBZCODE	5	X'02'

DIR (CICS, SERV)

This is the APL library directory entry. It is a keyed logical record that generally resides in a VSAM KSDS (the APL directory). A DIR may describe either an APL workspace or a file. Special forms of the DIR describe APL users (mapped by the APLKPRO macro), the library freespace map (mapped by the APLKFSP macro), file extents (mapped by the APLKFEB macro), and signon messages (no special mapping). This control block is mapped by the APLKDIR macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	64	DIR	
0	(0)	CHARACTER	4	DIRHEADR	HEADER TO DIRECTORY ENTRY
0	(0)	SIGNED	2	DIRLENHW	DIRECTORY ENTRY LENGTH
2	(2)	SIGNED	2		RESERVED
4	(4)	CHARACTER	14	DIRKEY	14 BYTE VSAM KEY
4	(4)	UNSIGNED	3	DIRLIBNO	USER LIBRARY NUMBER
7	(7)	CHARACTER	11	DIRWSNAM	WORKSPACE OR FILE NAME
7	(7)	UNSIGNED	1	DIRCODE	CODE FIELD
8	(8)	CHARACTER	7		REST OF 8 BYTE FILE/WS NAME
15	(F)	UNSIGNED	1	DIRFEBID	LOCATION OF FEB IDENTIFIER
16	(10)	CHARACTER	1		RESERVED FOR HI AND FEB
17	(11)	UNSIGNED	1	DIRHICNT	HI MSG SEQUENCE NUMBER
18	(12)	CHARACTER	1	DIRTYPE	TYPE BYTE
		1...		DIRFREE	FREE SPACE RECORD 80
		.1..			RESERVED
		..1.		DIRUPROF	USER PROFILE BIT 20
		...1			RESERVED
		.... 1...		DIRCICS	CICS DIRECTORY ENTRY 08
		.... .1..		DIRPUB	PUBLIC LIBRARY 04
		.... ..1.			RESERVED
		1		DIRPRIV	PRIVATE LIBRARY 01
19	(13)	CHARACTER	1	DIRFLAG1	FLAG BYTE 1
		11..			RESERVED
		..1.		DIRSHR	FILE CAN BE SHARED 20
		...1		DIRSCFLG	ACTIVE SCROLL FILE FLAG 10
		.... 11..			RESERVED
		.... ..1.		DIRPASW	WS OR FILE HAS PASSWORD 02
		1		DIRLOCK	USER IS LOCKED 01
20	(14)	CHARACTER	8	DIRPSWD	WS OR FILE PASSWORD
28	(1C)	BITSTRING	8	DIRSWTS	SAVE WRITE APL STD TIME
36	(24)	SIGNED	4	DIRNCI	NUMBER OF ALLOCATED CI-S
40	(28)	SIGNED	4	DIRLCIDL	DATA WRITTEN IN LAST CI
44	(2C)	CHARACTER	1	DIRCATTR	CONTENT ATTRIBUTE
45	(2D)	CHARACTER	1	DIRFTYPE	TYPE OF ENTRY
		1111			RESERVED
		.... 1...		DIRWS	THIS IS A WORKSPACE 08
		.... .1..			RESERVED
		.... ..1.		DIRSF	THIS IS A SEQ FILE 02
		1		DIRDF	THIS IS A DIRECT FILE 01
46	(2E)	SIGNED	2		RESERVED

DIR (CICS, SERV) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
48	(30)	SIGNED	4	DIRCFLSZ	CURRENT FILE SIZE
52	(34)	SIGNED	4	DIRNLR	NUMBER OF LOGICAL RECORDS
56	(38)	SIGNED	4	DIRLRS	LOGICAL RECORD SIZE
60	(3C)	UNSIGNED	4	DIRLCIWR	LAST CI WRITTEN INTO (EOF)
52	(34)	STRUCTURE	12	DIRFONLY	REDEFINE UNIQUE FIELDS
52	(34)	CHARACTER	3	DIROPRO	VSPC OBJECT PROG OFFSET =2048+WSMFREEA-WSM
55	(37)	CHARACTER	3	DIRLDSIZ	SIZE OF GETMAIN FOR)LOAD
58	(3A)	SIGNED	2		RESERVED
60	(3C)	UNSIGNED	4	DIRFRBA	FIRST ALLOCATED RBA

CROSS REFERENCE

DIR	0	(0)
DIRCATTR	44	(2C)
DIRCFLSZ	48	(30)
DIRCICS	18	X'08'
DIRCODE	7	(7)
DIRDF	45	X'01'
DIRFEBID	15	(F)
DIRFLAG1	19	(13)
DIRFONLY	52	(34)
DIRFRBA	60	(3C)
DIRFREE	18	X'80'
DIRFTYPE	45	(2D)
DIRHEADR	0	(0)
DIRHICNT	17	(11)
DIRKEY	4	(4)
DIRLCIDL	40	(28)
DIRLCIWR	60	(3C)
DIRLDSIZ	55	(37)
DIRLENHW	0	(0)
DIRLIBNO	4	(4)
DIRLOCK	19	X'01'
DIRLRS	56	(38)
DIRNCI	36	(24)
DIRNLR	52	(34)
DIROPRO	52	(34)
DIRPASW	19	X'02'
DIRPRIV	18	X'01'
DIRPSWD	20	(14)
DIRPUB	18	X'04'
DIRSCFLG	19	X'10'
DIRSF	45	X'02'
DIRSHR	19	X'20'
DIRSWTS	28	(1C)
DIRTYPE	18	(12)
DIRUPROF	18	X'20'
DIRWS	45	X'08'
DIRWSNAM	7	(7)

DMP (CICS, XSYS, AP)

This is the common system executor services dump request block which describes areas of storage to be dumped. It is mapped by the APLXDMP macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	108	DMP	DUMP REQUEST BLOCK
0	(0)	SIGNED	32	DMPWRK	WORK AREA FOR DMP RTN
32	(20)	CHARACTER	4	DMPID	ID FOR DUMP
36	(24)	CHARACTER	64	DMPREG	REGISTERS TO DUMP
36	(24)	SIGNED	4	DMPR0	REG 0
40	(28)	SIGNED	4	DMPR1	REG 1
44	(2C)	SIGNED	4	DMPR2	REG 2
48	(30)	SIGNED	4	DMPR3	REG 3
52	(34)	SIGNED	4	DMPR4	REG 4
56	(38)	SIGNED	4	DMPR5	REG 5
60	(3C)	SIGNED	4	DMPR6	REG 6
64	(40)	SIGNED	4	DMPR7	REG 7
68	(44)	SIGNED	4	DMPR8	REG 8
72	(48)	SIGNED	4	DMPR9	REG 9
76	(4C)	SIGNED	4	DMPR10	REG 10
80	(50)	SIGNED	4	DMPR11	REG 11
84	(54)	SIGNED	4	DMPR12	REG 12
88	(58)	SIGNED	4	DMPR13	REG 13
92	(5C)	SIGNED	4	DMPR14	REG 14
96	(60)	SIGNED	4	DMPR15	REG 15
100	(64)	A-ADDRESS	8	DMPLHDR	USED BY DUMP SERVICES
108	(6C)	CHARACTER	0	DMPLIST	START OF DUMP LIST

CROSS REFERENCE

DMP	0	(0)
DMPID	32	(20)
DMPLHDR	100	(64)
DMPLIST	108	(6C)
DMPREG	36	(24)
DMPR0	36	(24)
DMPR1	40	(28)
DMPR10	76	(4C)
DMPR11	80	(50)
DMPR12	84	(54)
DMPR13	88	(58)
DMPR14	92	(5C)
DMPR15	96	(60)
DMPR2	44	(2C)
DMPR3	48	(30)
DMPR4	52	(34)
DMPR5	56	(38)
DMPR6	60	(3C)
DMPR7	64	(40)
DMPR8	68	(44)
DMPR9	72	(48)
DMPWRK	0	(0)

DRB (TSO, XSYS)

This is the request block for DAIR services used by the TSO executor. It is mapped by the APLYDRB macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	54	DRB	
0	(0)	CHARACTER	44	DRBMAIN	
0	(0)	SIGNED	4	DRBREQ	TYPE OF REQUEST, SEE BELOW
4	(4)	CHARACTER	8	DRBDDNAM	DD NAME WHEN REQUIRED BY REQUEST
12	(C)	A-ADDRESS	4	DRB@DSN	ADDRESS OF DSN WHEN REQUIRED BY REQUEST. THE DSN AT THE ADDRESS IS MAPPED BY DRBDSN, DESCRIBED BELOW
16	(10)	CHARACTER	8	DRBSER	FOR "ALLOCATE NEW" REQUESTS, A SERIAL NUMBER PADDED WITH BLANKS, OR BLANKS. FOR "ALLOC OLD" OR "ALLOC SHR" THE SERIAL WHERE THE DS WAS FOUND
24	(18)	CHARACTER	8	DRBUNIT	FOR "ALLOC NEW" REQUESTS A UNIT TYPE PADDED WITH BLANKS, OR BLANKS. FOR "ALLOC OLD" OR "ALLOC SHR" THE UNIT TYPE WHERE THE DS WAS FOUND
32	(20)	SIGNED	4	DRBRC	RETURN CODE AFTER REQUEST IS COMPLETE. SEE BELOW
36	(24)	SIGNED	4	DRBRS	WHEN DRBRC=DRBDAIRC, THE RETURN CODE FROM DAIR
40	(28)	SIGNED	2	DRBDARC	WHEN DRBRC=DRBDAIRC, THE DARC FROM DAIR
42	(2A)	SIGNED	2	DRBCTRC	WHEN DRBRC=DRBDAIRC, THE CTRC FROM DAIR
44	(2C)	CHARACTER	10	DRBAPRMS	
44	(2C)	SIGNED	4	DRBPRMRY	NUMBER OF UNITS FOR PRIMARY ALLOCATION IN NEW DATASETS
48	(30)	SIGNED	4	DRBSCNDY	NUMBER OF UNITS FOR SECONDARY ALLOCATION IN NEW DATASETS
52	(34)	SIGNED	2	DRBBLKSZ	AVG. BLKSIZE FOR NEW DATASETS

CROSS REFERENCE

DRB	0	(0)
DRB@DSN	12	(C)
DRBAPRMS	44	(2C)
DRBBLKSZ	52	(34)
DRBCTRC	42	(2A)
DRBDARC	40	(28)
DRBDDNAM	4	(4)
DRBMAIN	0	(0)
DRBPRMRY	44	(2C)
DRBRC	32	(20)
DRBREQ	0	(0)
DRBRS	36	(24)
DRBSCNDY	48	(30)
DRBSER	16	(10)
DRBUNIT	24	(18)

ECA (VSPC)

This is the VS APL executor work area for VSPC. This control block is mapped by the APLPECA macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	FLOATING	8	ECA	
WORK AREAS					
0	(0)	FLOATING	8	ECAWORK1	WORKAREA FOR ARITHMETIC
8	(8)	FLOATING	8	ECAWORK2	ANOTHER WORKAREA
16	(10)	CHARACTER	320	ECAWKBF	WORK BUFFER TO ZCODE-EBCDIC TRAN
				ECABFLN	"320" LENGTH OF WORK BUFFER
		...1		ECASCV	"ECAWKBF" EXECUTOR SCV-BUILD AREA
		..1. .1..		ECASCVL	"SCVFLAG2+L'SCVFLAG2-SCVENTRY LTH OF SCV
SAVE AREA					
		.1.. 1...		ECASVLN	"18*4" LENGTH OF EACH ENTRY
336	(150)	SIGNED	4	ECASAVE	5 SAVE AREAS
				ECASVEND	"*"
696	(2B8)	SIGNED	4	ECASVPTR	RELATIVE PTR TO CURRENT SV
ECA MISCELLANEOUS FIELDS					
700	(2BC)	SIGNED	4	ECAWSPTR	PTR TO CURRENT WORKSPACE
704	(2C0)	SIGNED	4	ECAPTC	PTC PTR
708	(2C4)	SIGNED	4	ECADUMP	DUMP NUMBER
712	(2C8)	CHARACTER	11	ECAWSNAM	ACTIVE WSNAME
723	(2D3)	CHARACTER	1		MUST FOLLOW ECAWSNM
724	(2D4)	SIGNED	4	ECALIBNO	ACTIV LIBNO
728	(2D8)	SIGNED	2	ECAMM	MINUTES
730	(2DA)	SIGNED	2	ECAHH	HOURS
732	(2DC)	SIGNED	2	ECASS	SECONDS
WORKSPACE EQUATES					
				ECAWKLN	"2048" LENGTH OF CONTROL AREAS
PCO REQUEST CODE SAVED WHEN UNEXPECTED ERROR RETURN					
734	(2DE)	SIGNED	2	ECARQER	PCO REQUEST CODE ON ERROR
GENERAL RETURN CODE DEFINITIONS FOR PCO					
		.... .1..		ECAWARN	"4" WARNING
		.... 1...		ECABORT	"8" ABORTED,UNUSUAL CONDITION
		.... 11..		ECAFAIL	"12" NOT DONE,INVALID SITUATION
		...1		ECAREJ	"16" REQUEST REJECTED
EXECUTOR CONTROL BIT					
736	(2E0)	HEX	1	ECACNTRL	EXECUTOR CONTROL BITS

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
DEFINITION OF ECACNTRL					
		1...		ECAMICRO	"BIT0" MICRO CODE AVAILABLE
		.1..		ECASTAT	"BIT1" 0=SUPERVISOR,1=INTERPRETER
737	(2E1)	HEX	1		RESERVED
738	(2E2)	SIGNED	2	ECAPATHN	GDDX ACTIVE PATH COUNTER AP126
VSPC PSEUDO-AP FILE TABLE ENTRIES (APFT)					
740	(2E4)	SIGNED	4	APFTENT	
740	(2E4)	HEX	1		
				APFTEND	"*"
1520	(5F0)	FLOATING	8	ECACNTME	TERMINAL CONNECT TIME FOR INVOC
1528	(5F8)	FLOATING	8	ECAKEYTM	USER KEYING TIME FOR INVOCATION
FOLLOWING FIELDS ARE FOR DISPLAY TERMINAL I/O					
1536	(600)	SIGNED	4	ECADBFLN	BUFFER LENGTH (DISPLAY)
1540	(604)	SIGNED	4	ECADDTPT	HOLD RELATIVE DATA PTR
THE NEXT TWO FIELDS ARE FOR THE AP 101 PSEUDO-INPUT STACK					
1544	(608)	SIGNED	2	ECASTCKL	LENGTH OF DATA IN AP 101 STACK
1546	(60A)	SIGNED	2	ECAFENCL	LENGTH OF STACK DATA PAST FENCE
FOLLOWING FIELDS ARE FOR DISPLAY TERMINAL I/O					
1548	(60C)	SIGNED	2	ECADDATL	LENGTH OF DATA HELD
1550	(60E)	SIGNED	2	ECADCURS	HOLD REAL CURSR POSITION
1552	(610)	HEX	1	ECADFLAG	FLAGS
		1...		ECACFSAP	"X'80'" CURRENT FULL SCREEN IS AP 124
		.1..		ECAAPSSA	"X'40'" AP 124 FULL SCREEN SSA EXISTS
		..1.		ECACFSXE	"X'20'" CURRENT FULL SCREEN IS XEDIT
		.... .1..		ECASPURG	"X'04'" AP 101 STACK MUST BE PURGED
		.... ..1.		ECADFLNL	"X'02'" NL NOT EXPECTED IF ON
		1		ECADCANC	"X'01'" O U T CONDITION
1553	(611)	HEX	1	ECAFSFLG	FULL SCREEN EDITOR FLAGS 37123
		1...		FSEDINIT	"X'80'" FS EDITOR INITIALIZED
		.1..		FSMSGFUL	"X'40'" FS MSG AREA IS FULL
		..1.		FSMBFLSH	"X'20'" FS MSG AREA HAS BEEN FLUSHED
		...1		FSMSCSTK	"X'10'" SCREEN 'STACK' IN USE
		.... 1...		FSEDOPEN	"X'08'" FS EDITOR CURRENTLY OPEN
		.... .1..		FSLPROT	"X'04'" FS ED LINE NUMBERS PROTECTED
		1		FSMALARM	"X'01'" FS ED MESSAGE REQUIRES ALARM
1554	(612)	SIGNED	2	ECADBSCT	COUNT OF BACKSPACES HELD
1556	(614)	SIGNED	4	ECAEND	
				ECALEN	"*-ECA" LENGTH OF ECA

NOTE: 132 BYTES JUST BEFORE THE START OF THE WSM HAS BEEN BROUGHT INTO USE AS A 3270 BUFFER. THEREFORE THERE ARE ONLY A FEW EXPANSION BYTES BEYOND THE ECA SINCE THE SPACE FROM THE WSH TO THE WSM MUST REMAIN AT 2K TOTAL

CROSS REFERENCE

APFTEND	1520	ECAPATHN	738(2E2)
APFTENT	740(2E4)	ECAPTC	704(2C0)
ECA	0 (0)	ECAREJ	734 X'10'
ECAAPSSA	1552 X'40'	ECARQER	734(2DE)
ECABFLN	320	ECASAVE	336(150)
ECABORT	734 X'08'	ECASCV	16 X'10'
ECACFSAP	1552 X'80'	ECASCVL	16 X'24'
ECACFSXE	1552 X'20'	ECASPURG	1552 X'04'
ECACNTME	1520(5F0)	ECASS	732(2DC)
ECACNTRL	736(2E0)	ECASTAT	736 X'40'
ECADBFLN	1536(600)	ECASTCKL	1544(608)
ECADBSCT	1554(612)	ECASVEND	696
ECADCANC	1552 X'01'	ECASVLN	16 X'48'
ECADCURS	1550(60E)	ECASVPTR	696(2B8)
ECADDATL	1548(60C)	ECAWARN	734 X'04'
ECADDTPT	1540(604)	ECAWKBF	16 (10)
ECADFLAG	1552(610)	ECAWKLN	2048
ECADFLNL	1552 X'02'	ECAWORK1	0 (0)
ECADUMP	708(2C4)	ECAWORK2	8 (8)
ECAEND	1556(614)	ECAWSNAM	712(2C8)
ECAFAIL	734 X'0C'	ECAWSPTR	700(2BC)
ECAFENCL	1546(60A)	FSEDINIT	1553 X'80'
ECAFSFLG	1553(611)	FSEDOPEN	1553 X'08'
ECAHH	730(2DA)	FSLPROT	1553 X'04'
ECAKEYTM	1528(5F8)	FSMALARM	1553 X'01'
ECALEN	1556	FSMBFLSH	1553 X'20'
ECALIBNO	724(2D4)	FSMSCSTK	1553 X'10'
ECAMICRO	736 X'80'	FSMSGFUL	1553 X'40'
ECAMM	728(2D8)		

FAB (CICS, XSYS, AP)

This is the APL file access block. It maintains the current processing options and position of APL files that are open. It is used to pass requests and records between the library manager and either auxiliary processor 121 or the scrolling routines of the screen format manager. All auxiliary processor FABs are chained from the GBL. FABs associated with scrolling are also pointed to by the PTK. This control block is mapped by the APLKFAB macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	144	FAB	
0	(0)	CHARACTER	24	FABGRE	GRE FOR GLOBAL SERVICES
0	(0)	CHARACTER	8	FABNPSWD	NEW PASSWORD FOR CHANGE PWD
0	(0)	BITSTRING	4		ECB POSTED AT COMPLETION
4	(4)	A-ADDRESS	4		NEXT GRE ON CHAIN OR 0
8	(8)	UNSIGNED	1		SERVICE REQUEST CODE FIELD
9	(9)	UNSIGNED	1		TYPE QUALIFIER
10	(A)	BITSTRING	2	FABGRERC	GLOBAL SERVICE RETURN CODE
10	(A)	BITSTRING	2	FABVSERR	NOT SUPPORTED ERROR MSGHERE
12	(C)	SIGNED	4	FABGREPM	GRE INPUT PARAMETER
16	(10)	SIGNED	4	FABLPRM1	FIRST LIBRARY SERVICES PARM
20	(14)	A-ADDRESS	4	FABLPRM2	SECOND LIBRARY SERVICES PRM
20	(14)	SIGNED	2	FABLRCOD	ALSO USED FOR RETURN CODE
24	(18)	A-ADDRESS	4	FABOPCHN	OPEN FILE CHAIN-ANCHORED IN GLOBAL TABLE-0 AT CHAIN END
28	(1C)	A-ADDRESS	4	FABCHAIN	FAB CHAIN POINTER FOR USER >SCRSG FOR SCROLL FILES
32	(20)	A-ADDRESS	4	FABFEBPT	FEB POINTER IN BUFFER
36	(24)	SIGNED	4	FABCURCI	CURRENT CI NUMBER (LOGICAL)
40	(28)	SIGNED	2	FABREQST	FAB REQUEST CODES
42	(2A)	BITSTRING	2	FABSTAT	FILE STATUS INDICATORS
42	(2A)	BITSTRING	1	FABSTAT1	FIRST STATUS BYTE
		1...		FABNEW	NEW FILE, NOT YET OPEN 80
		.1..		FABFIRD	FIRST READ OPEN FLAG 40
		..1.		FABCLOSE	FILE IS CLOSED 20
		...1		FABOPEN	FILE IS OPEN 10
		.... 1...		FABOPRD	READ IS ACTIVE ON FILE 08
		.... .1..		FABOPWR	WRITE IS ACTIVE ON FILE 04
		.... ..1.		FABOPSQ	FILE IS BEING PROCESSED 02 SEQUENTIALLY
		1		FABOPDR	FILE IS BEING PROCESSED 01 DIRECTLY
43	(2B)	BITSTRING	1	FABSTAT2	SECOND STATUS BYTE
		1...		FABWRLST	LAST OPERAT WAS A WRITE 80
		.1..		FABCIUP	CI HAS NEW DATA 40
		..1.		FABDIRUP	DIRECTORY REC-NEW DATA 20
		...1		FABCHNED	FAB CURRENTLY CHAINED 10
		.... 1...		FABEOF	LAST REQUEST CAUSED EOF 08
		.... .1..		FABOPNW	FILE OPENED ORIG FOR WR 04
		.... ..1.		FABNXEOF	RET EOF ON NEXT REQUEST 02
		1			RESERVED

FAB (CICS, XSYS, AP) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
44	(2C)	SIGNED	4	FABCRREC	CURRENT RECORD NUMBER, POSITION IN THE FILE
48	(30)	A-ADDRESS	4	FABCIPTR	CI BUFFER ADDRESS
52	(34)	A-ADDRESS	4	FABDATAP	ACTIVE DATA POINTER
56	(38)	A-ADDRESS	4	FABBFPTR	CURRENT LOCATION IN BUFFER >DUMMY CITW FOR SCROLL BELOW IS THE DIRECTORY REC
60	(3C)	CHARACTER	84	FABDIR	BEGINNING OF DIR ENTRY
60	(3C)	CHARACTER	4	FABHEADR	HEADER TO DIRECTORY ENTRY
60	(3C)	SIGNED	2	FABLENHW	DIRECTORY ENTRY LENGTH
62	(3E)	SIGNED	2		RESERVED
64	(40)	CHARACTER	14	FABKEY	14 BYTE VSAM KEY
64	(40)	UNSIGNED	3	FABLIBNO	DDNAME FOR AP121 FILE
67	(43)	CHARACTER	11	FABWSNAM	WORKSPACE NAME
67	(43)	CHARACTER	8	FABFNAME	OR FILE NAME
75	(4B)	CHARACTER	3	FABRSVDN	RESERVED
78	(4E)	CHARACTER	1	FABTYPE	TYPE BYTE
		1...		FABFREE	FREE SPACE RECORD 80
		.1..			RESERVED
		..1.		FABUPROF	USER PROFILE BIT 20
		...1			RESERVED
		.... 1...		FABCICS	CICS DIRECTORY ENTRY 08
		.... .1..		FABPUB	PUBLIC LIBRARY 04
		.... ..1.			RESERVED
		1		FABPRIV	PRIVATE LIBRARY 01
79	(4F)	CHARACTER	1	FABFLAG1	FLAG BYTE 1
		11..			RESERVED
		..1.		FABSHR	FILE CAN BE SHARED 20
		...1		FABSCFLG	ACTIVE SCROLL FILE FLAG 10
		.... 11..			RESERVED
		.... ..1.		FABPASW	WS OR FILE HAS PASSWORD 02
		1		FABLOCK	USER IS LOCKED 01
80	(50)	CHARACTER	8	FABPSWD	WS OR FILE PASSWORD
88	(58)	CHARACTER	12	FABTFLD1	MAP FOR TSO RECO USAGE
88	(58)	BITSTRING	8	FABSWTS	SAVE WRITE APL STD TIME
96	(60)	SIGNED	4	FABNCI	NUMBER OF ALLOCATED CIS
96	(60)	SIGNED	4	FABMXSZ	MAX SIZE FOR TSO USAGE
100	(64)	UNSIGNED	4	FABLCIDL	DATA LENGTH IN LAST CI
104	(68)	CHARACTER	1	FABCATTR	CONTENT ATTRIBUTE
105	(69)	CHARACTER	15	FABTFLD2	TRY TO KEEP PLS HAPPY
105	(69)	BITSTRING	1	FABFTYPE	TYPE OF ENTRY
		1111			RESERVED
		.... 1...		FABWS	THIS IS A WORKSPACE 08
		.... .1..			RESERVED
		.... ..1.		FABSF	THIS IS A SEQ FILE 02
		1		FABDF	THIS IS A DIRECT FILE 01
106	(6A)	SIGNED	2	FABMAXSZ	MAX FILE SIZE IN CONTROL INTERVAL INCR (6K-LEN(CIT))

FAB (CICS, XSYS, AP) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
108	(6C)	SIGNED	4	FABCFLSZ	CURRENT FILE SIZE
112	(70)	SIGNED	4	FABNLR	NUMBER OF LOGICAL RECORDS
116	(74)	SIGNED	4	FABLRS	LOGICAL RECORD SIZE-DIRECT (= FABCIDAT FOR SCROLL) MAX RECORD SIZE-SEQUENTIAL
120	(78)	BITSTRING	8	FABTCDT	DATE FILE CREATED
120	(78)	SIGNED	4	FABLCIWR	LAST CI WRITTEN INTO (REL CI NUMBER , ORIGIN 1), IS END OF RECORD AND FILE
124	(7C)	SIGNED	4		KEEP PLACE FOR TSO TSO DEFINES FOR
128	(80)	SIGNED	4	FABBUFSZ	BUFFER SIZE
132	(84)	A-ADDRESS	4	FABAFBI	ADDRESS OF ABF FOR FILE
136	(88)	A-ADDRESS	4	FABRDBUF	SEQUENTIAL READ BUFFER PTR
140	(8C)	A-ADDRESS	4	FABRECOC	COUNTER WHEN TO WRITE REC 0

CROSS REFERENCE

FAB	0	(0)	FABMAXSZ	106	(6A)
FABAFBI	132	(84)	FABMXSZ	96	(60)
FABBFPTR	56	(38)	FABNCI	96	(60)
FABBUFSZ	128	(80)	FABNEW	42	X'80'
FABCATTR	104	(68)	FABNLR	112	(70)
FABCFLSZ	108	(6C)	FABNPSWD	0	(0)
FABCHAIN	28	(1C)	FABNXEOF	43	X'02'
FABCHNED	43	X'10'	FABOPCHN	24	(18)
FABCICS	78	X'08'	FABOPDR	42	X'01'
FABCIPTR	48	(30)	FABOPEN	42	X'10'
FABCIUP	43	X'40'	FABOPNW	43	X'04'
FABCLOSE	42	X'20'	FABOPRD	42	X'08'
FABCRREC	44	(2C)	FABOPSQ	42	X'02'
FABCURCI	36	(24)	FABOPWR	42	X'04'
FABDATAP	52	(34)	FABPASW	79	X'02'
FABDF	105	X'01'	FABPRIV	78	X'01'
FABDIR	60	(3C)	FABPSWD	80	(50)
FABDIRUP	43	X'20'	FABPUB	78	X'04'
FABEOF	43	X'08'	FABRDBUF	136	(88)
FABFEBPT	32	(20)	FABRECOC	140	(8C)
FABFIRD	42	X'40'	FABREQST	40	(28)
FABFLAG1	79	(4F)	FABRSVDN	75	(4B)
FABFNAME	67	(43)	FABSCFLG	79	X'10'
FABFREE	78	X'80'	FABSF	105	X'02'
FABFTYPE	105	(69)	FABSHR	79	X'20'
FABGRE	0	(0)	FABSTAT	42	(2A)
FABGREPM	12	(C)	FABSTAT1	42	(2A)
FABGRERC	10	(A)	FABSTAT2	43	(2B)
FABHEADR	60	(3C)	FABSWTS	88	(58)
FABKEY	64	(40)	FABTCDT	120	(78)
FABLCIDL	100	(64)	FABTFLD1	88	(58)
FABLCIWR	120	(78)	FABTFLD2	105	(69)
FABLENHW	60	(3C)	FABTYPE	78	(4E)
FABLIBNO	64	(40)	FABUPROF	78	X'20'
FABLOCK	79	X'01'	FABVSERR	10	(A)
FABLPRM1	16	(10)	FABWRLST	43	X'80'
FABLPRM2	20	(14)	FABWS	105	X'08'
FABLRCOD	20	(14)	FABWSNAM	67	(43)
FABLRS	116	(74)			

FB (CONV, NTRP)

This is the interpreter definition of the function close parameter list. It is mapped by the APLFBLST macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	0	FBLIST	0
0	(0)	HEX	4	FBSRCE	SEMI-REL ADDR OF THIS SOURCE LINE
4	(4)	HEX	4	FBSRCEL	COUNT OF BYTES IN SOURCE LINE (WHEN ITCLOSET CALLED, SIZE OF LONGEST)
8	(8)	HEX	4	FBUILD	SEMI-REL BUILD-AREA POINTER
12	(C)	HEX	4	FBUILDL	BYTE LENGTH OF BUILD AREA REMAINING
16	(10)	HEX	2	FBLABELS	COUNT OF LABELS IN THIS FUNCTION
18	(12)	HEX	2	FBLBLOFF	OFFSET TO (LABELS) IN FN. HEADER
20	(14)	HEX	4	FBDNWORD	SEMI-REL ADDRESS OF FN. DN-WORD
24	(18)	HEX	0	FBLABLO	LABEL HEADER ENTRY
24	(18)	HEX	2	FBTHISLB	THIS LINE'S LABEL'S NAME
26	(1A)	HEX	2	FBTHISLN	THIS LINE'S NUMBER
28	(1C)	HEX	2	FBFUNAME	NAME OF THIS FUNCTION OBJECT
30	(1E)	HEX	1	FBFLAG	USED BY ITLINEO TO SIGNAL COUNT OF NAMES FOUND. 0=FUNCTION-NAME ONLY.
31	(1F)	HEX	4	FBSYNT	USED BY ITLINEO TO RETURN SYNTAX CLASS; SET TO SBITFUN0, SBITFUN1, OR SBITFUN2
		..1.		FBLISTL	"*-FBLIST" LENGTH OF LIST FOR CLEARING

CROSS REFERENCE

FBDNWORD	20	(14)
FBFLAG	30	(1E)
FBFUNAME	28	(1C)
FBLABELS	16	(10)
FBLABLO	24	(18)
FBLBLOFF	18	(12)
FBLIST	0	(0)
FBLISTL	31	X'20'
FBSRCE	0	(0)
FBSRCEL	4	(4)
FBSYNT	31	(1F)
FBTHISLB	24	(18)
FBTHISLN	26	(1A)
FBUILD	8	(8)
FBUILDL	12	(C)

FEB (CICS, SERV)

This is the file extent block used by the CICS/VS executor which describes the library extents for AP 121 and scroll files. It is mapped by the APLKFEB macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	152	FEB	
0	(0)	CHARACTER	4	FEBHEADR	HEADER TO DIRECTORY ENTRY
0	(0)	SIGNED	2	FEBLENHW	DIRECTORY ENTRY LENGTH
2	(2)	SIGNED	2		RESERVED
4	(4)	CHARACTER	14	FEBKEY	14 BYTE VSAM KEY
4	(4)	UNSIGNED	3	FEBLIBNO	USER LIBRARY NUMBER
7	(7)	CHARACTER	11	FEBWSNAM	FILE NAME 11 CHARACTERS
7	(7)	CHARACTER	8	FEBFNAME	TRUE FILE NAME 8 CHARS
15	(F)	CHARACTER	3	FEBIDENT	FEB IDENTIFIER HEX FFXXXX
15	(F)	UNSIGNED	1	FEBCODE	UNIQUE FEB CODE HEX FF
16	(10)	SIGNED	2	FEBSEQ	SEQUENTIAL FEB NUMBER
18	(12)	CHARACTER	1	FEBTYPE	TYPE BYTE
		1...		FEBFREE	FREE SPACE RECORD
		.1..		FEBFEB	THIS IS A FEB
		..1.		FEBUPROF	USER PROFILE BIT
		...1			RESERVED
		.... 1...		FEBCICS	CICS DIRECTORY ENTRY
		.... .1..		FEBPUB	PUBLIC LIBRARY
		.... ..1.			RESERVED
		1		FEBPRIV	PRIVATE LIBRARY
19	(13)	CHARACTER	1	FEBFLAG1	FLAG BYTE 1
		11..			RESERVED
		..1.		FEBSHR	FILE CAN BE SHARED
		...1 11..			RESERVED
		.... ..1.		FEBPASW	WS OR FILE HAS PASSWORD
		1		FEBLOCK	USER IS LOCKED
20	(14)	SIGNED	4	FEBNUMEX	NUMBER OF 8 BYTE EXTENTS
24	(18)	CHARACTER	128	FEBEXTNT	16 ALLOCATION EXTENTS
24	(18)	SIGNED	4	FEBNUMCI	NUMBER OF CONTIGUOUS CIS
28	(1C)	UNSIGNED	4	FEBFCRBA	FIRST RBA IN THE ALLOCATION

CROSS REFERENCE

FEB	0	(0)
FEBCICS	18	X'08'
FEBCODE	15	(F)
FEBEXTNT	24	(18)
FEBFCRBA	28	(1C)
FEBFEB	18	X'40'
FEBFLAG1	19	(13)
FEBFNAME	7	(7)
FEBFREE	18	X'80'
FEBHEADR	0	(0)
FEBIDENT	15	(F)
FEBKEY	4	(4)
FEBLENHW	0	(0)
FEBLIBNO	4	(4)
FEBLOCK	19	X'01'
FEBNUMCI	24	(18)
FEBNUMEX	20	(14)
FEBPASW	19	X'02'
FEBPRIV	18	X'01'
FEBPUB	18	X'04'
FEBSEQ	16	(10)
FEBSHR	19	X'20'
FEBTYPE	18	(12)
FEBUPROF	18	X'20'
FEBWSNAM	7	(7)

FFLD (VSPC)

This is the display screen field information table entry for the FSM internal auxiliary processor for VSPC. Each display screen field defined by the user to the FSM auxiliary processor is described in an FFLD entry in the FSMFLD table. The FSM work area contains the variable size FSMFLD table. This control block is mapped by the APLPFSM macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	0	FSMWORK	
0	(0)	SIGNED	2	FSMSCRNL	LTH OF DATA IN SCREEN BUFFER
2	(2)	SIGNED	2		RESERVED
4	(4)	CHARACTER	3836	FSMSCRN	FSM AP SCREEN BUFFER
3840	(F00)	SIGNED	2	FMAXROW	FSM SCREEN MAX ROW
3842	(F02)	SIGNED	2	FMAXCOL	FSM SCREEN MAX COLUMN
3844	(F04)	SIGNED	4	FSMCURSR	CURSOR SETTING FOR WRITE COMMAND
3844	(F04)	SIGNED	2	FSMCROW	ROW
3846	(F06)	SIGNED	2	FSMCCOL	COL
3848	(F08)	SIGNED	4	FSMPARMD	DISPLMT TO FSMPARM
3852	(F0C)	SIGNED	4	FSMPARMN	# ELEMENTS IN FSMPARM
3856	(F10)	SIGNED	4	FSMPARMR	# REMAINING FSMPARM SLOTS
3860	(F14)	SIGNED	4	FSMVFN	# FLD NUMBER INTEGERS IN VF-SAVE
3864	(F18)	A-ADDRESS	4	FSMWFLD(10)	FSM SUBROUTINE WORK AREA
3904	(F40)	A-ADDRESS	4	FSMSSAVE	BASE REGISTER SAVE FOR LEVEL-E SUBROUTINES:FSMSUB1,FSMSUB3
				FWORKL	"*-FSMWORK" LTH FSMWORK HEADER
3908	(F44)	SIGNED	4	FSMFLD	FORMAT FIELD DESCRIPTION TABLE (SEE FFLD DSECT)
3908	(F44)	SIGNED	4	FSMPARM	PARAMETER LIST FOR VSPC TFSCRN SERVICE REQUESTS. ALSO USED AS V-SAVE AND VF-SAVE AREA

FFLD DSECT
DESCRIBES FSMFLD ENTRY CONTAINED IN FSMWORK.FSMFLD LIST

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	0	FFLD	
0	(0)	SIGNED	4	FLOC	FLD POSITION (UPPER LEFT CORNER)
0	(0)	SIGNED	2	FROW	ROW
2	(2)	SIGNED	2	FCOL	COLUMN
4	(4)	SIGNED	4	FSIZE	FLD SIZE
4	(4)	SIGNED	2	FWID	FLD WIDTH (NUMBER OF COLUMNS)
6	(6)	SIGNED	2	FHT	FLD HEIGHT (NUMBER OF ROWS)
8	(8)	SIGNED	4	FFLDNO	FLD IDENTIFICATION NUMBER. FOR FLD#0, FFLDNO=TOTAL NUMBER OF FFLD ENTRIES IN FSMFLD.

FFLD (VSPC) continued

```
 OFFSETS     TYPE      LENGTH  NAME               DESCRIPTION

-------------------------------------------------------------------------------
  12    (C) SIGNED        2   FMROW              DAT ROW NUMBER ASSOCIATED WITH FLD
  14    (E) HEX           7   FSHORTR            SHORT-ROW BIT FLAGS 0=ROW
                                                 LTH=FLD WIDTH 1=ROW LTH=FLD
                                                 WIDTH-1
  21   (15) HEX           1   FATTR              FIELD ATTRIBUTE FLAGS
===============================================================================
FATTR DEFINITIONS
            .... ...1         FDISPN             "X'01'" DISPLAY INTENSITY NORMAL
            .... ..1.         FDISPH             "X'02'" DISPLAY INTENSITY
                                                 HIGHLIGHT
            .... .1..         FDISPC             "X'04'" DISPLAY INTENSITY
                                                 CONFIDENTIAL
            .... 1...         FAPPEN             "X'08'" LIGHT PEN SENSITIVE
            ...1 ....         FNUM               "X'10'" NUMERIC INPUT
            ..1. ....         FPROHT             "X'20'". PROTECTED
            .1.. ....         FAPPENA            "X'40'" LIGHT PEN ATTENTION
            1... ....         FFSMPEN            "X'80'" LIGHT PEN SENSITIVE
===============================================================================
IF FFLAG.FFPENDA=PENDING ATTRIBUTE-CHANGE, THEN
FATTR.FAPPEN INDICATES USER REQUEST TO CHANGE FIELD
TO OR FROM PEN-SENSITIVE, AND FATTR.FFSMPEN INDICATES
CURRENT PEN ATTRIBUTE STATE OF FIELD.
  22   (16) HEX           1   FFLAG              FLAG
===============================================================================
FFLAG DEFINITIONS
            .... ...1         FFUNDF             "X'01'" UNDEFINED FLD
            .... ..1.         FFPENDA            "X'02'" PENDING ATTRIBUTE CHANGE
            .... .1..         FFSHORTR           "X'04'" FLD CONTAINS SHORT ROW(S)
            .... 1...         FFMAXWID           "X'08'" FLD WIDTH=SCREEN MAXIMUM
            ...1 ....         FFVF               "X'10'" FLD NUMBER NAMED IN CTL VF
                                                 VECTOR FOR WRITE COMMAND
  23   (17) HEX           1   FBADGEL            LTH(BADGE DATA)
-------------------------------------------------------------------------------
  24   (18) SIGNED        4   FBADGE             BADGE DATA POSITION
-------------------------------------------------------------------------------
  24   (18) SIGNED        2   FBROW              ROW
  26   (1A) SIGNED        2   FBCOL              COLUMN
            ...1 11..         FSMFLDL            "*-FFLD"
===============================================================================
TFSCRN-READ FIELD SPECIFICATION DSECT
-------------------------------------------------------------------------------
   0    (0) STRUCTURE     0   TRFPARM
-------------------------------------------------------------------------------
   0    (0) SIGNED        4   TRFOPT             FLD SPECIFICATION FLAGS
===============================================================================
TRFOPT VALUES
                              TRFMDT             "X'200'" MODIFIED DATA (MDT)
-------------------------------------------------------------------------------
   4    (4) SIGNED        4   TRFROW             FLD POSITION ROW(UPPER LEFT
                                                 CORNER)
-------------------------------------------------------------------------------
   8    (8) SIGNED        4   TRFCOL             FLD POSITION COLUMN
-------------------------------------------------------------------------------
  12    (C) SIGNED        4   TRFWID             FLD WIDTH
-------------------------------------------------------------------------------
  16   (10) SIGNED        4   TRFDAT             DATA POSITION
-------------------------------------------------------------------------------
  20   (14) SIGNED        4   TRFDATL            FLD AREA=TOTAL DATA LENGTH
            ...1 1...         TRFPARML           "*-TRFPARM"
```

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
TFSCRN-READ REQUEST HEADER DSECT					
0	(0)	STRUCTURE	0	TRRPARM	
0	(0)	SIGNED	4		RESERVED
4	(4)	SIGNED	4	TRRCURSR	CURSOR SETTING AFTER READ
4	(4)	SIGNED	4	TRRCROW	ROW
8	(8)	SIGNED	4	TRRCCOL	COLUMN
12	(C)	SIGNED	4	TRROPT	REQUEST OPTION FLAGS
TRROPT VALUES					
		1		TRRGET	"X'01'" GET DATA FROM VSPC WORKAREA
		.... ..1.		TRRREAD	"X'02'" READ DATA FROM SCREEN
		.... 11..		TRRCODE	"TRROPT" COMPLETION CODE
TRRCODE VALUES					
		1		TRRPROG	"1" PROGRAM REQUEST
		.... ..1.		TRRENT	"2" ENTER KEY
		11		TRRPEN	"3" SELECTOR PEN ATTENTION
		.... .1..		TRRCLEAR	"4" CLEAR KEY
		.... .1.1		TRRBADGE	"5" BADGE
				TRRPF1	"1001" PF-1 KEY
				TRRPF2	"1002" PF-2 KEY
				TRRPF3	"1003" PF-3 KEY
				TRRPF4	"1004" PF-4 KEY
				TRRPF5	"1005" PF-5 KEY
				TRRPF6	"1006" PF-6 KEY
				TRRPF7	"1007" PF-7 KEY
				TRRPF8	"1008" PF-8 KEY
				TRRPF9	"1009" PF-9 KEY
				TRRPF10	"1010" PF-10 KEY
				TRRPR11	"1011" PF-11 KEY
				TRRPR12	"1012" PF-12 KEY
				TRRPA1	"2001" PA-1 KEY
				TRRPA2	"2002" PA-2 KEY
				TRRPA3	"2003" PA-3 KEY
16	(10)	SIGNED	4	TRRMROW	DISPLAY SIZE (MAX ROW)
20	(14)	SIGNED	4	TRRMCOL	DISPLAY SIZE (MAX COLUMN)
		...1 1...		TRRPARML	"*-TRRPARM"
TFSCRN-WRITE FIELD SPECIFICATION DSECT					
0	(0)	STRUCTURE	0	TWFPARM	
0	(0)	SIGNED	4	TWFOPT	FLD SPECIFICATION FLAG
TWFOPT DEFINITIONS					
		1		TWFDATA	"X'01'" FILL FLD WITH DATA
		.... ..1.		TWFERASE	"X'02'" ERASE FLD
		.... .1..		TWFATTR	"X'04'" SET FLD CHARACTERISTICS
		.... 1...		TWFDISPN	"X'08'" DISPLAY INTENSITY NORMAL
		...1		TWFDISPH	"X'10'" DISPLAY INTENSITY HIGHLIGHT
		..1.		TWFDISPC	"X'20'" DISPLAY INTENSITY CONFIDENTIAL
		.1..		TWFPEN	"X'40'" SELECTOR PEN SENSITIVE
		1...		TWFNUM	"X'80'" NUMERIC INPUT FLD
				TWFPROHT	"X'100'" PROTECTED FLD

FFLD (VSPC) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
4	(4)	SIGNED	4	TWFROW	FLD POSITION ROW(UPPER LEFT CORNER)
8	(8)	SIGNED	4	TWFCOL	FLD POSITION COLUMN
12	(C)	SIGNED	4	TWFWID	FLD WIDTH
16	(10)	SIGNED	4	TWFDAT	DATA POSITION INDEX(ORIGIN 1)
20	(14)	SIGNED	4	TWFDATL	FLD AREA=TOTAL DATA LENGTH
		...1 1...		TWFPARML	"*-TWFPARM"
TFSCRN-WRITE REQUEST HEADER DSECT					
0	(0)	STRUCTURE	0	TWRPARM	
0	(0)	SIGNED	4		RESERVED
4	(4)	SIGNED	4	TWRCURSR	CURSOR SETTING AFTER WRITE
4	(4)	SIGNED	4	TWRCROW	ROW
8	(8)	SIGNED	4	TWRCCOL	COLUMN
12	(C)	SIGNED	4	TWROPT	REQUEST OPTION FLAGS
TWROPT DEFINITIONS					
		.... .1..		TWRMDTR	"X'04'" RESET MDT
		.... 1...		TWRBUZZ	"X'08'" SOUND ALARM
16	(10)	SIGNED	4		RESERVED
20	(14)	SIGNED	4		RESERVED
		...1 1...		TWRPARML	"*-TWRPARM"
TFSCRN-PAGE REQUEST HEADER DSECT					
0	(0)	STRUCTURE	0	TPRPARM	
0	(0)	SIGNED	4		RESERVED
4	(4)	SIGNED	4		RESERVED
8	(8)	SIGNED	4		RESERVED
12	(C)	SIGNED	4	TPROPT	REQUEST OPTION
TPROPT DEFINITIONS					
		1		TPRHCOPY	"X'01'" MAKE HARDCOPY OF SCREEN
16	(10)	SIGNED	4		RESERVED
20	(14)	SIGNED	4		RESERVED
		...1 1...		TPRPARML	"*-TPRPARM"

CROSS REFERENCE

FAPPEN	21 X'08'
FAPPENA	21 X'40'
FATTR	21 (15)
FBADGE	24 (18)
FBADGEL	23 (17)
FBCOL	26 (1A)
FBROW	24 (18)
FCOL	2 (2)
FDISPC	21 X'04'
FDISPH	21 X'02'
FDISPN	21 X'01'
FFLAG	22 (16)
FFLD	0 (0)
FFLDNO	8 (8)
FFMAXWID	22 X'08'
FFPENDA	22 X'02'
FFSHORTR	22 X'04'
FFSMPEN	21 X'80'
FFUNDF	22 X'01'
FFVF	22 X'10'
FHT	6 (6)
FLOC	0 (0)
FMAXCOL	3842(F02)
FMAXROW	3840(F00)
FMROW	12 (C)
FNUM	21 X'10'
FPROHT	21 X'20'
FROW	0 (0)
FSHORTR	14 (E)
FSIZE	4 (4)
FSMCCOL	3846(F06)
FSMCROW	3844(F04)
FSMCURSR	3844(F04)
FSMFLD	3908(F44)
FSMFLDL	26 X'1C'
FSMPARM	3908(F44)
FSMPARMD	3848(F08)
FSMPARMN	3852(F0C)
FSMPARMR	3856(F10)
FSMSCRN	4 (4)
FSMSCRNL	0 (0)
FSMSSAVE	3904(F40)
FSMVFN	3860(F14)
FSMWFLD	3864(F18)
FSMWORK	0 (0)
FWID	4 (4)
FWORKL	3908
TPRHCOPY	12 X'01'
TPROPT	12 (C)
TPRPARM	0 (0)
TPRPARML	20 X'18'
TRFCOL	8 (8)
TRFDAT	16 (10)
TRFDATL	20 (14)
TRFMDT	512
TRFOPT	0 (0)
TRFPARM	0 (0)
TRFPARML	20 X'18'
TRFROW	4 (4)
TRFWID	12 (C)
TRRBADGE	12 X'05'
TRRCCOL	8 (8)
TRRCLEAR	12 X'04'
TRRCODE	12 X'0C'
TRRCROW	4 (4)
TRRCURSR	4 (4)
TRRENT	12 X'02'
TRRGET	12 X'01'
TRRMCOL	20 (14)
TRRMROW	16 (10)
TRROPT	12 (C)
TRRPARM	0 (0)
TRRPARML	20 X'18'
TRRPA1	2001
TRRPA2	2002
TRRPA3	2003
TRRPEN	12 X'03'
TRRPF1	1001
TRRPF10	1010
TRRPF2	1002
TRRPF3	1003
TRRPF4	1004
TRRPF5	1005
TRRPF6	1006
TRRPF7	1007
TRRPF8	1008
TRRPF9	1009
TRRPROG	12 X'01'
TRRPR11	1011
TRRPR12	1012
TRRREAD	12 X'02'
TWFATTR	0 X'04'
TWFCOL	8 (8)
TWFDAT	16 (10)
TWFDATA	0 X'01'
TWFDATL	20 (14)
TWFDISPC	0 X'20'
TWFDISPH	0 X'10'
TWFDISPN	0 X'08'
TWFERASE	0 X'02'
TWFNUM	0 X'80'
TWFOPT	0 (0)
TWFPARM	0 (0)
TWFPARML	20 X'18'
TWFPEN	0 X'40'
TWFPROHT	256
TWFROW	4 (4)
TWFWID	12 (C)
TWRBUZZ	12 X'08'
TWRCCOL	8 (8)
TWRCROW	4 (4)
TWRCURSR	4 (4)
TWRMDTR	12 X'04'
TWROPT	12 (C)
TWRPARM	0 (0)
TWRPARML	20 X'18'

FHED (CONV, NTRP)

This is used by the interpreter and defines the fixed fields of VS APL function headers. It is mapped by the APLFHED macro. This control block is mapped by the APLFHED macro. The fixed fields of the header of a defined function are shown below.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	0	FHED	
0	(0)	SIGNED	4	FHEDDN	DESCRIPTOR, BACK POINTER
4	(4)	SIGNED	2	FHEDM	LINE COUNT
6	(6)	SIGNED	2	FHEDT	OFFSET, FHEDDN TO TAIL
8	(8)	SIGNED	2	FHEDS	TRANSLATOR FLAGS
		.... 1..1		FHEDBITS	"FHEDS+1" FLAG BITS FOR TRANSLATOR
		...1		FHEDLOCK	"X'10'". LOCKED FUNCTION
		..11		FHEDMAG	"X'30'". MAGIC FUNCTION
		1		FHEDQUAD	"X'01'". QUAD-INPUT TEMPORARY
		.... ..1.		FHEDEXEC	"X'02'". EXECUTE TEMPORARY
		.... .1..		FHEDTSOK	"X'04'" TIMESTAMP PRESENT BEFORE TAIL
10	(A)	SIGNED	2	FHEDK	40+8×NUMBER OF LOCALS
A LABEL COUNTS AS A LOCAL HERE					
12	(C)	SIGNED	2	FHEDZ	NAME OF RESULT; 0001 IF NONE
14	(E)	SIGNED	2	FHEDL	NAME OF L. ARG; 0001 IF NONE
16	(10)	SIGNED	2	FHEDR	NAME OF R. ARG; 0001 IF NONE
		...1 ..1.		FHEDLOCL	"*". START OF LOCAL NAMES

CROSS REFERENCE

FHED	0	(0)
FHEDBITS	8	X'09'
FHEDDN	0	(0)
FHEDEXEC	8	X'02'
FHEDK	10	(A)
FHEDL	14	(E)
FHEDLOCK	8	X'10'
FHEDLOCL	16	X'12'
FHEDM	4	(4)
FHEDMAG	8	X'30'
FHEDQUAD	8	X'01'
FHEDR	16	(10)
FHEDS	8	(8)
FHEDT	6	(6)
FHEDTSOK	8	X'04'
FHEDZ	12	(C)

FSP (CICS, SERV)

This is the free space descriptor used by the CICS/VS executor, and describes the VS APL library data set with the number of control intervals allocated, etc. It is mapped by the APLKFSP macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	30	FSP	
0	(0)	CHARACTER	4	FSPHEADR	HEADER TO DIRECTORY ENTRY
0	(0)	SIGNED	2	FSPLENHW	DIRECTORY ENTRY LENGTH
2	(2)	SIGNED	2		RESERVED
4	(4)	CHARACTER	14	FSPKEY	FILE LOCATER KEY
4	(4)	CHARACTER	3	FSPLIBNO	LIBRARY NUMBER '000000'X
7	(7)	BITSTRING	1	FSPNAME	'01'X FOR FREE SPACE RECORD
8	(8)	CHARACTER	10		ALL ZEROS
18	(12)	CHARACTER	1	FSPFTYPE	FILE TYPE:
		1...		FSPFREE	FREE SPACE RECORD
		.1..		FSPFEB	FILE EXTENT RECORD
		..1.		FSPUPROF	USER PROFILE BIT
		...1			RESERVED
		.... 1...		FSPCICS	CICS DIRECTORY ENTRY
		.... .1..		FSPPUB	PUBLIC LIBRARY
		.... ..1.			RESERVED
		1		FSPPRIV	PRIVATE LIBRARY
19	(13)	CHARACTER	1	FSPFLAG1	FLAG BYTE 1
		1...		FSPINVLD	LIBRARY IS INVALID FLAG
		.1..		FSPENQED	LIBRARY ENQUEUE FLAG
20	(14)	SIGNED	4	FSPLIBCI	TOTAL LIBRARY CI COUNT
24	(18)	SIGNED	2	FSPFSCI	LIBRARY FREE SPACE CI COUNT
26	(1A)	SIGNED	2	FSPLCIBY	NO. OF BYTES IN LAST FREE SPACE CONTROL INTERVAL
28	(1C)	SIGNED	2	FSPLBYTB	NO. OF BITS IN LAST FREE SPACE BYTE

CROSS REFERENCE

FSP	0	(0)
FSPCICS	18	X'08'
FSPENQED	19	X'40'
FSPFEB	18	X'40'
FSPFLAG1	19	(13)
FSPFREE	18	X'80'
FSPFSCI	24	(18)
FSPFTYPE	18	(12)
FSPHEADR	0	(0)
FSPINVLD	19	X'80'
FSPKEY	4	(4)
FSPLBYTB	28	(1C)
FSPLCIBY	26	(1A)
FSPLENHW	0	(0)
FSPLIBCI	20	(14)
FSPLIBNO	4	(4)
FSPNAME	7	(7)
FSPPRIV	18	X'01'
FSPPUB	18	X'04'
FSPUPROF	18	X'20'

GBL (CICS, XSYS, AP)

This is the CICS/VS APL global table. It is the primary anchor for all CICS/VS APL control blocks. The GBL is located at the beginning of module APLKASTB and is pointed to by the PTK and the user task TWA (the CICS/VS transaction work area). This control block is mapped by the APLKGBL macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	168	GBL	
0	(0)	A-ADDRESS	4	GBLSSM	EP TO SSM SERVICE REQ
4	(4)	CHARACTER	20		COMMON SERVICES
4	(4)	A-ADDRESS	4	GBLINIT	ROUTINE SETS UP GBL TABL
8	(8)	A-ADDRESS	4	GBLHIST	EP TO HISTOGRAM RECORDER
12	(C)	A-ADDRESS	4	GBLADEF	AUTHORIZATION CHECKS
16	(10)	A-ADDRESS	4	GBLDEST	EP TO DESTINATION MGR
20	(14)	A-ADDRESS	4		RESERVED
24	(18)	CHARACTER	28		PTRS TO OTHER TABLES
24	(18)	A-ADDRESS	4	GBLVCT	VCT: X-SYSTEM ROUTINES
28	(1C)	A-ADDRESS	4	GBLOPSYS	OPS: OPSYS DEPENDENT
32	(20)	A-ADDRESS	4	GBLTRQ	NON-GDDM TERM VECT TABLE
36	(24)	A-ADDRESS	4	GBLPRM	PRM: INSTALLATION PARMS
40	(28)	A-ADDRESS	4	GBLHISTD	HISTOGRAMS (ADMIN WS)
44	(2C)	A-ADDRESS	4	GBLTRAN	TRANSLATE TABLES(APLKTCD)
48	(30)	A-ADDRESS	4	GBLCSA	ADDR OF CICS CSA
52	(34)	A-ADDRESS	4	GBLAICBA	>DFHAICBA IN DFHEAI
56	(38)	CHARACTER	2		FLAG BYTES
56	(38)	CHARACTER	1	GBLFLAG1	MISC FLAGS
		1...			RESERVED
		.1..		GBLGUP	GLOBAL TABLE IS INIT-D
		..1.		GBLINTRP	INTERPRETER IN CONTROL
		...1		GBLNMICR	UCODE ASSIST UNAVAILABLE
		.... 1...		GBLMICRO	MICROCODE ASSIST AVAILABLE
		.... .1..		GBLPFO	PAGE FAULT BEING HANDLED
57	(39)	BITSTRING	1	GBLFLAG2	RESERVED
58	(3A)	CHARACTER	14		SIGNON CONTROL
58	(3A)	SIGNED	2	GBLCSGN	NR OF USERS CURRENTLY ON
60	(3C)	A-ADDRESS	4	GBLSGN	ADDR OF APL SIGNON TABLE
64	(40)	A-ADDRESS	4	GBLLSGN	LAST ACTIVE ENTRY IN SGN
68	(44)	A-ADDRESS	4		RESERVED
72	(48)	CHARACTER	16		DISPATCHER SERVICES
72	(48)	A-ADDRESS	4	GBLBSERV	APLKWAIT/KEXIT SERVICES
76	(4C)	A-ADDRESS	4	GBLDSERV	ADSP SERVICE ENTRY POINT T

GBL (CICS, XSYS, AP) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
80	(50)	A-ADDRESS	4	GBLBEXIT	ABND EXIT ROUTINE INTRFC
84	(54)	A-ADDRESS	4	GBLBSTRT	NEW PROCESS START INTRFC
88	(58)	CHARACTER	20		LIB SERVICES FIELDS
88	(58)	A-ADDRESS	4	GBLRDIR	EP TO READ-DIRECTORY RTN
92	(5C)	A-ADDRESS	4	GBLLIBF	EP TO LIB FILE SERVICES
96	(60)	A-ADDRESS	4	GBLFRSPC	ADDR OF FREESPACE MAP
100	(64)	A-ADDRESS	4	GBLFSDES	TO FREESPACE DESCRIPTOR
104	(68)	A-ADDRESS	4	GBLFILEC	HEAD OF FAB CHAIN
108	(6C)	CHARACTER	12		LIBRARY TASK FIELDS
108	(6C)	A-ADDRESS	4	GBLLIBS	ROUTINE STARTS LIB TASKS
112	(70)	BITSTRING	4	GBLNQECB	STRING ENQUEUE ECB
		1...			OS/VS WAIT BIT
		.1..		GBLNQECO	OS/VS POST BIT
		..11 1111			
113	(71)	1111 1111			UNUSED
114	(72)	1...		GBLNQECD	DOS/VS POST BIT
116	(74)	SIGNED	2	GBLMXSTR	TOT # OF APLDIR STRINGS
118	(76)	SIGNED	2	GBLCSTR	# APLDIR STRINGS IN USE
120	(78)	CHARACTER	40		INTERPRETER INTERFACE
120	(78)	CHARACTER	8	GBLINTRA	INTERPRETER ADDRESSES
120	(78)	A-ADDRESS	4	GBLINTL	LOW END OF INTERPRETER
124	(7C)	A-ADDRESS	4	GBLINTH	HIGH END OF INTERPRETER
128	(80)	A-ADDRESS	4	GBLADSPL	LOW END OF APLKADSP
132	(84)	A-ADDRESS	4	GBLIRESM	EP TO INTERPRETER RESUME
136	(88)	A-ADDRESS	4	GBLQUEND	QUANTUM END ENTRY POINT
140	(8C)	A-ADDRESS	4	GBLIRET	R14 ON ENTRY TO KIFIX
144	(90)	BITSTRING	8	GBLOLDIT	OLD ITTAB ENTRY
152	(98)	SIGNED	4	GBLTTIME	TIMER RESIDUE
156	(9C)	A-ADDRESS	4		RESERVED
160	(A0)	CHARACTER	8		DOS/VS ONLY FIELDS
160	(A0)	A-ADDRESS	4	GBLPFC	PAGE FLT CONTROL AREA
164	(A4)	A-ADDRESS	4	GBLPFAP	EP TO PG FLT APPENDAGE
168	(A8)	CHARACTER	0		END OF GLOBAL TABLE

CROSS REFERENCE

GBL	0	(0)
GBLADEF	12	(C)
GBLADSPL	128	(80)
GBLAICBA	52	(34)
GBLBEXIT	80	(50)
GBLBSERV	72	(48)
GBLBSTRT	84	(54)
GBLCSA	48	(30)
GBLCSGN	58	(3A)
GBLCSTR	118	(76)
GBLDEST	16	(10)
GBLDSERV	76	(4C)
GBLFILEC	104	(68)
GBLFLAG1	56	(38)
GBLFLAG2	57	(39)
GBLFRSPC	96	(60)
GBLFSDES	100	(64)
GBLGUP	56	X'40'
GBLHIST	8	(8)
GBLHISTD	40	(28)
GBLINIT	4	(4)
GBLINTH	124	(7C)
GBLINTL	120	(78)
GBLINTRA	120	(78)
GBLINTRP	56	X'20'
GBLIRESM	132	(84)
GBLIRET	140	(8C)
GBLLIBF	92	(5C)
GBLLIBS	108	(6C)
GBLLSGN	64	(40)
GBLMICRO	56	X'08'
GBLMXSTR	116	(74)
GBLNMICR	56	X'10'
GBLNQECB	112	(70)
GBLNQECD	114	X'80'
GBLNQECO	112	X'40'
GBLOLDIT	144	(90)
GBLOPSYS	28	(1C)
GBLPFAP	164	(A4)
GBLPFC	160	(A0)
GBLPFO	56	X'04'
GBLPRM	36	(24)
GBLQUEND	136	(88)
GBLRDIR	88	(58)
GBLSGN	60	(3C)
GBLSSM	0	(0)
GBLTRAN	44	(2C)
GBLTRQ	32	(20)
GBLTTIME	152	(98)
GBLVCT	24	(18)

This is the VSPC AP 126 and GDDM interface common area format. It is mapped by the APLPGDC macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	480	GDCOMMON	
0	(0)	A-ADDRESS	4	GDCAPFTO	OFFSET OF APFT
4	(4)	A-ADDRESS	4	GDCPATCU	CURRENT PATH BLOCK OFFSET
8	(8)	A-ADDRESS	28	GDCPATPT	ARRAY OF PATH POINTERS
36	(24)	BITSTRING	1	GDCFLAGS	FLAG BYTE
		1...		ERRSTOP	TERMINATING ERROR STOP
		.1..		GETFLOAT	RETURN NEXT ITEM AS FLOAT
		..1.		PERFORM	FOR ASQFLD, DO REQUEST
37	(25)	UNSIGNED	1	GDCOTYPE	OUTPUT CTL VAR DATA TYPE
38	(26)	SIGNED	2	GDCREACD	AP 126 ERROR: REASON CODE
40	(28)	A-ADDRESS	4	GDCICTLO	OFFSET OF INPUT CTL VAR
44	(2C)	SIGNED	4	GDCINELM	NUMBER OF INPUT CTL ELEM
48	(30)	SIGNED	4	GDCINCNT	INPUT CTL ELEMENT COUNTER
52	(34)	A-ADDRESS	4	GDCIDATO	OFFSET OF INPUT DAT ELEM
56	(38)	SIGNED	4	GDCIDATN	ACTUAL LENGTH OF INPUT DAT
60	(3C)	SIGNED	4	GDCDINLN	EXPECTED LENGTH OF INP DAT
64	(40)	SIGNED	4	GDCCONVI	CONVERT RTN INTEGER OUTPUT
64	(40)	BITSTRING	4	GDCCONVF	CONVERT RTN OUTPUT (FLOAT)
68	(44)	SIGNED	4	GDCREQCD	GDDM FUNCTION REQUEST CODE
72	(48)	A-ADDRESS	4	GDCENTPT	ADDRESS OF TABLE ENTRY
76	(4C)	A-ADDRESS	4	GDCOCTLO	OFFSET OF CTL OUTPUT BUFF
80	(50)	SIGNED	4	GDCSZCTL	LENGTH OF CTL OUTPUT BUFF
84	(54)	A-ADDRESS	4	GDCOUTPT	OFFSET OF CTL OUTPUT ELEM
88	(58)	A-ADDRESS	4	GDCRCVPT	OFFSET OF CTL RETURN CODE
92	(5C)	SIGNED	4	GDCOUTCT	OUTPUT CTL ELEMENT COUNTER
96	(60)	A-ADDRESS	4	GDCODATO	OFFSET OF DAT OUTPUT BUFF
100	(64)	SIGNED	4	GDCSZDAT	LENGTH OF DAT OUTPUT BUFF
104	(68)	A-ADDRESS	4	GDCDOUTP	OFFSET OF DAT OUTPUT ELEM
108	(6C)	A-ADDRESS	4	GDCQSAVE	R14 SAVE AREA FOR QFLD RTN
112	(70)	A-ADDRESS	4	GDCNSAVE	R14 SAVE AREA FOR NEXT RTN
116	(74)	SIGNED	4		SPARE
120	(78)	SIGNED	16	GDCTWORK	TEMPORARY VAR WORK ARRAY
136	(88)	SIGNED	4	GDCVWORK	TEMPORARY VAR WORK WORD

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
140	(8C)	BITSTRING	1	GDCWBYTE	WORK BYTE FOR BIT MASKING
141	(8D)	UNSIGNED	1		SPARE
142	(8E)	SIGNED	2		SPARE
144	(90)	SIGNED	4		SPARE
148	(94)	SIGNED	4	GDCXNMAX	MAXIMUM NUMERIC PARAMETERS
152	(98)	A-ADDRESS	4	GDCXNPBO	NUMERIC PARM BUFFER OFFSET
156	(9C)	A-ADDRESS	4	GDCXNNPO	NEXT NUMERIC PARM OFFSET
160	(A0)	SIGNED	32	GDCXPCNT	ARRAY OF COUNTS FOR PARMS
192	(C0)	BITSTRING	1	GDCXFLGS	GDDX FLAG BYTE
		1...		GDCXHOPN	HARDCOPY DESTINATION OPEN
193	(C1)	BITSTRING	1	GDCXROPT	GDDX OPTION FLAGS
		1...			RESERVED
		.1..		GDCXPASS	THIS IS A PASSTHROUGH REQ
		..1.			RESERVED
		...1		GDCXNOPG	NO PAGE SELECTION REQUIRED
		 1111			RESERVED
194	(C2)	SIGNED	2	GDCXRNUM	NUMBER OF GDDM PARMS
196	(C4)	A-ADDRESS	4	GDCXPSNP	PAGE SEL REQ NAME OFFSET
200	(C8)	A-ADDRESS	4	GDCXPSPP	PAGE SEL PAGE NO. OFFSET
204	(CC)	A-ADDRESS	4	GDCXREQP	GDDM REQUEST NAME OFFSET
208	(D0)	A-ADDRESS	32	GDCXRPTR	GDDM PARM OFFSETS
240	(F0)	CHARACTER	8	GDCXREQN	GDDM REQUEST NAME
248	(F8)	CHARACTER	8	GDCXPSRN	PAGE SEL REQUEST NAME
256	(100)	SIGNED	4	GDCXPGCU	CURRENT GDDM PAGE NUMBER
260	(104)	SIGNED	4	GDCXHOPC	HARDCOPY OPTION COUNT
264	(108)	SIGNED	32	GDCXHOPT	FSOPEN HARDCOPY OPTIONS
296	(128)	CHARACTER	8	GDCXHARD	HARDCOPY DESTINATION NAME
304	(130)	CHARACTER	160	GDCXNOER	DUMMY NO-ERROR FEEDBACK
464	(1D0)	SIGNED	2	GDCXRC	GDDM RETURN CODE
466	(1D2)	SIGNED	2	GDCXRS	GDDM REASON CODE
468	(1D4)	A-ADDRESS	4	GDCWHERE	OFFSET OF AREA REQUESTED
472	(1D8)	SIGNED	4	GDCSPACE	LENGTH TO GET OR FREE
476	(1DC)	A-ADDRESS	4	GDCFIRST	OFFSET OF FIRST FREE AREA
480	(1E0)	CHARACTER	0	GDCFREE	FREE SPACE BEGINS

CROSS REFERENCE

ERRSTOP	36 X'80'	GDCXPSRN	248 (F8)
GDCAPFTO	0 (0)	GDCXRC	464(1D0)
GDCCONVF	64 (40)	GDCXREQN	240 (F0)
GDCCONVI	64 (40)	GDCXREQP	204 (CC)
GDCDINLN	60 (3C)	GDCXRNUM	194 (C2)
GDCDOUTP	104 (68)	GDCXROPT	193 (C1)
GDCENTPT	72 (48)	GDCXRPTR	208 (D0)
GDCFIRST	476(1DC)	GDCXRS	466(1D2)
GDCFLAGS	36 (24)	GETFLOAT	36 X'40'
GDCFREE	480(1E0)	PERFORM	36 X'20'
GDCICTLO	40 (28)		
GDCIDATN	56 (38)		
GDCIDATO	52 (34)		
GDCINCNT	48 (30)		
GDCINELM	44 (2C)		
GDCNSAVE	112 (70)		
GDCOCTLO	76 (4C)		
GDCODATO	96 (60)		
GDCOMMON	0 (0)		
GDCOTYPE	37 (25)		
GDCOUTCT	92 (5C)		
GDCOUTPT	84 (54)		
GDCPATCU	4 (4)		
GDCPATPT	8 (8)		
GDCQSAVE	108 (6C)		
GDCRCVPT	88 (58)		
GDCREACD	38 (26)		
GDCREQCD	68 (44)		
GDCSPACE	472(1D8)		
GDCSZCTL	80 (50)		
GDCSZDAT	100 (64)		
GDCTWORK	120 (78)		
GDCVWORK	136 (88)		
GDCWBYTE	140 (8C)		
GDCWHERE	468(1D4)		
GDCXFLGS	192 (C0)		
GDCXHARD	296(128)		
GDCXHOPC	260(104)		
GDCXHOPN	192 X'80'		
GDCXHOPT	264(108)		
GDCXNMAX	148 (94)		
GDCXNNPO	156 (9C)		
GDCXNOER	304(130)		
GDCXNOPG	193 X'10'		
GDCXNPBO	152 (98)		
GDCXPASS	193 X'40'		
GDCXPCNT	160 (A0)		
GDCXPGCU	256(100)		
GDCXPSNP	196 (C4)		
GDCXPSPP	200 (C8)		

GDM (XSYS, AP)

This is the common system services GDDM request block interface for VS APL. It is mapped by the APLXGDM macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	64	GDM	USED WITH APLXG MACRO
0	(0)	A-ADDRESS	4	GDMRLINK	GDDM LINK PTR
4	(4)	A-ADDRESS	4	GDMRECBP	ECB ADDRESS FOR POST
8	(8)	BITSTRING	4	GDMREQCD	GDDM OR APL TYPE CODE
12	(C)	A-ADDRESS	4	GDMRPATH	PATH IDENTIFICATION PTR
16	(10)	SIGNED	4	GDMRPAGE	PAGE NUMBER
20	(14)	BITSTRING	1	GDMROPT	OPTION FLAGS
		1...		GDMRWAIT	ON FSFRCE,WAIT FOR COMPL
		.1..		GDMRPASS	THIS IS A PASSTHROUGH REQ
		..1.		GDMRAPL	GDMREQCD IS APL UNIQUE
		...1		GDMRNOPG	NO PAGE SELECTION REQD
		.... 1...		GDMRDOWN	ON GDMRTRM, BRING DOWN GDDX
		.... .111		GDMRORS	RESERVED
21	(15)	BITSTRING	1		RESERVED
22	(16)	SIGNED	2	GDMRNUM	NUMBER OF FIELDS
24	(18)	SIGNED	4	GDMRC	RETURN CODE
28	(1C)	SIGNED	4	GDMRS	REASON CODE
32	(20)	A-ADDRESS	4	GDMRPTR1	POINTER TO FIRST PARM
36	(24)	A-ADDRESS	4	GDMRPTR2	POINTER TO SECOND PARM
40	(28)	A-ADDRESS	4	GDMRPTR3	POINTER TO THIRD PARM
44	(2C)	A-ADDRESS	4	GDMRPTR4	POINTER TO FOURTH PARM
48	(30)	A-ADDRESS	4	GDMRPTR5	POINTER TO FIFTH PARM
52	(34)	A-ADDRESS	4	GDMRPTR6	POINTER TO SIXTH PARM
56	(38)	A-ADDRESS	4	GDMRPTR7	RESERVED
60	(3C)	A-ADDRESS	4	GDMRPTR8	RESERVED

CROSS REFERENCE

GDM	0	(0)
GDMRAPL	20	X'20'
GDMRC	24	(18)
GDMRDOWN	20	X'08'
GDMRECBP	4	(4)
GDMREQCD	8	(8)
GDMRLINK	0	(0)
GDMRNOPG	20	X'10'
GDMRNUM	22	(16)
GDMROPT	20	(14)
GDMRORS	20	X'07'
GDMRPAGE	16	(10)
GDMRPASS	20	X'40'
GDMRPATH	12	(C)
GDMRPTR1	32	(20)
GDMRPTR2	36	(24)
GDMRPTR3	40	(28)
GDMRPTR4	44	(2C)
GDMRPTR5	48	(30)
GDMRPTR6	52	(34)
GDMRPTR7	56	(38)
GDMRPTR8	60	(3C)
GDMRS	28	(1C)
GDMRWAIT	20	X'80'

LSC (CICS, SERV)

This is the library services constants for CICS/VS, and maps the CITs from the bit map CIs in the VS APL library. It is mapped by the APLKLSC macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	22	LSCAPLIB	CIT FOR BIT MAP CIS
0	(0)	SIGNED	4		LIBNO AND IDENT
4	(4)	CHARACTER	10		OUR ID
14	(E)	CHARACTER	1		FREE SPACE IDENTIFIER
15	(F)	BITSTRING	3		RDF
18	(12)	BITSTRING	4		4K CIDF

CROSS REFERENCE

LSCAPLIB 0 (0)

MAI (XSYS, AP)

This is the common system services main storage request block. It is mapped by the APLXMAI macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	28	MAI	
0	(0)	BITSTRING	2	MAIREQ	REQUEST CODE
		1...		MAIRQTYP	REQUEST TYPE, 0=GETMAIN,1=FREE
		.1..		MAIRVAR	GETMAIN FIXED=0, VARIABLE=1
		..1.		MAIINITF	=1, MAIINIT HAS INIT BYTE
		...1			RESERVED
		 1111			
1	(1)	1111 1111		MAISYS	SYSTEM REQUEST FLAGS
		.... 1...		MAIVPAGE	GETMAIN BNDRY PAGE=1, DWORD=0
		.... .1..		MAICHIGH	ALLOCATE FROM HIGH AREA
		.... ..1.		MAICLOW	ALLOCATE FROM LOW AREA
		1			RESERVED
1	(1)	1...		MAIKSYS	USE OPERATING SYSTEM SERVICES
		.1..		MAIKTERM	TERMINAL CLASS GETMAIN
		..1.		MAIKTRAN	TRANSIENT DATA CLASS GETMAIN
		...1		MAIKTEMP	TEMPORARY STORAGE CLASS GETMAIN
		.... 1...		MAIKPROG	PROGRAM CLASS GETMAIN
		.... .1..		MAIKSHR	SHARED TYPE GETMAIN
		11			reserved
2	(2)	SIGNED	2	MAIGMOPT	VARIOUS OPTION FIELDS
2	(2)	UNSIGNED	1	MAIVSUBP	SUBPOOL NUMBER FOR OS GETMAINS
3	(3)	UNSIGNED	1	MAIINIT	INITIALIZATION BYTE
4	(4)	A-ADDRESS	4	MAIADDR	ADDR OF GETMAIN OR FREEMAIN AREA
8	(8)	SIGNED	4	MAISIZE	GETMAIN/FREEMAIN SIZE, IN BYTES
8	(8)	SIGNED	4	MAIMAX	VARIABLE GETMAIN MAXIMUM VALUE
12	(C)	CHARACTER	16	MAIWORK	STORAGE SERVICES WORK AREA
12	(C)	SIGNED	4	MAIWORK1	WORK AREA WORD 1
16	(10)	SIGNED	4	MAIWORK2	WORK AREA WORD 2
20	(14)	SIGNED	4	MAIWORK3	WORK AREA WORD 3
24	(18)	SIGNED	4	MAIWORK4	WORK AREA WORD 4

CROSS REFERENCE

MAI	0	(0)
MAIADDR	4	(4)
MAICHIGH	0	X'04'
MAICLOW	0	X'02'
MAIGMOPT	2	(2)
MAIINIT	3	(3)
MAIINITF	0	X'20'
MAIKPROG	1	X'08'
MAIKSHR	1	X'04'
MAIKSYS	1	X'80'
MAIKTEMP	1	X'10'
MAIKTERM	1	X'40'
MAIKTRAN	1	X'20'
MAIMAX	8	(8)
MAIREQ	0	(0)
MAIRQTYP	0	X'80'
MAIRVAR	0	X'40'
MAISIZE	8	(8)
MAISYS =		12
MAIVPAGE	0	X'08'
MAIVSUBP	2	(2)
MAIWORK	12	(C)
MAIWORK1	12	(C)
MAIWORK2	16	(10)
MAIWORK3	20	(14)
MAIWORK4	24	(18)

OPS (CICS, AP)

This is the system-dependent services branch table in CICS/VS system-dependent modules. It is mapped by the APLKOPS macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	44	OPS	
0	(0)	CHARACTER	12		INTERPRETER INTERFACE
0	(0)	A-ADDRESS	4	OPSINIEX	INIT TIMER EXIT AND PAGE
4	(4)	A-ADDRESS	4	OPSSETEX	SET PROGRAM CHECK AND
8	(8)	A-ADDRESS	4	OPSRSTEX	RESTORE THE OLD EXITS
12	(C)	CHARACTER	16		LIBRARY SERVICES
12	(C)	A-ADDRESS	4	OPSLOPEN	VSAM OPEN
16	(10)	A-ADDRESS	4	OPSLCLOS	VSAM CLOSE
20	(14)	A-ADDRESS	4	OPSLGET	VSAM GET
24	(18)	A-ADDRESS	4	OPSLPUT	VSAM PUT
28	(1C)	CHARACTER	4		DESTINATION MGR
28	(1C)	A-ADDRESS	4	OPSDATTR	EXTRN FILE ATTRIBUTES
32	(20)	CHARACTER	4		DISPATCHER
32	(20)	A-ADDRESS	4	OPSPCREG	PROGRAM CHECK REGISTERS
36	(24)	CHARACTER	8		AP 123
36	(24)	A-ADDRESS	4	OPSER123	VSAM/ISAM ERROR ANALYSIS
40	(28)	A-ADDRESS	4	OPSDPFAP	DISCONNECT PAGE FAULT
44	(2C)	CHARACTER	0		END OF LIST

CROSS REFERENCE

OPS	0	(0)
OPSDATTR	28	(1C)
OPSDPFAP	40	(28)
OPSER123	36	(24)
OPSINIEX	0	(0)
OPSLCLOS	16	(10)
OPSLGET	20	(14)
OPSLOPEN	12	(C)
OPSLPUT	24	(18)
OPSPCREG	32	(20)
OPSRSTEX	8	(8)
OPSSETEX	4	(4)

PCV (ALL)

This is the processor control vector (PCV), which contains information about an auxiliary processor. It is used in communication between auxiliary processor and shared storage manager. This control block is mapped by the APLPCV macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	24	PCV	THIS IS THE BEGINNING OF THE PCV
0	(0)	SIGNED	4	PCVID	ID FOR THIS PROCESSOR
4	(4)	SIGNED	4	PCVQUOTA	QUOTA OF SHARED VARIABLES
8	(8)	SIGNED	4	PCVSPACE	BYTES OF SHARED MEMORY THIS
12	(C)	A-ADDRESS	4	PCVECB	ECB ADDR FOR THIS PROCESSOR
16	(10)	CHARACTER	8	PCVPASS	PASSWORD FOR THIS PROCESSOR
24	(18)	CHARACTER	0	PCVEND	END OF PLS PCV MAPPING
0	(0)	STRUCTURE	24	APLPCV	
0	(0)	CHARACTER	24	PCV	
0	(0)	SIGNED	4	PCVID	
4	(4)	SIGNED	4	PCVQUOTA	
8	(8)	SIGNED	4	PCVSPACE	
12	(C)	A-ADDRESS	4	PCVECB	
16	(10)	CHARACTER	8	PCVPASS	
24	(18)	CHARACTER	0	PCVEND	

CROSS REFERENCE

APLPCV	0	(0)
PCV	0	(0)
PCVECB	12	(C)
PCVEND	24	(18)
PCVID	0	(0)
PCVPASS	16	(10)

PRD [XSYS, AP]

This is the common system services print request block descriptor. It is mapped by the APLXPRD macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	20	PRD	PRINT REQUEST DESCRIPTOR
0	(0)	UNSIGNED	1	PRDTYPE	REQUEST TYPE AS FOR GDDM REQ
1	(1)	BITSTRING	1	PRDOPT	OPTIONS
		1...		PRDHDR	OPEN HEADER GENERATION
2	(2)	SIGNED	2	PRDLEN	LENGTH OF DATA IN BUFFER
4	(4)	SIGNED	2	PRDRETCD	RETURN CODE
6	(6)	SIGNED	2	PRDTCODE	TYPE CODE (TRANSLATION CONTROL)
8	(8)	A-ADDRESS	4	PRDBUFP	ADDR OF BUFFER
12	(C)	CHARACTER	8	PRDDEST	DESTINATION NAME
20	(14)	CHARACTER	0	PRDEND	END OF PRD

CROSS REFERENCE

PRD	0	(0)
PRDBUFP	8	(8)
PRDDEST	12	(C)
PRDEND	20	(14)
PRDHDR	1	X'80'
PRDLEN	2	(2)
PRDOPT	1	(1)
PRDRETCD	4	(4)
PRDTCODE	6	(6)
PRDTYPE	0	(0)

PRM (CICS, XSYS, AP)

This is the installation parameter list. It contains the installation-specified variables used in initializing the APL transaction. The PRM is located at the beginning of module APLKPARM and is pointed to by the GBL. This control block is mapped by the APLKPRM macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	172	PRM	INSTALLATION PARAMETERS
0	(0)	SIGNED	2	PRMUSERS	MAX SIGN-ON COUNT
2	(2)	SIGNED	2	PRMSSMSZ	MAX SHARED STORAGE SIZE IN K-BYTES
4	(4)	SIGNED	2	PRMSPMAX	MAXIMUM PROCESSORS SHARING INCLUDES USERS, DEP-APS, AND INDEPENDENT-APS
6	(6)	SIGNED	2	PRMSVMAX	MAXIMUM SHARED VARIABLES
8	(8)	SIGNED	2	PRMCHAP	LOWER PRIORITY FOR FULL USE OF ONE QUANTUM
10	(A)	SIGNED	2		RESERVED
12	(C)	SIGNED	4	PRMLIM	HARDCOPY PRINT LIMIT
16	(10)	CHARACTER	28	PRMTRANS	APL TRANSACTIONS
16	(10)	CHARACTER	4	PRMSON	APL SIGN-ON TRANSACTION
20	(14)	CHARACTER	4	PRMSES	APL USER SESSION TRAN
24	(18)	CHARACTER	4	PRMLIB	LIBRARY TRANSACTION
28	(1C)	CHARACTER	4	PRMTERM	APL TERMINAL I/O TRAN
32	(20)	CHARACTER	4	PRMHCP	APL HARDCOPY TRANSACTION
36	(24)	CHARACTER	4	PRMDYN	APL AP 100 DYNAMIC TRAN
40	(28)	CHARACTER	4	PRMTERX	GDDM TRMNL I/O TRAN
44	(2C)	CHARACTER	4	PRMXLEVL	RELEASE #: 00VVRRMM
48	(30)	CHARACTER	1	PRMSYS	SYSTEM TYPE, 1-VS1,2-MVS, 3-DOS/VS
49	(31)	CHARACTER	1	PRMCREL	CICS VERSION/RELEASE
50	(32)	SIGNED	2	PRMAPCNT	ENTRY COUNT IN AP TABLE
52	(34)	SIGNED	4	PRMSLICE	APL TIME SLICE SYS DEP
56	(38)	A-ADDRESS	4	PRMDAPPT	ADDRESS OF DEPENDENDT APS
60	(3C)	A-ADDRESS	4	PRMSGN	ADR OF SIGNON TABLE

CICS RELEASE DEPENDENT OFFSETS

NOTICE
THE FOLLOWING TABLE MUST BE USED TO DEFINE ALL FIELDS IN CICS CONTROL BLOCKS WHICH ARE USED BY APL CODE, EXCEPT FOR FIELDS DOCUMENTED IN THE CICS APPLICATION PROGRAMMER'S REFERENCE MANUAL (MACRO LEVEL) AS BEING PART OF THE APPLICATION PROGRAMMER INTERFACE (API).

```
DCL WRKP PTR                      * POINTS DIRECTLY TO FIELD
DCL FLD BASED(WRKP) ...           * GIVE APPROPRIATE ATTRIBUTES
WRKP= ...PTR + PRM...             * POINT TO REQUIRED FIELD
FLD=FLD | PRM ...                 * OR IN MASK TO TURN IT ON
FLD=FLD & (PRM ...&& 'FF'X)       * AND COMPLEM TO TURN IT OFF
DCL WRK BIT(8)                    * TEMPORARY WORK AREA
WRK= PRM... & FLD                 * REMOVE ALL OTHER BITS
```

OFFSETS	TYPE	LENGTH	NAME	DESCRIPTION
CSA.....				
64 (40)	SIGNED	2	PRMCSAPN	PINI C(2) NUMBER PGM CHKS(PACK)
66 (42)	SIGNED	2	PRMCSAPP	PIPSW P(31) PGMCHK PSW
68 (44)	SIGNED	2	PRMCSAPL	PLBA P(31) CICS LOW BOUNDARY
70 (46)	SIGNED	2	PRMCSAPU	PUBA P(31) CICS HIGH BOUNDARY
72 (48)	SIGNED	4		RESERVED
76 (4C)	SIGNED	2	PRMCSADA	DATFT C(1) FORMAT OF DATE
78 (4E)	CHARACTER	1	PRMCSAMD	DATFM FORMAT AS MMDDYY
79 (4F)	CHARACTER	1		RESERVED
80 (50)	SIGNED	2	PRMDCTBA	DCTBA P(31) DTF/DCB PTR
82 (52)	SIGNED	2	PRMDCTID	DCTIDI C(4) INDIRECT DESTID
84 (54)	SIGNED	2	PRMDCTDT	DCTTDT C(1) DESTINATION TYPE
86 (56)	CHARACTER	1	PRMDCINT	INDTBM INTRA- PARTITION
87 (57)	CHARACTER	1	PRMDCEXT	EXTRBM EXTRA- PARTITION
88 (58)	CHARACTER	1	PRMDCIND	INDBM INDIR DESTINATION
89 (59)	CHARACTER	1		RESERVED
90 (5A)	SIGNED	2		RESERVED
92 (5C)	SIGNED	2	PRMFIODC	DCB P(31) > DCB
94 (5E)	SIGNED	2		RESERVED
FCT....				
96 (60)	SIGNED	2	PRMFCID	DSID C(7) DLBL DDNAME
98 (62)	SIGNED	2	PRMFCKL	DSKL F(8)
100 (64)	SIGNED	2	PRMFCREC	DSREC F(15) RECORD LENGTH
102 (66)	SIGNED	2	PRMFCRKP	DSRKP F(15) REL KEY POSITION
104 (68)	SIGNED	2	PRMFCACB	DSACB C(0) START OF ACB
106 (6A)	SIGNED	2		RESERVED
108 (6C)	SIGNED	2	PRMFCOPN	DSOPN C(1) OPEN
110 (6E)	CHARACTER	1	PRMFCOP	OPNIM..OPEN BIT
111 (6F)	CHARACTER	1		RESERVED

PRM (CICS, XSYS, AP) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
112	(70)	SIGNED	2	PRMFCDBM	DSBDM C(1) BDAM INDICATOR
114	(72)	CHARACTER	1	PRMFCDAM	BDAMI BDAM BIT
115	(73)	CHARACTER	1		RESERVED
116	(74)	SIGNED	2	PRMFCISM	DSISM C(1) ISAM INDICATOR
118	(76)	CHARACTER	1	PRMFCISA	ISAMI ISAM BIT
119	(77)	CHARACTER	1		RESERVED
120	(78)	SIGNED	2	PRMFCVLI	DSVLI C(1) RECOR
122	(7A)	CHARACTER	1	PRMFCFIX	FIXIM FIXED LNG
123	(7B)	CHARACTER	1	PRMFCVRL	VRLIM VAR LNGTH
124	(7C)	SIGNED	2	PRMFCDLI	DSDLI C(1) DL/1
126	(7E)	CHARACTER	1	PRMFCDL1	DLII DL/I BIT
127	(7F)	CHARACTER	1		RESERVED
128	(80)	SIGNED	2	PRMFCVSM	DSVSM C(1) VSAM INDICATOR
130	(82)	CHARACTER	1	PRMFCVSA	VSAMI VSAM BIT
131	(83)	CHARACTER	1		RESERVED
132	(84)	SIGNED	2	PRMFCESD	DSESD C(1) VSAM INDICATOR
134	(86)	CHARACTER	1	PRMFCES	ESDS ENTRY SEQ
135	(87)	CHARACTER	1		RESERVED
136	(88)	SIGNED	2	PRMFCKSD	DSKSD C(1) VSAM KSDS INDICATOR
138	(8A)	CHARACTER	1	PRMFCKS	KSDS KEY SEQ
139	(8B)	CHARACTER	1		RESERVED
140	(8C)	SIGNED	2	PRMFCRRD	DSRRD C(1) VSAM RRDS INDICATOR
142	(8E)	CHARACTER	1	PRMFCRR	RRDS REL REC
143	(8F)	CHARACTER	1		RESERVED
144	(90)	SIGNED	2	PRMPCTOF	TWA F(15) TWA SIZE
146	(92)	SIGNED	2	PRMPCTPF	IPIA C(8) PROG NAM
148	(94)	SIGNED	2	PRMPCTTI	TI C(4) TRANS NAM
150	(96)	SIGNED	2		RESERVED
152	(98)	SIGNED	2	PRMPCTFF	FDPOP C(1) FORMAT DUMP
154	(9A)	CHARACTER	1	PRMFASRA	FASRA DUMP ON ASRA
155	(9B)	CHARACTER	1		RESERVED
156	(9C)	SIGNED	2	PRMPCTOG	FLAG C(1) TRANSACTION
158	(9E)	CHARACTER	1	PRMDSABL	DSABL DISABLED
159	(9F)	CHARACTER	1		RESERVED
160	(A0)	SIGNED	2	PRMTATLR	TPLRC F(8) TC LOCATE RC
162	(A2)	SIGNED	2	PRMTASYA	SYAA P(31) PREFIX POINTER
164	(A4)	SIGNED	2	PRMTATPC	TCPC P(31) PCT PTR
166	(A6)	SIGNED	2	PRMTATKA	KCTTA C(3) TASK # (PACK)
168	(A8)	SIGNED	2	PRMTCTTC	TETC C(4) NEXT TRANID
170	(AA)	SIGNED	2	PRMTCTOI	TERMINAL OP ID
172	(AC)	CHARACTER	0	PRMDAP	MULTIPLE DEP-AP'S DEFINED
172	(AC)	CHARACTER	8	PRMAPNAM	PROGRAM NAME FOR LOAD
180	(B4)	SIGNED	4	PRMAPNUM	DEPENDENT AP NUMBER
184	(B8)	A-ADDRESS	4	PRMAPPTR	ENTRY POINT OF AP-IF LOADED

CROSS REFERENCE

PRM	0	(0)
PRMAPCNT	50	(32)
PRMAPNAM	172	(AC)
PRMAPNUM	180	(B4)
PRMAPPTR	184	(B8)
PRMCHAP	8	(8)
PRMCREL	49	(31)
PRMCSADA	76	(4C)
PRMCSAMD	78	(4E)
PRMCSAPL	68	(44)
PRMCSAPN	64	(40)
PRMCSAPP	66	(42)
PRMCSAPU	70	(46)
PRMDAP	172	(AC)
PRMDAPPT	56	(38)
PRMDCEXT	87	(57)
PRMDCIND	88	(58)
PRMDCINT	86	(56)
PRMDCTBA	80	(50)
PRMDCTDT	84	(54)
PRMDCTID	82	(52)
PRMDSABL	158	(9E)
PRMDYN	36	(24)
PRMFASRA	154	(9A)
PRMFCACB	104	(68)
PRMFCDAM	114	(72)
PRMFCDBM	112	(70)
PRMFCDLI	124	(7C)
PRMFCDL1	126	(7E)
PRMFCES	134	(86)
PRMFCESD	132	(84)
PRMFCFIX	122	(7A)
PRMFCID	96	(60)
PRMFCISA	118	(76)
PRMFCISM	116	(74)
PRMFCKL	98	(62)
PRMFCKS	138	(8A)
PRMFCKSD	136	(88)
PRMFCOP	110	(6E)
PRMFCOPN	108	(6C)
PRMFCREC	100	(64)
PRMFCRKP	102	(66)
PRMFCRR	142	(8E)
PRMFCRRD	140	(8C)
PRMFCVLI	120	(78)
PRMFCVRL	123	(7B)
PRMFCVSA	130	(82)
PRMFCVSM	128	(80)
PRMFIODC	92	(5C)
PRMHCP	32	(20)
PRMLIB	24	(18)
PRMLIM	12	(C)
PRMPCTFF	152	(98)
PRMPCTOF	144	(90)
PRMPCTOG	156	(9C)
PRMPCTPF	146	(92)
PRMPCTTI	148	(94)
PRMSES	20	(14)
PRMSGN	60	(3C)
PRMSLICE	52	(34)
PRMSON	16	(10)
PRMSPMAX	4	(4)
PRMSSMSZ	2	(2)
PRMSVMAX	6	(6)
PRMSYS	48	(30)
PRMTASYA	162	(A2)
PRMTATKA	166	(A6)
PRMTATLR	160	(A0)
PRMTATPC	164	(A4)
PRMTCTOI	170	(AA)
PRMTCTTC	168	(A8)
PRMTERM	28	(1C)
PRMTERX	40	(28)
PRMTRANS	16	(10)
PRMUSERS	0	(0)
PRMXLEVL	44	(2C)

PRO (CICS, SERV)

This is the APL user profile. This is a special form of the DIR containing the user's quotas and authorization, accounting information, and session attributes. The PRO resides in main storage as an extension of the PTH and PTK while the user is signed on to the system. This control block is mapped by the APLKPRO macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	116	PRO	
0	(0)	CHARACTER	4	PROHEADR	PROFILE HEADER FOR CICS
0	(0)	SIGNED	2	PROLENHW	PROFILE LENGTH
2	(2)	SIGNED	2		RESERVED
4	(4)	CHARACTER	14	PROKEY	14 BYTE VSAM KEY
4	(4)	UNSIGNED	3	PROLIBNO	USER LIBRARY NUMBER
7	(7)	CHARACTER	1	PROCODE	PROFILE ENTRY CODE-X'00'
8	(8)	SIGNED	4	PRODEFWS	DEFAULT WS SIZE
12	(C)	SIGNED	4	PROSSMAX	SHARED STORAGE SIZE LIMIT IN BYTES
16	(10)	SIGNED	2	PROSSOBM	SHARED VARIABLE MAXIMUM
18	(12)	CHARACTER	1	PROTYPE	TYPE BYTE
		1...		PROFREE	FREE SPACE RECORD
		.1..		PROFEB	FILE EXTENT RECORD
		..1.		PROUPROF	USER PROFILE BIT
		...1			RESERVED
		.... 1...		PROCICS	CICS PROFILE FLAG
		.... .1..		PROPUB	PUBLIC LIBRARY
		.... ..1.			RESERVED
		1		PROPRIV	PRIVATE LIBRARY
19	(13)	CHARACTER	1	PROFLAG1	FLAG BYTE 1
		11..			RESERVED
		..1.		PROCWS	CONTINUE WORKSPACE SAVED
		...1 111.			RESERVED
		1		PROLOCK	USER IS LOCKED
20	(14)	CHARACTER	8	PROPSWD	LOGON PASSWORD
28	(1C)	BITSTRING	3	PROAUTH	USER AUTHORIZATION MASK
31	(1F)	CHARACTER	9		SESSION MODIFIABLE FIELDS
31	(1F)	CHARACTER	1	PROFLAG2	FLAG BYTE 2
		1...		PROCONCY	CONTINUOUS COPY FLAG
		.1..		PRODSES	DISPLAY APL SESSION ON USER TERMINAL
		..11			RESERVED
		.... 1...		PROHCDEF	HARDCOPY DESTINATION IS DEFINED FOR THIS USER
		.... .1..		PROPWCNG	LOGON PASSWORD CHANGED
		11			RESERVED
32	(20)	CHARACTER	4	PROXLATE	COPY OUTPUT TRANSLATE TABLE
36	(24)	CHARACTER	4	PRODEST	CICS HARDCOPY DESTINATION
40	(28)	SIGNED	4	PROMAXWS	MAXIMUM WS SIZE THIS USER
44	(2C)	SIGNED	4	PRODEFIL	AP121 DEFAULT FILE SIZE IN INCRS 4K-LEN(CI TRAILER)

PRO (CICS, SERV) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
48	(30)	SIGNED	4	PRODASMX	DASD SPACE LIMIT (IN INCR OF 4K-LENGTH(CI TRAILER))
52	(34)	SIGNED	4	PRODASU	DASD SPACE USED (AS ABOVE)
56	(38)	SIGNED	4		RESERVED
60	(3C)	SIGNED	4	PRONLATD	NO. OF LIBRARY ACCESSES TO DATE
64	(40)	SIGNED	4	PRONLATS	NO. OF LIBRARY ACCESSES THIS SESSION
68	(44)	SIGNED	4	PROCPUTD	CPU TIME TO DATE M SECS
72	(48)	SIGNED	4	PROCPUTS	CPU TIME THIS SESSION-MSECS
76	(4C)	SIGNED	4	PROTCTD	TERM CONNECT TIME TO DATE IN SECONDS (AS IN TCTS)
80	(50)	SIGNED	4	PROTCTS	TERM CONNECT TIME THIS SESS
84	(54)	SIGNED	4	PROSCRLS	SCROLL FILE SIZE IN LINES
88	(58)	CHARACTER	28	PRONAME	INSTALLATION DEFINED NAME FIELD

CROSS REFERENCE

PRO	0	(0)
PROAUTH	28	(1C)
PROCICS	18	X'08'
PROCODE	7	(7)
PROCONCY	31	X'80'
PROCPUTD	68	(44)
PROCPUTS	72	(48)
PROCWS	19	X'20'
PRODASMX	48	(30)
PRODASU	52	(34)
PRODEFIL	44	(2C)
PRODEFWS	8	(8)
PRODEST	36	(24)
PRODSES	31	X'40'
PROFEB	18	X'40'
PROFLAG1	19	(13)
PROFLAG2	31	(1F)
PROFREE	18	X'80'
PROHCDEF	31	X'08'
PROHEADR	0	(0)
PROKEY	4	(4)
PROLENHW	0	(0)
PROLIBNO	4	(4)
PROLOCK	19	X'01'
PROMAXWS	40	(28)
PRONAME	88	(58)
PRONLATD	60	(3C)
PRONLATS	64	(40)
PROPRIV	18	X'01'
PROPSWD	20	(14)
PROPUB	18	X'04'
PROPWCNG	31	X'04'
PROSCRLS	84	(54)
PROSSMAX	12	(C)
PROSSOBM	16	(10)
PROTCTD	76	(4C)
PROTCTS	80	(50)
PROTYPE	18	(12)
PROUPROF	18	X'20'
PROXLATE	32	(20)

PTH (ALL)

This is the PERTERM header. PTH provides information about the active user with regard to the system environment, and completes the communication path between interpreter and executor. This control block is mapped by the APLPTH macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	72	PTH	
0	(0)	CHARACTER	4	PTHWORD1	
0	(0)	CHARACTER	1	PTHASYNC	
		1...		PTHDATTN	DOUBLE-ATTENTION SIGNALLED
		.1..		PTHQEND	QUANTUM-END REQUESTED
		..1.		PTHCPULM	CPU LIMIT EXCEEDED.
		...1 1...			UNUSED
		.... .1..		PTHNOOUT	'CANCEL OUTPUT' SIGNAL RECEIVED
		.... ..1.		PTHFOFF	LINE-DROP OR BOUNCE
		1		PTHATTN	SINGLE ATTENTION SIGNALLED
1	(1)	CHARACTER	2		RESERVED
3	(3)	CHARACTER	1	PTHSUSP1	EXECUTOR SUSPENSION BITS
		1...		PTHCWBIT	CLOCK WAIT BIT
		.1..		PTHWABIT	YYWATE BIT
		..1.		PTHSVBIT	SH. VAR. WAIT BIT
		...1 1111			UNUSED
4	(4)	CHARACTER	1	PTHWSTAT	
		1...		PTHSVON	THIS USER SIGNED ON TO SVP
		.111 11..			RESERVED
		.... ..1.		PTHSINK	THIS IS A COPY SINK
		1		PTHSORS	THIS IS A COPY SOURCE
5	(5)	CHARACTER	1	PTHUSTAT	
		1...		PTHLOCKB	WE KEEP HIS KBD LOCKED
		.1..		PTHMDY	DATE FORMAT FLAG
		..1.		PTHMSBLK	WE BLOCK HIS MESSAGES
		...1		PTHMICRO	APL MICROCODE WILL BE USED.
		.... 1...		PTHFSAVL	RESERVED FOR FULLSCREEN EDIT
		.... .1..		PTHUEXTN	PTH EXTENSION (PTX) EXISTS
		11			RESERVED
6	(6)	SIGNED	2	PTHQVAR	
8	(8)	CHARACTER	4	PTHYYRC	
8	(8)	BITSTRING	2	PTHYYCOD	
		1...		PTHSPCLY	HI-ORDER BIT ON IF 'SPECIAL' YYCODE
10	(A)	BITSTRING	2	PTHSRCOD	
12	(C)	CHARACTER	2		RESERVED
14	(E)	SIGNED	2	PTHWIDTH	
16	(10)	BITSTRING	2		RESERVED
18	(12)	SIGNED	2	PTHCURSR	
20	(14)	SIGNED	4	PTHQSIZE	
24	(18)	BITSTRING	4	PTHPARM1	
28	(1C)	BITSTRING	4	PTHPARM2	
32	(20)	SIGNED	4	PTHWSLEN	
36	(24)	SIGNED	4	PTHACCNO	

PTH (ALL) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
IE A FLOATING POINT NUMBER OF MICROSECONDS, POSSIBLY FRACTIONAL. TIME-OF-DAY VALUES ARE FROM THE BEGINNING OF THE APL EPOCH. INTERVALS ARE SIMPLY MICROSECOND COUNTS.					
40	(28)	CHARACTER	8	PTHLOCAL	
48	(30)	CHARACTER	8	PTHCPUTM	
56	(38)	CHARACTER	8	PTHKEYTM	
64	(40)	CHARACTER	8	PTHCNCTM	
72	(48)	CHARACTER	0	PTHEND	END OF PERTERM HEADER.

CROSS REFERENCE

PTH	0	(0)
PTHACCNO	36	(24)
PTHASYNC	0	(0)
PTHATTN	0	X'01'
PTHCNCTM	64	(40)
PTHCPULM	0	X'20'
PTHCPUTM	48	(30)
PTHCURSR	18	(12)
PTHCWBIT	3	X'80'
PTHDATTN	0	X'80'
PTHEND	72	(48)
PTHFOFF	0	X'02'
PTHFSAVL	5	X'08'
PTHKEYTM	56	(38)
PTHLOCAL	40	(28)
PTHLOCKB	5	X'80'
PTHMDY	5	X'40'
PTHMICRO	5	X'10'
PTHMSBLK	5	X'20'
PTHNOOUT	0	X'04'
PTHPARM1	24	(18)
PTHPARM2	28	(1C)
PTHQEND	0	X'40'
PTHQSIZE	20	(14)
PTHQVAR	6	(6)
PTHSINK	4	X'02'
PTHSORS	4	X'01'
PTHSPCLY	8	X'80'
PTHSRCOD	10	(A)
PTHSUSP1	3	(3)
PTHSVBIT	3	X'20'
PTHSVON	4	X'80'
PTHUEXTN	5	X'04'
PTHUSTAT	5	(5)
PTHWABIT	3	X'40'
PTHWIDTH	14	(E)
PTHWORD1	0	(0)
PTHWSLEN	32	(20)
PTHWSTAT	4	(4)
PTHYYCOD	8	(8)
PTHYYRC	8	(8)

PTK (CICS, XSYS, AP)

This is the CICS/VS extension to the APL perterm header (the PTH). It contains CICS/VS-unique information about the user session. The PTK is the primary anchor for all storage and control blocks associated with the user session. In CICS/VS, the PTH is pointed to by the WSM, the user task CICS/VS TWA (which is an extension to the CICS/VS task control area), and the SGN entries. This control block is mapped by the APLKPTK macro.

APLXXPTX - APL EXECUTOR COMMON PERTERM EXTENTION
THE PERTERM PROVIDES PRIMARY CONTROL OF THE USER SESSION.
IT CONTAINS A SYSTEM INDEPENDENT HEADER, FOLLOWED THIS
COMMON EXECUTOR CONTROL BLOCK, THEN FOLLOWED BY THE
SUBSYSTEM DEPENDENT CONTROL BLOCKS.
THIS IS A PLS EXPANSION OF THE EXECUTOR COMMON PER TERM
EXTENTION (PTX). THE FOLLOWING ARE THE FIELDS:

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
72	(48)	STRUCTURE	108	PTX	ADDR OF ACTIVE WORKSPACE
72	(48)	A-ADDRESS	4	PTXWSM	ADDR OF VECTOR TABLE
76	(4C)	A-ADDRESS	4	PTXVCT	ADDR OF SP STACK
80	(50)	A-ADDRESS	4	PTXSTACK	ADDR OF SESSION TABLE
84	(54)	A-ADDRESS	4	PTXSMTBP	ADDR OF GDDX CONTROL TABLE
88	(58)	A-ADDRESS	4	PTXGXTBP	ADDR OF CURRENT GDM
92	(5C)	A-ADDRESS	4	PTXGXGDM	ADDR OF PRINT SERVICES TABLE
96	(60)	A-ADDRESS	4	PTXPRTBP	ADDR OF FILE SERVICES TABLE
100	(64)	A-ADDRESS	4	PTXFSTBP	ADDR OF ACTIVE ATTENTIO ROUTINE
104	(68)	A-ADDRESS	4	PTXATTN	
108	(6C)	BITSTRING	4	PTXFLAG	SUBSYSTEM IDENTIFYER
108	(6C)	BITSTRING	1	PTXSUBSY	THIS IS A TSO USER X'80'
		1...		PTXTSO	THIS IS A CMS USER X'40'
		.1..		PTXCMS	THIS IS A CICS USER X'20'
		..1.		PTXCICS	THIS IS A VSPC USER X'10'
		...1		PTXVSPC	DEBUG OPTIONS
109	(6D)	BITSTRING	1	PTXDEBUG	DEBUG NO MICROCODE TEST X'80'
		1...		DBGMICRO	DEBUG CANCEL ESTAE EXITS X'40'
		.1..		DBGNSTAE	DEBUG RESERVED X'20'
		..1.			DEBUG RESERVED X'10'
		...1			DEBUG RESERVED X'08'
		.... 1...			DEBUG RESERVED X'04'
		.... .1..			DEBUG ECHO STACK (CMD PARM)X'02'
		.... ..1.		DBGECHO	
		1		DBGMSG	GENERAL USE FLAGS
110	(6E)	BITSTRING	1	PTXFLAGS	PURGE ALTERNAT INPUT STACK X'80'
		1...		PTXAIPUR	
		.1..		PTXFSRST	
		..11			ADSM OWNS THE SESSION X'08'
		.... 1...		PTXADSM	RESERVED
111	(6F)	BITSTRING	1		DUMP SERVICES TABLE POINTER
112	(70)	A-ADDRESS	4	PTXDXTBP	VS APL RELEASE LEVEL
116	(74)	SIGNED	4	PTXLEVEL	GDDM TERMINAL TYPE CODE KIC0381
120	(78)	SIGNED	4	PTXCODE	

PTK (CICS, XSYS, AP) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
124	(7C)	CHARACTER	28	PTXSCRTH	ADSM PASSWORD RETURN AREA
124	(7C)	CHARACTER	8	PTXSMPSD	INDICATE ON WORD BOUNDARY
124	(7C)	SIGNED	4	PTXSMP1	ADSM PARM2 FIELD
128	(80)	SIGNED	4	PTXSMP2	ADDITIONAL ADSM PARM FIELDS
132	(84)	CHARACTER	20		INDICATE ON WORD BOUNDARY
132	(84)	SIGNED	4	PTXSMP3	ADSM PARM4 FIELD
136	(88)	SIGNED	4	PTXSMP4	ADSM PARM5 FIELD
140	(8C)	SIGNED	4	PTXSMP5	ADSM PARM6 FIELD
144	(90)	SIGNED	4	PTXSMP6	ADSM PARM7 FIELD
148	(94)	SIGNED	4	PTXSMP7	SF,OUT-ATTR,IN-ATTR,FLAGS HILITE
152	(98)	SIGNED	4	PTXHILIT	
152	(98)	CHARACTER	1	PTXHISF	OUTPUT ATTRIBUTE BYTE HILITE
153	(99)	CHARACTER	1	PTXHIAOT	INPUT ATTRIBUTE BYTE HILITE
154	(9A)	CHARACTER	1	PTXHIIOT	FLAGS FOR SESSION MANAGER HILITE
155	(9B)	BITSTRING	1	PTXHIFLG	
		1...		PTXHIOHI	INPUT HILITE REQUESTED HILITE
		.1..		PTXHIIHI	ADDRESS OF MSQ QUEUING RTN)MORE
156	(9C)	A-ADDRESS	4	PTXHELPQ	
160	(A0)	A-ADDRESS	4	PTXUSRWA	RESERVED
164	(A4)	SIGNED	4	PTXRSV01	RESERVED
168	(A8)	SIGNED	4	PTXRSV02	RESERVED
172	(AC)	SIGNED	4	PTXRSV03	RESERVED
176	(B0)	SIGNED	4	PTXRSV04	END OF PTX
176	(B0)	STRUCTURE	416	PTK	IMMEDIATELY FOLLOWS PTX
176	(B0)	CHARACTER	40		COMMON FIELDS
176	(B0)	A-ADDRESS	4	PTKSGN	ADDR OF SIGNON TABLE ENTRY
180	(B4)	A-ADDRESS	4	PTKGBL	ADDR OF GLOBAL TABLE
180	(B4)	CHARACTER	0	PTXEND	
184	(B8)	A-ADDRESS	4	PTKWSM	ADDR OF WORKSPACE
188	(BC)	A-ADDRESS	4	PTKRSV01	RESERVED
192	(C0)	CHARACTER	12	PTKRSAVE	SAVE REGS ACROSS INTERP
192	(C0)	A-ADDRESS	4	PTKSTKP	STACK POINTER
196	(C4)	A-ADDRESS	4	PTKTCA	TCA POINTER
200	(C8)	A-ADDRESS	4	PTKCSA	CSA POINTER
204	(CC)	CHARACTER	12	PTKIECBL	ECB LIST FOR INTERP PROCESS
204	(CC)	A-ADDRESS	4	PTKECB1	PTR TO DOUBLE ATTN ECB

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
208	(D0)	A-ADDRESS	4	PTKECB2	PTR TO REQUESTOR ECB
212	(D4)	SIGNED	4	PTKECBZ	ALL FFFF END OF LIST
216	(D8)	CHARACTER	56		HISTOGRAM CLOCKS (ALL STCK)
216	(D8)	CHARACTER	8	PTKTUNLK	LAST KEYBOARD UNLOCK
224	(E0)	CHARACTER	8	PTKTDPND	LAST DEPEND PROCESS DSPCH
232	(E8)	CHARACTER	8	PTKTLOAD	LAST)LOAD START
240	(F0)	CHARACTER	8	PTKTSAVE	LAST)SAVE START
248	(F8)	CHARACTER	8	PTKTCOPY	LAST)COPY START
256	(100)	CHARACTER	8	PTKISTIM	TIMESLICE INTERRUPT TIME
256	(100)	CHARACTER	8	PTKTENTR	ALSO INPUT COMPLETE TIME
264	(108)	CHARACTER	8	PTKTSON	SIGN ON TIME
272	(110)	CHARACTER	104	PTKXISAV	TRANSPARENT INTERRUPT SAVE AREA
272	(110)	CHARACTER	8	PTKXPSW	PROGRAM STATUS WORD
272	(110)	CHARACTER	4	PTKXWRD1	PSW 1ST WORD
276	(114)	CHARACTER	4	PTKXWRD2	
276	(114)	BITSTRING	1	PTKXFLGS	ILC,CC,PROGRAM MASK
277	(115)	A-ADDRESS	3	PTKXADDR	INSTRUCTION ADDR
280	(118)	CHARACTER	64	PTKXREGS	GENERAL REGISTERS
280	(118)	UNSIGNED	4	PTKXR0	
284	(11C)	UNSIGNED	60	PTKXREG	
284	(11C)	BITSTRING	1		
285	(11D)	A-ADDRESS	3	PTKXREGP	
344	(158)	BITSTRING	8	PTKXFPR0	FLOATING POINT REGS
352	(160)	BITSTRING	8	PTKXFPR2	
360	(168)	BITSTRING	8	PTKXFPR4	
368	(170)	BITSTRING	8	PTKXFPR6	
376	(178)	CHARACTER	16		INTERP INTERFACE FIELDS
376	(178)	BITSTRING	1	PTKPCOP	OP CODE SAVED BY TIMER
377	(179)	BITSTRING	1	PTKPMASK	SAVED CICS/INTERP PROGRAM MASK
378	(17A)	CHARACTER	1	PTKMFLG	MISCELLANEOUS FLAGS
		1...		PTKMXUSE	PTKXISAV IN USE (TIMER/PGFLT)
		.1..		PTKIWAIT	IFIX HAS ISSUED APLKWAIT
		..1.		PTKMINEX	APL TIMER EXIT SET
		...1		PTKINTRP	IN INTERPRETER CODE
		.... 1...		PTKMNDMP	NOP YYDUMP (TOOK EXEC DUMP)
		.... .1..		PTKMEXA	ABEND EXIT IN CONTROL
		.... ..1.		PTKTIMEO	TIMER EXIT IN CONTROL
		1		PTKMTPOP	TIMER POP OUTSIDE INTERPRETER

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
379	(17B)	CHARACTER	1	PTKDFLG	DISPATCHER FLAG BYTE
		1...		PTKDMAC	DISPATCHER MACRO WAS ISSUED
		.1..		PTKDWAIT	DISPATCHER IN A CICS WAIT
		..1.		PTKDELAY	INITIATING A TASK FOR QUAD-DELAY
		...1		PTKDWTED	WAIT DONE SINCE DISPATCH
380	(17C)	UNSIGNED	1	PTKOPRIO	OLD PRIORITY (BEFORE CHAP)
381	(17D)	CHARACTER	1	PTKRSV31	RESERVED
382	(17E)	SIGNED	2	PTKXDCNT	EXEC DUMP COUNTER, RESET AT TYI
384	(180)	SIGNED	4	PTKITCNT	HOG TIME SLICE CNTR
388	(184)	SIGNED	4	PTKIPROC	INTERP TIME PER INPUT
392	(188)	CHARACTER	32		MISCELLANEOUS SERVICES
392	(188)	SIGNED	2	PTKTABN	NUMBER OF TABS
394	(18A)	CHARACTER	28	PTKTABS	SESSION TABS
422	(1A6)	CHARACTER	1	PTKSYNC	SYNCPOINT FLAGS ALL TURNED
		1...		PTKSYDST	TURNED OFF BY DESTINATION MGR
		.1..		PTKSY125	TURNED OFF BY AP125 (DL/I)
423	(1A7)	CHARACTER	1	PTKRSV41	RESERVED
424	(1A8)	CHARACTER	24		SCREEN FORMAT FIELDS
424	(1A8)	SIGNED	2	PTKCURSR	CURSOR ROW ON SCREEN
426	(1AA)	SIGNED	2	PTKCURSC	CURSOR COLUMN ON SCREEN
428	(1AC)	SIGNED	2	PTKSESST	SESSION LINE AT SCREEN TOP
430	(1AE)	SIGNED	2	PTKSESSC	SESSION RELATIVE CURS LINE
432	(1B0)	CHARACTER	1	PTKFFLG1	SCREEN FORMAT FLAG BYTE
		1...		PTKFMSG	RECALL LINE NOT DISPLAYED
		.1..		PTKFINPT	INPUT PROVIDED BY SIGNON
		..1.		PTKFPROF	SCREEN REFORMATTED FOR PROFILE
		...1		PTKFDEST	ON IF PRODEST WAS CHANGED
		.... 1...		PTKFXLAT	ON IF PROXLATE WAS CHANGED
		.... .1..		PTKNOGDM	ON IF NO GDDM OPTION
		.... ..1.		PTKUZGDM	ON IF YES GDDM OPTION
433	(1B1)	UNSIGNED	1	PTKFMODE	CURRENT SCREEN MODE
434	(1B2)	SIGNED	2	PTKFPRIM	LEN OF DATA IN QPRIME BUFFER
436	(1B4)	A-ADDRESS	4	PTKPRIME	TO QPRIME (TERMINAL I/O) BUFFER
440	(1B8)	A-ADDRESS	4	PTKFECB	TO AP139 ECB
444	(1BC)	A-ADDRESS	4	PTKFFABS	PTR TO FILE ACCESS BLOCKS
448	(1C0)	CHARACTER	8		COMMON ECBS
448	(1C0)	BITSTRING	4	PTKT2ECB	ECB FOR DOUBLE ATTN
448	(1C0)	CHARACTER	3		
451	(1C3)	UNSIGNED	1	PTKT2RET	POST CODE FOR SIGNON
452	(1C4)	BITSTRING	4	PTKALECB	DISPATCHER ECB FOR CICS
452	(1C4)	BITSTRING	2		OS/VS ONLY PORTION
		1...			OS/VS WAIT BIT
		.1..		PTKALECO	OS/VS POST BIT
454	(1C6)	BITSTRING	2	PTKALEC2	DOS/VS ECB STARTS HERE
		1...		PTKALECD	DOS/VS POST BIT
456	(1C8)	CHARACTER	48		TERMINAL MANAGER FIELDS

PTK (CICS, XSYS, AP) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
456	(1C8)	A-ADDRESS	4	PTKRSV51	RESERVED
460	(1CC)	SIGNED	2	PTKTLEN	LINES ON SCREEN
462	(1CE)	SIGNED	2	PTKTWID	CHARACTERS PER LINE
464	(1D0)	CHARACTER	1	PTKTTYPE	TERMINAL TYPE AND FEATURES
		1...		PTKT327E	NDS TYPE TERMINAL
		.1..		PTKTAPTX	APL/TEXT FEATURE (DAF)
		..1.		PTKTTXKB	TEXT KEYBOARD
		...1		PTKTAPKB	APL KEYBOARD
		.... 1...		PTKTTXPR	TEXT PRINTER
465	(1D1)	CHARACTER	1	PTKSTAT	STATUS FLAGS
		1...		PTKPACT	TMGR ACTIVE (PRIM SCR)
		.1..		PTKTABND	TCTL HAS ABENDED
		..1.		PTKTRIP	REQUEST IN PROGRESS
		...1		PTKDOWT	TERM TASK WAIT NEEDED
		.... 1...		PTKTINWT	TERM TASK IN WAIT
		.... .1..		PTKPINWT	PRIM SCR PROCESS IN WAIT
		.... ..1.		PTKAINWT	ALT SCREEN PROCESS IN WAIT
466	(1D2)	SIGNED	2	PTKRSV61	RESERVED
468	(1D4)	A-ADDRESS	4	PTKBUFST	BUFFER FOR STD LOG SCREEN
472	(1D8)	A-ADDRESS	4	PTKBUFAL	BUFFER FOR ALT LOG SCREEN
476	(1DC)	A-ADDRESS	4	PTKTSFST	SCREEN FORMAT STANDARD
480	(1E0)	A-ADDRESS	4	PTKTSFAL	SCREEN FORMAT ALTERNATE
484	(1E4)	A-ADDRESS	4	PTKSSPST	STATUS FOR STD LOG DISPLY
488	(1E8)	A-ADDRESS	4	PTKSSPAL	STATUS FOR ALT LOG DISPLY
492	(1EC)	A-ADDRESS	4	PTKTOWN	CURRENT SCREEN STATUS PTR
496	(1F0)	A-ADDRESS	4	PTKRSV62	RESERVED
500	(1F4)	BITSTRING	4	PTKTWECB	ECB FOR TERM TRANS WAIT
504	(1F8)	CHARACTER	8		SHARED VARIABLE INTERFACE FLDS
504	(1F8)	A-ADDRESS	4	PTKSECBS	SET OF SSM ECBS FOR INTERPRETER
508	(1FC)	A-ADDRESS	4	PTKRSV71	RESERVED
512	(200)	CHARACTER	16		DESTINATION MANAGER FIELDS
512	(200)	A-ADDRESS	4	PTKDIB	HEAD OF DEST INTRFC BLOCK QUEUE
516	(204)	A-ADDRESS	4	PTKCCDIB	ADDR OF DIB FOR CONTIN COPY
520	(208)	CHARACTER	8	PTKSLCTM	STORE CLOCK LIMIT
528	(210)	CHARACTER	64		LIBRARY MANAGER FIELDS
528	(210)	CHARACTER	1	PTKCSFLG	LIBRARY SERVICES FLAGS
		1...		PTKCSSRZ	COPZ INVOKED FOR SOURCE
		.1..		PTKCSSKZ	COPZ INVOKED FOR SINK
		..1.		PTKCSFEF	COPI FIRST ENTRY
		...1		PTKCSSYS	SYSTEM ERROR DURING COPY
		.... 1...		PTKCSPAS	ACTIVE WS HAS PASSWORD
		.... .1..		PTKCSDIR	AP123 IS USING APLDIR FILE
529	(211)	CHARACTER	1	PTKRSV91	RESERVED
530	(212)	UNSIGNED	3	PTKLIBNO	CURRENT LIB NR (BINARY)
533	(215)	CHARACTER	11	PTKLNAME	CURRENT WS NAME

PTK (CICS, XSYS, AP) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
544	(220)	CHARACTER	8	PTKLPASS	CURRENT PASSWORD
552	(228)	SIGNED	4	PTKNEWSZ	SIZE OF NEW WS (LOAD)
556	(22C)	SIGNED	4	PTKCSRLN	LEN OF COPY WS
560	(230)	A-ADDRESS	4	PTKCSWSM	ADDR OF COPY WS
564	(234)	A-ADDRESS	4	PTKCSINK	ADDR OF SINK WS DURING COPY
568	(238)	SIGNED	4	PTKTRMID	TERMINAL ID
572	(23C)	A-ADDRESS	4	PTKGKTBP	ADDR OF GDDX CNTROL BLK
576	(240)	SIGNED	4	PTKGOECB	SIGNOFF ECB FOR GDDX
580	(244)	SIGNED	4	PTKSOECB	SIGNOFF ECB FOR ADSM
584	(248)	A-ADDRESS	4	PTKUEIBP	ADDR OF USER TASK EIB
588	(24C)	A-ADDRESS	4	PTKRSV92	RESERVED
592	(250)	CHARACTER	0	PTKPRO	PROFILE RECORD
592	(250)	STRUCTURE	116	PRO	PROFILE HEADER FOR CICS
592	(250)	CHARACTER	4	PROHEADR	PROFILE LENGTH
592	(250)	SIGNED	2	PROLENHW	RESERVED
594	(252)	SIGNED	2		14 BYTE VSAM KEY
596	(254)	CHARACTER	14	PROKEY	USER LIBRARY NUMBER
596	(254)	UNSIGNED	3	PROLIBNO	ALSO USER SIGN ON NUMBER
599	(257)	CHARACTER	1	PROCODE	DEFAULT WS SIZE
600	(258)	SIGNED	4	PRODEFWS	SHARED STORAGE SIZE LIMIT IN BYTES
604	(25C)	SIGNED	4	PROSSMAX	
608	(260)	SIGNED	2	PROSSOBM	TYPE BYTE
610	(262)	CHARACTER	1	PROTYPE	FREE SPACE RECORD
		1...		PROFREE	FILE EXTENT RECORD
		.1..		PROFEB	USER PROFILE BIT
		..1.		PROUPROF	RESERVED
		...1			CICS PROFILE FLAG
		.... 1...		PROCICS	PUBLIC LIBRARY
		.... .1..		PROPUB	RESERVED
		.... ..1.			PRIVATE LIBRARY
		1		PROPRIV	FLAG BYTE 1
611	(263)	CHARACTER	1	PROFLAG1	RESERVED
		11..			CONTINUE WORKSPACE SAVED
		..1.		PROCWS	RESERVED
		...1 111.			USER IS LOCKED
		1		PROLOCK	LOGON PASSWORD
612	(264)	CHARACTER	8	PROPSWD	USER AUTHORIZATION MASK

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
620	(26C)	BITSTRING	3	PROAUTH	SESSION MODIFIABLE FIELDS
623	(26F)	CHARACTER	9		FLAG BYTE 2
623	(26F)	CHARACTER	1	PROFLAG2	CONTINUOUS COPY FLAG
		1...		PROCONCY	DISPLAY APL SESSION ON USER TERMINAL
		.1..		PRODSES	
		..11			HARDCOPY DESTINATION IS DEFINED FOR THIS USER
		.... 1...		PROHCDEF	
		.... .1..		PROPWCNG	RESERVED
		11			COPY OUTPUT TRANSLATE TABLE
624	(270)	CHARACTER	4	PROXLATE	CICS HARDCOPY DESTINATION
628	(274)	CHARACTER	4	PRODEST	MAXIMUM WS SIZE THIS USER
632	(278)	SIGNED	4	PROMAXWS	AP121 DEFAULT FILE SIZE IN INCRS 4K-LEN(CI TRAILER)
636	(27C)	SIGNED	4	PRODEFIL	
640	(280)	SIGNED	4	PRODASMX	
644	(284)	SIGNED	4	PRODASU	RESERVED
648	(288)	SIGNED	4		NO. OF LIBRARY ACCESSES TO DATE
652	(28C)	SIGNED	4	PRONLATD	
656	(290)	SIGNED	4	PRONLATS	
660	(294)	SIGNED	4	PROCPUTD	CPU TIME THIS SESSION-MSECS
664	(298)	SIGNED	4	PROCPUTS	TERM CONNECT TIME TO DATE IN SECONDS (AS IN TCTS)
668	(29C)	SIGNED	4	PROTCTD	
672	(2A0)	SIGNED	4	PROTCTS	SCROLL FILE SIZE IN LINES
676	(2A4)	SIGNED	4	PROSCRLS	INSTALLATION DEFINED NAME FIELD
680	(2A8)	CHARACTER	28	PRONAME	

CROSS REFERENCE

DBGECHO	109 X'02'	PROFLAG2	623(26F)	PROTCTS	672(2A0)
DBGMICRO	109 X'80'	PROFREE	610 X'80'	PROTYPE	610(262)
DBGMSG	109 X'01'	PROHCDEF	623 X'08'	PROUPROF	610 X'20'
DBGNSTAE	109 X'40'	PROHEADR	592(250)	PROXLATE	624(270)
PRO	592(250)	PROKEY	596(254)	PTH	0 (0)
PROAUTH	620(26C)	PROLENHW	592(250)	PTHACCNO	36 (24)
PROCICS	610 X'08'	PROLIBNO	596(254)	PTHASYNC	0 (0)
PROCODE	599 (257)	PROLOCK	611 X'01'	PTHATTN	0 X'01'
PROCONCY	623 X'80'	PROMAXWS	632(278)	PTHCNCTM	64 (40)
PROCPUTD	660(294)	PRONAME	680(2A8)	PTHCPULM	0 X'20'
PROCPUTS	664(298)	PRONLATD	652(28C)	PTHCPUTM	48 (30)
PROCWS	611 X'20'	PRONLATS	656(290)	PTHCURSR	18 (12)
PRODASMX	640(280)	PROPRIV	610 X'01'	PTHCWBIT	3 X'80'
PRODASU	644(284)	PROPSWD	612(264)	PTHDATTN	0 X'80'
PRODEFIL	636(27C)	PROPUB	610 X'04'	PTHEND	72 (48)
PRODEFWS	600(258)	PROPWCNG	623 X'04'	PTHFOFF	0 X'02'
PRODEST	628(274)	PROSCRLS	676(2A4)	PTHFSAVL	5 X'08'
PRODSES	623 X'40'	PROSSMAX	604(25C)	PTHKEYTM	56 (38)
PROFEB	610 X'40'	PROSSOBM	608(260)	PTHLOCAL	40 (28)
PROFLAG1	611(263)	PROTCTD	668(29C)	PTHLOCKB	5 X'80'

CROSS REFERENCE

PTHMDY	5 X'40'
PTHMICRO	5 X'10'
PTHMSBLK	5 X'20'
PTHNOOUT	0 X'04'
PTHPARM1	24 (18)
PTHPARM2	28 (1C)
PTHQEND	0 X'40'
PTHQSIZE	20 (14)
PTHQVAR	6 (6)
PTHSINK	4 X'02'
PTHSORS	4 X'01'
PTHSPCLY	8 X'80'
PTHSRCOD	10 (A)
PTHSUSP1	3 (3)
PTHSVBIT	3 X'20'
PTHSVON	4 X'80'
PTHUEXTN	5 X'04'
PTHUSTAT	5 (5)
PTHWABIT	3 X'40'
PTHWIDTH	14 (E)
PTHWORD1	0 (0)
PTHWSLEN	32 (20)
PTHWSTAT	4 (4)
PTHYYCOD	8 (8)
PTHYYRC	8 (8)
PTKAINWT	465 X'02'
PTKALECB	452(1C4)
PTKALECD	454 X'80'
PTKALECO	452 X'40'
PTKALEC2	454(1C6)
PTKBUFAL	472(1D8)
PTKBUFST	468(1D4)
PTKCCDIB	516(204)
PTKCSDIR	528 X'04'
PTKCSFEF	528 X'20'
PTKCSFLG	528(210)
PTKCSINK	564(234)
PTKCSPAS	528 X'08'
PTKCSRLN	556(22C)
PTKCSSKZ	528 X'40'
PTKCSSRZ	528 X'80'
PTKCSSYS	528 X'10'
PTKCSWSM	560(230)
PTKCURSC	426(1AA)
PTKCURSR	424(1A8)
PTKDIB	512(200)
PTKDOWT	465 X'10'
PTKFDEST	432 X'10'
PTKFECB	440(1B8)
PTKFFABS	444(1BC)
PTKFFLG1	432(1B0)
PTKFINPT	432 X'40'
PTKFMODE	433(1B1)
PTKFMSG	432 X'80'
PTKFPRIM	434(1B2)
PTKFPROF	432 X'20'
PTKFXLAT	432 X'08'
PTKGKTBP	572(23C)
PTKGOECB	576(240)
PTKLIBNO	530(212)
PTKLNAME	533(215)
PTKLPASS	544(220)
PTKNEWSZ	552(228)
PTKNOGDM	432 X'04'
PTKPACT	465 X'80'
PTKPINWT	465 X'04'
PTKPRIME	436(1B4)
PTKPRO	592(250)
PTKRSV41	423(1A7)
PTKRSV51	456(1C8)
PTKRSV61	466(1D2)
PTKRSV62	496(1F0)
PTKRSV71	508(1FC)
PTKRSV81	524(20C)
PTKRSV91	529(211)
PTKRSV92	588(24C)
PTKSECBS	504(1F8)
PTKSESSC	430(1AE)
PTKSESST	428(1AC)
PTKSOECB	580(244)
PTKSSPAL	488(1E8)
PTKSSPST	484(1E4)
PTKSTAT	465(1D1)
PTKSYDST	422 X'80'
PTKSYNC	422(1A6)
PTKSY125	422 X'40'
PTKTABN	392(188)
PTKTABND	465 X'40'
PTKTABS	394(18A)
PTKTAPKB	464 X'10'
PTKTAPTX	464 X'40'
PTKTINWT	465 X'08'
PTKTLEN	460(1CC)
PTKTOWN	492(1EC)
PTKTRIP	465 X'20'
PTKTRMID	568(238)
PTKTSFAL	480(1E0)
PTKTSFST	476(1DC)
PTKTTXKB	464 X'20'
PTKTTXPR	464 X'08'
PTKTTYPE	464(1D0)
PTKTWECB	500(1F4)
PTKTWID	462(1CE)
PTKT2ECB	448(1C0)
PTKT2RET	451(1C3)
PTKT327E	464 X'80'
PTKUEIBP	584(248)
PTKUZGDM	432 X'02'
PTKXLTAB	520(208)
PTX	72 (48)
PTXADSM	110 X'08'
PTXAIPUR	110 X'80'
PTXATTN	104 (68)
PTXCICS	108 X'20'
PTXCMS	108 X'40'
PTXCODE	120 (78)
PTXDEBUG	109 (6D)
PTXDXTBP	112 (70)
PTXEND	180 (B4)
PTXFLAG	108 (6C)
PTXFLAGS	110 (6E)
PTXFSRST	110 X'40'
PTXFSTBP	100 (64)
PTXGXGDM	92 (5C)
PTXGXTBP	88 (58)
PTXHELPQ	156 (9C)
PTXHIAOT	153 (99)
PTXHIFLG	155 (9B)
PTXHIIHI	155 X'40'
PTXHIIOT	154 (9A)
PTXHILIT	152 (98)
PTXHIOHI	155 X'80'
PTXHISF	152 (98)
PTXLEVEL	116 (74)
PTXPRTBP	96 (60)
PTXRSV01	164 (A4)
PTXRSV02	168 (A8)
PTXRSV03	172 (AC)
PTXRSV04	176 (B0)
PTXSCRTH	124 (7C)
PTXSMPSD	124 (7C)
PTXSMP1	124 (7C)
PTXSMP2	128 (80)
PTXSMP3	132 (84)
PTXSMP4	136 (88)
PTXSMP5	140 (8C)
PTXSMP6	144 (90)
PTXSMP7	148 (94)
PTXSMTBP	84 (54)
PTXSTACK	80 (50)
PTXSUBSY	108 (6C)
PTXTSO	108 X'80'
PTXUSRWA	160 (A0)
PTXVCT	76 (4C)
PTXVSPC	108 X'10'
PTXWSM	72 (48)

This is the executor common services extension of the PTH, and contains session information associated with a single user. (The format of this layout is the one used in publications titled "Data Areas and Symbolic Names Cross-Reference Table", usually distributed on microfiche.) This control block is mapped by the APLXXPTX macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	108	PTX	IMMEDIATELY FOLLOWS PTH
0	(0)	A-ADDRESS	4	PTXWSM	ADDR OF ACTIVE WORKSPACE
4	(4)	A-ADDRESS	4	PTXVCT	ADDR OF VECTOR TABLE
8	(8)	A-ADDRESS	4	PTXSTACK	ADDR OF SP STACK
12	(C)	A-ADDRESS	4	PTXSMTBP	ADDR OF SESSION TABLE
16	(10)	A-ADDRESS	4	PTXGXTBP	ADDR OF GDDX CONTROL TABLE
20	(14)	A-ADDRESS	4	PTXGXGDM	ADDR OF CURRENT GDM
24	(18)	A-ADDRESS	4	PTXPRTBP	ADDR OF PRINT SERVICES TABLE
28	(1C)	A-ADDRESS	4	PTXFSTBP	ADDR OF FILE SERVICES TABLE
32	(20)	A-ADDRESS	4	PTXATTN	ADDR OF ACTIVE ATTENTIO ROUTINE
36	(24)	BITSTRING	4	PTXFLAG	PTX FLAG WORD
36	(24)	BITSTRING	1	PTXSUBSY	SUBSYSTEM IDENTIFYER
		1...		PTXTSO	THIS IS A TSO USER X'80'
		.1..		PTXCMS	THIS IS A CMS USER X'40'
		..1.		PTXCICS	THIS IS A CICS USER X'20'
		...1		PTXVSPC	THIS IS A VSPC USER X'10'
37	(25)	BITSTRING	1	PTXDEBUG	DEBUG OPTIONS
		1...		DBGMICRO	DEBUG NO MICROCODE TEST X'80'
		.1..		DBGNSTAE	DEBUG CANCEL ESTAE EXITS X'40'
		..1.			DEBUG RESERVED X'20'
		...1			DEBUG RESERVED X'10'
		.... 1...			DEBUG RESERVED X'08'
		.... .1..			DEBUG RESERVED X'04'
		.... ..1.		DBGECHO	DEBUG ECHO STACK (CMD PARM)X'02'
		1		DBGMSG	DEBUG ERROR MESSAGES X'01'
38	(26)	BITSTRING	1	PTXFLAGS	GENERAL USE FLAGS
		1...		PTXAIPUR	PURGE ALTERNAT INPUT STACK X'80'
		.1..		PTXFSRST	FULLSCREEN RESTORE REQUIREDX'40'
		..11			RESERVED X'30'
		.... 1...		PTXADSM	ADSM OWNS THE SESSION X'08'
39	(27)	BITSTRING	1		RESERVED

PTX (ALL) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
40	(28)	A-ADDRESS	4	PTXDXTBP	DUMP SERVICES TABLE POINTER
44	(2C)	SIGNED	4	PTXLEVEL	VS APL RELEASE LEVEL
48	(30)	SIGNED	4	PTXCODE	GDDM TERMINAL TYPE CODE KIC0381
52	(34)	CHARACTER	28	PTXSCRTH	7 WORD SCRATCH PAD AREA
52	(34)	CHARACTER	8	PTXSMPSD	ADSM PASSWORD RETURN AREA
52	(34)	SIGNED	4	PTXSMP1	ADSM PARM1 FIELD
56	(38)	SIGNED	4	PTXSMP2	ADSM PARM2 FIELD
60	(3C)	CHARACTER	20		ADDITIONAL ADSM PARM FIELDS
60	(3C)	SIGNED	4	PTXSMP3	ADSM PARM3 FIELD
64	(40)	SIGNED	4	PTXSMP4	ADSM PARM4 FIELD
68	(44)	SIGNED	4	PTXSMP5	ADSM PARM5 FIELD
72	(48)	SIGNED	4	PTXSMP6	ADSM PARM6 FIELD
76	(4C)	SIGNED	4	PTXSMP7	ADSM PARM7 FIELD
80	(50)	SIGNED	4	PTXHILIT	SF,OUT-ATTR,IN-ATTR,FLAGS HILITE
80	(50)	CHARACTER	1	PTXHISF	START FIELD 3270 ORDER HILITE
81	(51)	CHARACTER	1	PTXHIAOT	OUTPUT ATTRIBUTE BYTE HILITE
82	(52)	CHARACTER	1	PTXHIIOT	INPUT ATTRIBUTE BYTE HILITE
83	(53)	BITSTRING	1	PTXHIFLG	FLAGS FOR SESSION MANAGER HILITE
		1...		PTXHIOHI	OUTPUT HILITE REQUESTED HILITE
		.1..		PTXHIIHI	INPUT HILITE REQUESTED HILITE
84	(54)	A-ADDRESS	4	PTXHELPQ	ADDRESS OF MSQ QUEUING RTN)MORE
88	(58)	A-ADDRESS	4	PTXUSRWA	USER GETMAIN AREA ADDR EXIT
92	(5C)	SIGNED	4	PTXRSV01	RESERVED
96	(60)	SIGNED	4	PTXRSV02	RESERVED
100	(64)	SIGNED	4	PTXRSV03	RESERVED
104	(68)	SIGNED	4	PTXRSV04	RESERVED
108	(6C)	CHARACTER	0	PTXEND	END OF PTX

CROSS REFERENCE

DBGECHO	37	X'02'
DBGMICRO	37	X'80'
DBGMSG	37	X'01'
DBGNSTAE	37	X'40'
PTX	0	(0)
PTX	72	(48)
PTXADSM	38	X'08'
PTXADSM	110	X'08'
PTXAIPUR	38	X'80'
PTXAIPUR	110	X'80'
PTXATTN	32	(20)
PTXCICS	36	X'20'
PTXCMS	36	X'40'
PTXCODE	48	(30)
PTXDEBUG	37	(25)
PTXDXTBP	40	(28)
PTXEND	108	(6C)
PTXFLAG	36	(24)
PTXFLAGS	38	(26)
PTXFSRST	38	X'40'
PTXFSTBP	28	(1C)
PTXGXGDM	20	(14)
PTXGXTBP	16	(10)
PTXHELPQ	84	(54)
PTXHIAOT	81	(51)
PTXHIFLG	83	(53)
PTXHIIHI	83	X'40'
PTXHIIOT	82	(52)
PTXHILIT	80	(50)
PTXHIOHI	83	X'80'
PTXHISF	80	(50)
PTXLEVEL	44	(2C)
PTXPRTBP	24	(18)
PTXRSV01	92	(5C)
PTXRSV02	96	(60)
PTXRSV03	100	(64)
PTXRSV04	104	(68)
PTXSCRTH	52	(34)
PTXSMPSD	52	(34)
PTXSMP1	52	(34)
PTXSMP2	56	(38)
PTXSMP3	60	(3C)
PTXSMP4	64	(40)
PTXSMP5	68	(44)
PTXSMP6	72	(48)
PTXSMP7	76	(4C)
PTXSMTBP	12	(C)
PTXSTACK	8	(8)
PTXSUBSY	36	(24)
PTXTSO	36	X'80'
PTXUSRWA	88	(58)
PTXVCT	4	(4)
PTXVSPC	36	X'10'
PTXWSM	0	(0)

Share control vector (SCV), which contains information about a shared variable. It is used in communication between auxiliary processor and shared storage manager. This control block is mapped by the APLSCV macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	36	SCV	
0	(0)	SIGNED	4	SCVID	PROCESSOR ID
4	(4)	SIGNED	4	SCVPART	PARTNER'S PROCESSOR ID
8	(8)	SIGNED	2	SCVNO	OFFER SEQUENCE NUMBER
10	(A)	SIGNED	2	SCVINAME	PERSHARE INDEX
12	(C)	A-ADDRESS	4	SCVECB	ECB POINTER
16	(10)	A-ADDRESS	4	SCVVALUE	POINTER TO VALUE BUFFER
20	(14)	A-ADDRESS	4	SCVNAME	POINTER TO NAME
24	(18)	SIGNED	4	SCVSIZE	SIZE OF BUFFER IN BYTES
28	(1C)	SIGNED	4	SCVLEN	LENGTH OF VARIABLE
32	(20)	UNSIGNED	1	SCVATYPE	TYPE OF AUTH. CHECK
32	(20)	CHARACTER	1	SCVACV	ACCESS CONTROL VECTOR
		1...		SCVAMSPC	CONTROL MY SPECIFICATION
		.1..		SCVAPSPC	CONTROL PARTNER'S SPEC
		..1.		SCVAMREF	CONTROL MY REFERENCE
		...1		SCVAPREF	CONTROL PARTNER'S REFER
33	(21)	UNSIGNED	1	SCVALEVL	LEVEL OF AUTH. CHECK
33	(21)	CHARACTER	1	SCVFLAGS	FLAGS
		1...		SCVALLID	QUERY FOR ALL ID'S
		.1..		SCVNAMES	QUERY FOR ALL NAMES
		..1.		SCVFHOLD	HOLD AFTER COPY
		...1		SCVFGOFR	GENERAL OFFER
		.... 1...		SCVFISPC	IGNORE VALUE WAITING
		.... .1..		SCVFOFR2	OFFERED BY PARTNER
		.... ..1.		SCVFSHR	VARIABLE IS SHARED
		1		SCVFOFR1	OFFERED BY THIS USER
34	(22)	UNSIGNED	1	SCVNAMEL	LENGTH OF NAME
35	(23)	BITSTRING	1	SCVFLAG2	FLAGS
		1...		SCV2EBCD	MAP CHAR DATA TO/FROM EBCDIC
36	(24)	CHARACTER	0	SCVEND	END OF SCV
0	(0)	STRUCTURE	36	SCV	BEGINNING OF SCV
0	(0)	SIGNED	4	SCVID	THIS PROCESSOR'S NUMERIC ID
4	(4)	SIGNED	4	SCVPART	NUMERIC ID FOR PARTNER
8	(8)	SIGNED	4	SCVNO	NUMERIC OFFER SEQUENCE NUMBER
12	(C)	A-ADDRESS	4	SCVECB	POINTER TO THIS SVC'S ECB
16	(10)	A-ADDRESS	4	SCVVALUE	POINTER TO BUFFER CONTAINING VAL

SCV (ALL) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
20	(14)	A-ADDRESS	4	SCVNAME	POINTER TO NAME OF VALUE
24	(18)	SIGNED	4	SCVSIZE	SIZE OF SCVVALUE IN BYTES
28	(1C)	SIGNED	4	SCVLEN	LENGTH WANTED BY COPY, REF, SPEC
32	(20)	SIGNED	4	SCVMISC	MISC LENGTH AND FLAG BYTES
32	(20)	CHARACTER	1	SCVACV	LOGICAL ACCESS CONTROL VECTOR
		1...		SCVAMSPC	ON TO CONTROL MY SPECS X'80'
		.1..		SCVAPSPC	ON TO CONTROL PRTRS SPECS X'40'
		..1.		SCVAMREF	ON.TO CONTROL MY REFS X'20'
		...1		SCVAPREF	ON TO CONTROL PARTNER REFS X'10'
33	(21)	CHARACTER	1	SCVFLAGS	LOGICAL FLAGS
		1...		SCVALLID	QUERY FOR ALL PARTNER ID'S X'80'
		.1..		SCVNAMES	QUERY FOR ALL NAMES X'40'
		..1.		SCVFHOLD	HOLD AFTER SCOPY, NEXT OP X'20'
		...1		SCVFGOFR	GENERAL OFFER X'10'
		.... 1...		SCVFISPC	IQNOR VALUE WAITING ON SPC X'08'
		.... .1..		SCVFOFR2	VARBLE IS OFFERED BY PART X'04'
		.... ..1.		SCVFSHR	VARIABLE IS SHARED X'02'
		1		SCVFOFR1	VARIABLE OFFERED-THIS USER X'01'
34	(22)	BITSTRING	1	SCVNAMEL	NUMBER OF CHARACTERS IN NAME
35	(23)	CHARACTER	1	SCVFLAG2	RESERVED ALL BUT ONE
		1...		SCVFDOFR	DOUBLE OFFER X'80'
0	(0)	STRUCTURE	36	APLSCV	
0	(0)	CHARACTER	36	SCV	
0	(0)	SIGNED	4	SCVID	
4	(4)	SIGNED	4	SCVPART	VS APL TSO/CMS SCV PLS MAPPING
8	(8)	SIGNED	4	SCVNO	BEGINNING OF SCV
12	(C)	A-ADDRESS	4	SCVECB	THIS PROCESSOR'S NUMERIC ID
16	(10)	A-ADDRESS	4	SCVVALUE	NUMERIC ID FOR PARTNER
20	(14)	A-ADDRESS	4	SCVNAME	POINTER TO THIS SVC'S ECB
24	(18)	SIGNED	4	SCVSIZE	POINTER TO BUFFER CONTAINING VAL
28	(1C)	SIGNED	4	SCVLEN	POINTER TO THIS SVC'S ECB
32	(20)	SIGNED	4	SCVMISC	

SCV (ALL) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
32	(20)	CHARACTER	1	SCVACV	POINTER TO NAME OF VALUE
		1...		SCVAMSPC	SIZE OF SCVVALUE IN BYTES
		.1..		SCVAPSPC	LENGTH WANTED BY COPY, REF, SPEC
		..1.		SCVAMREF	MISC LENGTH AND FLAG BYTES
		...1		SCVAPREF	LOGICAL ACCESS CONTROL VECTOR
33	(21)	CHARACTER	1	SCVFLAGS	ON TO CONTROL PRTRS SPECS X'40'
		1...		SCVALLID	ON TO CONTROL MY REFS X'20'
		.1..		SCVNAMES	
		..1.		SCVFHOLD	LOGICAL FLAGS
		...1		SCVFGOFR	
		.... 1...		SCVFISPC	QUERY FOR ALL NAMES X'40'
		.... .1..		SCVFOFR2	
		.... ..1.		SCVFSHR	IQNOR VALUE WAITING ON SPC X'08'
		1		SCVFOFR1	
34	(22)	BITSTRING	1	SCVNAMEL	VARIABLE OFFERED-THIS USER X'01'
35	(23)	CHARACTER	1	SCVFLAG2	VARBLE IS OFFERED BY PART X'04'
		1...		SCVFDOFR	DOUBLE OFFER X'80'
36	(24)	CHARACTER	0	SCVEND	END OF BASIC SCV

CROSS REFERENCE

APLSCV	0 (0)
SCV	0 (0)
SCVACV	32 (20)
SCVALEVL	33 (21)
SCVALLID	33 X'80'
SCVAMREF	32 X'20'
SCVAMSPC	32 X'80'
SCVAPREF	32 X'10'
SCVAPSPC	32 X'40'
SCVATYPE	32 (20)
SCVECB	12 (C)
SCVEND	36 (24)
SCVFDOFR	35 X'80'
SCVFGOFR	33 X'10'
SCVFHOLD	33 X'20'
SCVFISPC	33 X'08'
SCVFLAGS	33 (21)
SCVFLAG2	35 (23)
SCVFOFR1	33 X'01'
SCVFOFR2	33 X'04'
SCVFSHR	33 X'02'
SCVID	0 (0)
SCVINAME	10 (A)
SCVLEN	28 (1C)
SCVMISC	32 (20)
SCVNAME	20 (14)
SCVNAMEL	34 (22)
SCVNAMES	33 X'40'
SCVNO	8 (8)
SCVPART	4 (4)
SCVSIZE	24 (18)
SCVVALUE	16 (10)
SCV2EBCD	35 X'80'

SGN (CICS, XSYS, AP)

This is the APL signon table. It contains an entry for each user signed on to APL. It identifies the user's terminal and points to the perterm. The SGN is pointed to by the GBL; individual entries in the SGN are pointed to by the PTK. This control block is mapped by the APLKSGN macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	12	SGN	
0	(0)	A-ADDRESS	4	SGNPTH	ADDR OF USER PERTERM
		1...		SGNFREE	ON FOR A FREE ENTRY
4	(4)	SIGNED	4	SGNUSID	USER ID
8	(8)	A-ADDRESS	4	SGNTCTTE	ADDR OF CICS TERM ENTRY

CROSS REFERENCE

SGN	0	(0)
SGNFREE	0	X'80'
SGNPTH	0	(0)
SGNTCTTE	8	(8)
SGNUSID	4	(4)

SHVAB (XSYS)

This is the shared variable block in shared memory, used within the shared storage manager. This control block is mapped by the APLSHVAB macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	0	APLSHVAB	
0	(0)	SIGNED	4	VABENTRY	
0	(0)	SIGNED	4	VABPID1	FIRST PROCESSOR ID OR A(PB1).
4	(4)	SIGNED	4	VABPID2	SECOND PROCESSOR ID OR A(PRB2).
8	(8)	HEX	1	VABACV	CURRENT ACCESS CONTROL VECTOR.
9	(9)	HEX	1	VABACV1	ACCESS CONTROL VECTOR FOR 1ST PARTNER.
10	(A)	HEX	1	VABACV2	ACCESS CONTROL VECTOR FOR 2ND PARTNER.
11	(B)	HEX	1	VABFLAGS	FLAGS AS DEFINED BELOW.
12	(C)	SIGNED	4	VABECB1	POINTER TO ECB FOR 1ST PARTNER.
16	(10)	SIGNED	4	VABECB2	POINTER TO ECB FOR 2ND PARTNER.
20	(14)	SIGNED	4	VABDATA	POINTER TO DATA IN SHARED MEMORY.
24	(18)	SIGNED	4	VABDSIZE	SIZE OF DATA ENTRY IN BYTES.
28	(1C)	HEX	1	VABFLAG2	FLAGS AS DEFINED BELOW.
29	(1D)	HEX	1	VABNAMEL	LENGTH OF NAME IN BYTES.
30	(1E)	HEX	1	VABNAME	NAME.
		.... ..1.		VABTYPE	"2"IDENTIFIER FOR VAB ENTRIES.
		...1 111.		VABSIZE	"VABNAME-VABENTRY"

DEFINITIONS FOR VABFLAGS:

1...	VABFOFR1	"X'80'"VARIABLE OFFERED BY 1ST PARTNER.
.1..	VABFOFR2	"X'40'"VARIABLE OFFERED BY 2ND PARTNER.
..1.	VABFSPC1	"X'20'"VARIABLE LAST SPECIFIED BY 1ST PARTNER.
...1	VABFSPC2	"X'10'"VARIABLE LAST SPECIFIED BY 2ND PARTNER.
.... 1...	VABFPID1	"X'08'"VABPID1 CONTAINS ADDRESS OF PRB1.
.... .1..	VABFPID2	"X'04'"VABPID2 CONTAINS ADDRESS OF PRB2.
.... ..1.	VABFLCK1	"X'02'"VARIABLE HELD BY 1ST PARTNER.
....1	VABFLCK2	"X'01'"VARIABLE HELD BY 2ND PARTNER.

OFFSETS	TYPE	LENGTH	NAME	DESCRIPTION
DEFINITIONS FOR THE ACCESS CONTROL VECTORS (VABACV, VABACV1 AND VABACV2):				
	1...		VABASPC1	"X'80'"CONTROL FOR SPECIFICATIONS BY PARTNER 1.
	.1..		VABASPC2	"X'40'"CONTROL FOR SPECIFICATIONS BY PARTNER 2.
	..1.		VABAREF1	"X'20'"CONTROL FOR REFERENCES BY PARTNER 1.
	...1		VABAREF2	"X'10'"CONTROL FOR REFERENCES BY PARTNER 2.
DEFINITIONS FOR VABFLAG2:				
	1...		VABFSIN1	"X'80'"1ST PARTNER'S SPECIFICATIONS INTERLOCKED.
	.1..		VABFSIN2	"X'40'"2ND PARTNER'S SPECIFICATIONS INTERLOCKED.
	..1.		VABFRIN1	"X'20'"1ST PARTNER'S REFERENCES INTERLOCKED.
	...1		VABFRIN2	"X'10'"2ND PARTNER'S REFERENCES INTERLOCKED.

CROSS REFERENCE

APLSHVAB	0	(0)
VABACV	8	(8)
VABACV1	9	(9)
VABACV2	10	(A)
VABAREF1	30	X'20'
VABAREF2	30	X'10'
VABASPC1	30	X'80'
VABASPC2	30	X'40'
VABDATA	20	(14)
VABDSIZE	24	(18)
VABECB1	12	(C)
VABECB2	16	(10)
VABENTRY	0	(0)
VABFLAGS	11	(B)
VABFLAG2	28	(1C)
VABFLCK1	30	X'02'
VABFLCK2	30	X'01'
VABFOFR1	30	X'80'
VABFOFR2	30	X'40'
VABFPID1	30	X'08'
VABFPID2	30	X'04'
VABFRIN1	30	X'20'
VABFRIN2	30	X'10'
VABFSIN1	30	X'80'
VABFSIN2	30	X'40'
VABFSPC1	30	X'20'
VABFSPC2	30	X'10'
VABNAME	30	(1E)
VABNAMEL	29	(1D)
VABPID1	0	(0)
VABPID2	4	(4)
VABSIZE	30	X'1E'
VABTYPE	30	X'02'

STK (CICS, XSYS, AP)

This is the stack entry and stack block control block. It describes an entry in a work stack and control information at the beginning of a stack block. (The format of this layout is the one used in publications titled "Data Areas and Symbolic Names Cross-Reference Table," usually distributed on microfiche.) This control block is mapped by the APLXSTK macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	16	STK	
0	(0)	A-ADDRESS	4	STKPEXIT	ADDR OF ABEND EXIT ROUTINE
		1...		STKPEX1	FLAG IS ON AT OWNER LEVEL
4	(4)	A-ADDRESS	4	STKP13	SAVED VALUE OF REG13 AS USED AT
4	(4)	SIGNED	2	STKBSIZE	SIZE OF ENTIRE BLOCK (OWNER)
6	(6)	SIGNED	2	STKBLEN1	LEN. OF 1ST STACK ENTRY (OWNER)
8	(8)	CHARACTER	8	STKPNAME	ENTRY POINT NAME (OR OWNER NAME)

CROSS REFERENCE

STK	0	(0)
STKBLEN1	6	(6)
STKBSIZE	4	(4)
STKPEXIT	0	(0)
STKPEX1	0	X'80'
STKPNAME	8	(8)
STKP13	4	(4)

TBL (VSPC, AP)

This is the VSPC AP 126 address and request table. It is mapped by the APLXGTBL macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	0	TBL	
0	(0)	SIGNED	4	TBL126RQ	AP 126 REQUEST CODE
4	(4)	CHARACTER	8	TBLRQNAM	GDDM CALL NAME
12	(C)	A-ADDRESS	4	TBLRQPT	POINTER TO TABLE ENTRY

CROSS REFERENCE

TBL	0	(0)
TBLRQNAM	4	(4)
TBLRQPT	12	(C)
TBL126RQ	0	(0)

TCD (CICS, AP)

This is the CICS/VS executor translation routine request block. It is mapped by the APLKTCD macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	0	TRAN	PARM LIST PTR IN R1
0	(0)	HEX	1	TRANCD	REQUEST CODES
1	(1)	HEX	1	TRANOPT	OPTIONS
2	(2)	SIGNED	2	TRANDLEN	LENGTH DATA TO TRANSLATE
4	(4)	A-ADDRESS	4	TRANDPTR	PTR TO SOURCE DATA
		.... 1...		TRANENDM	"*" END OF MINIMUM LIST
END OF MINIMUM LIST, OPTION FLAGS INDICATE PRESENCE OF OTHER VALS					
8	(8)	A-ADDRESS	4	TRANTPTR	PTR TO TARGET AREA IF MOVE REQUESTED
12	(C)	A-ADDRESS	4	TRANTEND	PTR, END TARGET AREA(EXP/CONT) FOR CALL, PASS MAX END ON RETURN, PTR TO NEXT UNUSED BYTE
16	(10)	A-ADDRESS	4	TRANTBL	PTR TO CALLER-PROVIDED TABLE(1-FOR-1)
		...1 .1..		TRANEND	"*" END OF TRAN PARM LIST
TRANSLATE TABLE REQUEST CODES (USED WITH TRAN ROUTINE)					
		....		TRANZS	"0" ZCODE TO STANDARD EBCDIC
		1		TRANSZ	"1" STANDARD EBCDIC TO ZCODE
		.... ..1.		TRANSO	"2" STANDARD EBC TO OLD 3270 BASIC
		11		TRANSOP1	"3" STANDARD EBC TO OLD 3270 APLTEXT,PG 1 RESERVE ONE CODE, PAGE 2
		.... .1.1		TRANS88	"5" STANDARD EBC TO OLD 3288 TEXT FEATURE
		.... .11.		TRANOS	"6" OLD 3270 TO STANDARD EBC
		.... .11.		TRANOSP1	"6" SAME TABLE FOR PAGE1 RESERVE ONE CODE, PAGE 2
		.... 1...		TRANSN	"8" STANDARD EBC TO NEW 3270
		.... 1...		TRANSNP1	"8" NEW 3270 APLTEXT PG1 = BASIC PG 1 RESERVE ONE CODE FOR PAGE 2
		.... 1.1.		TRANNS	"10" NDS 3270 TO STANDARD EBC
		.... 1.1.		TRANNSP1	"10" USE SAME TABLE FOR APLTEXT PG 1 RESERVE ONE CODE, PAGE 2
		1...		TRANUSR	"X'80'" USER-PROVIDED TRANSLATION
OPTION FLAGS USED WITH TRAN ROUTINE (0 IF NOT APPLICABLE)					
		1...		TRANAPLT	"X'80'" APLTEXT DOUBLE TABLE, ONLY WITH TRANSO,TRANOS,TRANSN,TRANNS
		.1..		TRANMOVE	"X'40'" DO MOVE, WITH ANY
		..1.		TRANEXP	"X'20'" EXPANSION MAY OCCUR (TLEN REQUIRED) ONLY WITH TRANSO,TRANSN
		...1		TRANCONT	"X'10'" CONTR MAY OCCUR (TLEN REQD IF MOVE) ONLY WITH TRANOS,TRANNS
RETURN CODES FROM TRAN ROUTINE					
		....		TRANRCOK	"0" OK
		..11 .1.1		TRANRCLE	"53" LENGTH ERROR (SAME AS TRCNOSP)
		.11. .1.1		TRANRCSE	"101" INVALID PARMS (SAME AS TRCBAD)

OFFSETS	TYPE	LENGTH	NAME	DESCRIPTION
EQUATES WHICH GOVERN PAGE1, PAGE2 TABLE ORDER IN TRAN RTN AND TBLS				
	....		TRANPG1	"0" DISP FROM BASIC PTR, PAGE 1TABLE
			TRANPG2	"TRANPG1+256" DISP FROM BASIC PTR, PAGE2 TABLE

CROSS REFERENCE

TRAN	0	(0)
TRANAPLT	16	X'80'
TRANCD	0	(0)
TRANCONT	16	X'10'
TRANDLEN	2	(2)
TRANDPTR	4	(4)
TRANEND	16	X'14'
TRANENDM	4	X'08'
TRANEXP	16	X'20'
TRANMOVE	16	X'40'
TRANNS	16	X'0A'
TRANNSP1	16	X'0A'
TRANOPT	1	(1)
TRANOS	16	X'06'
TRANOSP1	16	X'06'
TRANPG1	16	X'00'
TRANPG2 =		256
TRANRCLE	16	X'35'
TRANRCOK	16	X'00'
TRANRCSE	16	X'65'
TRANSN	16	X'08'
TRANSNP1	16	X'08'
TRANSO	16	X'02'
TRANSOP1	16	X'03'
TRANSZ	16	X'01'
TRANS88	16	X'05'
TRANTBL	16	(10)
TRANTEND	12	(C)
TRANTPTR	8	(8)
TRANUSR	16	X'80'
TRANZS	16	X'00'

TRD (XSYS, AP)

This is the common system services translation request descriptor. It is mapped by the APLXTRD macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	48	TRD	PARM LIST PTR IN R1
0	(0)	UNSIGNED	1	TRDREQCD	REQUEST CODE
1	(1)	UNSIGNED	1		RESERVED
2	(2)	SIGNED	2	TRDRC	RETURN CODE
4	(4)	SIGNED	4	TRDSDLEN	LENGTH DATA TO TRANSLATE
8	(8)	A-ADDRESS	4	TRDSDPTR	PTR TO SOURCE DATA
12	(C)	A-ADDRESS	4	TRDTDPTR	PTR TO TARGET DATA (MAY=SRCE)
16	(10)	A-ADDRESS	4	TRDUTRAN	USER TRANSLATE TABLE (OPT)
20	(14)	SIGNED	4	TRDR14	R14 SAVE AREA
24	(18)	SIGNED	24	TRDREGS	WORK REG SAVE AREA (R2-7)

CROSS REFERENCE

TRD	0	(0)
TRDRC	2	(2)
TRDREGS	24	(18)
TRDREQCD	0	(0)
TRDR14	20	(14)
TRDSDLEN	4	(4)
TRDSDPTR	8	(8)
TRDTDPTR	12	(C)
TRDUTRAN	16	(10)

TRQ (CICS, XSYS)

This is the CICS/VS executor terminal request descriptor for non-GDDM terminal services. It is mapped by the APLKTRQD macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	20	TRQD	USED WITH APLKTERM MACRO
0	(0)	BITSTRING	1	TRQTYP1	MAIN TYPE CODE-SEE BELOW
1	(1)	BITSTRING	1	TRQTYP2	SECONDARY TYPE FLAGS
		1111			RESERVED
		.... 1...		TRQREST	RESTORE SCREEN
		.... .1..		TRQALARM	RING ALARM
		.... ..1.		TRQCUR	SET CURSOR POSITION
		1		TRQHC	HARDCOPY
2	(2)	BITSTRING	1	TRQOPT	OPTION FLAGS
		1...		TRQEBC	ON RD/WR, DATA IN EBCDIC
		.1..		TRQWAIT	ON WRITE, WAIT FOR COMPL
		..1.		TRQRFOR	ON FORMAT, REFORMAT
		...1		TRQFCHK	ON FORMAT, DO FORMAT CHECK
		.... 1...		TRQWW	ON RD, WRITE TO FOLLOW
		.... .1..		TRQNULL	ON WR,TRAILING BLNKS=NULLS
		.... ..1.		TRQNODAT	ON RD,PASS NO DATALEN INFO ON GD,PASS FLDLEN, NO DATA
		1		TRQALT	ALTERNATE SCREEN
3	(3)	UNSIGNED	1	TRQFLAG	RESERVED FOR MORE FLAGS
4	(4)	SIGNED	2	TRQFNUM	NUMBER OF FIELDS
6	(6)	SIGNED	2	TRQLEN	BUFFER/DATA LEN
8	(8)	A-ADDRESS	4	TRQBUF	BUFFER PTR
12	(C)	UNSIGNED	1	TRQCOD1	READ COMPL CODE
13	(D)	UNSIGNED	1	TRQCOD2	READ COMPL CODE MODIFIER
14	(E)	SIGNED	2	TRQCFID	CURSOR FIELD NO
16	(10)	SIGNED	2	TRQCROW	ROW ADDR, CURSOR
18	(12)	SIGNED	2	TRQCCOL	COL ADDR, CURSOR
20	(14)	CHARACTER	0	TRQEND	END OF TRQD

CROSS REFERENCE

TRQALARM	1	X'04'
TRQALT	2	X'01'
TRQBUF	8	(8)
TRQCCOL	18	(12)
TRQCFID	14	(E)
TRQCOD1	12	(C)
TRQCOD2	13	(D)
TRQCROW	16	(10)
TRQCUR	1	X'02'
TRQD	0	(0)
TRQEBC	2	X'80'
TRQEND	20	(14)
TRQFCHK	2	X'10'
TRQFLAG	3	(3)
TRQFNUM	4	(4)
TRQHC	1	X'01'
TRQLEN	6	(6)
TRQNODAT	2	X'02'
TRQNULL	2	X'04'
TRQOPT	2	(2)
TRQREST	1	X'08'
TRQRFOR	2	X'20'
TRQTYP1	0	(0)
TRQTYP2	1	(1)
TRQWAIT	2	X'40'
TRQWW	2	X'08'

TSOGL (TSO, XSYS, AP)

This is the TSO executor global table mapping. For more detailed description, see "Executor Data Areas." It is mapped by the APLTSOGL macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	0	TSOGL	
0	(0)	FLOATING	8	PTH	

THE PERTERM HEADER PROVIDES INFORMATION ABOUT THE ACTIVE USER WITH REGARD TO THE SYSTEM ENVIRONMENT, AND COMPLETES THE COMMUNICATION PATH BETWIXT INTERPRETER AND EXECUTOR.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	SIGNED	4	PTHWORD1	
0	(0)	HEX	1	PTHASYNC	
		1...		PTHDATTN	"X'80'". DOUBLE-ATTENTION SIGNALLED
		.1..		PTHQEND	"X'40'". QUANTUM-END REQUESTED
		..1.		PTHCPULM	"X'20'" CPU LIMIT EXCEEDED.
		.... .1..		PTHNOOUT	"X'04'" 'CANCEL OUTPUT' SIGNAL RECEIVED.
		.... ..1.		PTHFOFF	"X'02'". LINE-DROP OR BOUNCE
		1		PTHATTN	"X'01'". SINGLE ATTENTION SIGNALLED
1	(1)	HEX	1	(2)	RESERVED
3	(3)	HEX	1	PTHSUSP1	SUPERVISOR SUSPENSION BITS
		1...		PTHCWBIT	"X'80'". CLOCK WAIT BIT
		.1..		PTHWABIT	"X'40'". YYWATE BIT
		..1.		PTHSVBIT	"X'20'". SH. VAR. WAIT BIT

PTHWSTAT HOLDS THE PROCESSING STATE OF THIS WS

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
4	(4)	HEX	1	PTHWSTAT	
		1...		PTHSVON	"X'80'". THIS USER SIGNED ON TO SVP
		.... ..1.		PTHSINK	"X'02'". THIS IS A COPY SINK
		1		PTHSORS	"X'01'". THIS IS A COPY SOURCE

PTHUSTAT RECALLS THINGS WE'RE DOING FOR OR TO THIS USER

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
5	(5)	HEX	1	PTHUSTAT	
		1...		PTHLOCKB	"X'80'". WE KEEP HIS KBD LOCKED
		.1..		PTHMDY	"X'40'". DATE FORMAT FLAG

PTHMDY=1='MM/DD/YY'
PTHMDY=0='DD-MM-YY'

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
		..1.		PTHMSBLK	"X'20'". WE BLOCK HIS MESSAGES
		...1		PTHMICRO	"X'10'". APL MICROCODE WILL BE USED.
		.... 1...		PTHFSAVL	"X'08'" RESERVED FOR FULLSCREEN EDIT
		.... .1..		PTHUEXTN	"X'04'" PTH EXTENSION (PTX) EXISTS

PTHQVAR IS THE MAXIMUM NUMBER OF VARIABLES HE MAY SHARE

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
6	(6)	SIGNED	2	PTHQVAR	

PTHYYCOD CONTAINS THE YYCODE OF THE LAST SVCC ISSUED
PTHSRCOD CONTAINS THE RETURN CODE THAT RESULTED.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
8	(8)	SIGNED	4	PTHYYRC	
8	(8)	SIGNED	2	PTHYYCOD	
		1...		PTHSPCLY	"X'80'"HI-ORDER BIT ON IF 'SPECIAL' YYCODE
10	(A)	SIGNED	2	PTHSRCOD	

```
 OFFSETS     TYPE      LENGTH  NAME            DESCRIPTION

================================================================================
PTHWIDTH IS THIS TERMINAL'S CURRENT LINE-WIDTH SETTING
--------------------------------------------------------------------------------
  12     (C) SIGNED        2                   RESERVED
  14     (E) SIGNED        2   PTHWIDTH
================================================================================
PTHCURSR IS THE TYPEBALL POSITION RESULTING FROM THE LAST
TYO OR TYI. PTHCURSR=0='AT THE LEFT MARGIN'.
--------------------------------------------------------------------------------
  16    (10) SIGNED        2                   RESERVED
  18    (12) SIGNED        2   PTHCURSR
================================================================================
PTHQSIZE IS THE MAXIMUM SIZE A SHARED VARIABLE MAY OBTAIN
--------------------------------------------------------------------------------
  20    (14) SIGNED        4   PTHQSIZE
================================================================================
PTHPARM1, PTHPARM2 ARE RETURN PARAMETER FIELDS FOR
SOME SVCC FUNCTIONS.
--------------------------------------------------------------------------------
  24    (18) FLOATING      8
--------------------------------------------------------------------------------
  24    (18) SIGNED        4   PTHPARM1
--------------------------------------------------------------------------------
  28    (1C) SIGNED        4   PTHPARM2
================================================================================
PTHWSLEN CONTAINS THE SIZE OF THE WS ADDRESS SPACE
--------------------------------------------------------------------------------
  32    (20) SIGNED        4   PTHWSLEN
================================================================================
PTHACCNO CONTAINS THE BINARY ACCOUNT NUMBER OF THIS USER
--------------------------------------------------------------------------------
  36    (24) SIGNED        4   PTHACCNO
================================================================================
TIME FIELDS: ALL ARE IN APL-STANDARD TIME FORMAT
 IE A FLOATING POINT NUMBER OF MICROSECONDS,
 POSSIBLY FRACTIONAL. TIME-OF-DAY VALUES
 ARE FROM THE BEGINNING OF THE APL EPOCH.
 INTERVALS ARE SIMPLY MICROSECOND COUNTS.
PTHLOCAL IS THE OFFSET OF THIS USER FROM GMT.
PTHCPUTM IS THE CPU TIME THIS SESSION.
PTHKEYTM IS THE UNLOCKED-KBD TIME THIS SESSION.
PTHCNCTM IS THE DATE/TIME HE SIGNED ON.
--------------------------------------------------------------------------------
  40    (28) FLOATING      8
--------------------------------------------------------------------------------
  40    (28) FLOATING      8   PTHLOCAL
--------------------------------------------------------------------------------
  48    (30) FLOATING      8   PTHCPUTM
--------------------------------------------------------------------------------
  56    (38) FLOATING      8   PTHKEYTM
--------------------------------------------------------------------------------
  64    (40) FLOATING      8   PTHCNCTM
             .1.. 1...         PTHSIZE         "*-PTH" SIZE OF PERTERM HEADER.
================================================================================
 PTX
THE 'COMMON EXECUTOR PERTERM EXTENTION' IMMEDIATELY
FOLLOWS THE PTH IN THE GLOBAL TABLE.  THIS CONTROL
BLOCK FACILITATES COMMUNICATION BETWEEN THE COMMON
EXECUTOR MODULES.
--------------------------------------------------------------------------------
  72    (48) FLOATING      8   PTX             PERTERM EXTENSION FOR EXECUTOR
                                               COMMON SERVICES
```

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
OTHER TABLE POINTERS					
72	(48)	A-ADDRESS	4	PTXWSM	ADDR OF ACTIVE WORKSPACE
76	(4C)	A-ADDRESS	4	PTXVCT	ADDR OF VECTOR TABLE
80	(50)	A-ADDRESS	4	PTXSTACK	ADDR OF SP STACK
84	(54)	A-ADDRESS	4	PTXSMTBP	ADDR OF SESSION TABLE
88	(58)	A-ADDRESS	4	PTXGXTBP	ADDR OF GDDX CONTROL TABLE
92	(5C)	A-ADDRESS	4	PTXGXGDM	ADDR OF CURRENT GDM
96	(60)	A-ADDRESS	4	PTXPRTBP	ADDR OF PRINT SERVICES TABLE
100	(64)	A-ADDRESS	4	PTXFSTBP	ADDR OF FILE SERVICES TABLE
104	(68)	A-ADDRESS	4	PTXATTN	ADDR OF ACTIVE ATTENTION ROUTINE
108	(6C)	SIGNED	4	PTXFLAG	DEFINE WORD OF FLAGS
108	(6C)	HEX	1	PTXSUBSY	SUBSYSTEM FLAGS
		1...		PTXTSO	"X'80'" THIS IS A TSO USER
		.1..		PTXCMS	"X'40'" THIS IS A CMS USER
		..1.		PTXCICS	"X'20'" THIS IS A CICS USER
		...1		PTXVSPC	"X'10'" THIS IS A VSPC USER
109	(6D)	HEX	1	PTXDEBUG	VARIOUS DEBUG OPTIONS
		1...		DBGMICRO	"X'80'" DEBUG CANCEL MICROCODE TEST DEBUG
		.1..		DBGNSTAE	"X'40'" DEBUG CANCEL ESTAE EXITS DEBUG
		.... ..1.		DBGECHO	"X'02'" DEBUG ECHO STACK (CMD PARM) DEBUG
		1		DBGMSG	"X'01'" DEBUG ERROR MESSAGES DEBUG
110	(6E)	HEX	1	PTXFLAGS	GENERAL USE FLAGS
		1...		PTXAIPUR	"X'80'" PURGE THE ALTERNATE INPUT STACK
		.1..		PTXFSRST	"X'40'" FULLSCREEN RESTORE REQUIRED
		.... 1...		PTXADSM	"X'08'" ADSM OWNS THE SESSION
111	(6F)	HEX	1		RESERVED
112	(70)	A-ADDRESS	4	PTXDXTBP	DUMP SERVICES TABLE POINTER
116	(74)	SIGNED	4	PTXLEVEL	VS APL RELEASE LEVEL
120	(78)	SIGNED	4	PTXCODE	TERMINAL TYPE (GDDM) CODE
COMMON WORK AREA, USED FOR/BY ADSM AND IS IS AVAILABLE FOR OTHER USERS AS A SCRATCH					
124	(7C)	CHARACTER	28	PTXSCRTH	7 WORD SCRATCH PAD AREA
124	(7C)	CHARACTER	8	PTXSMPSD	ADSM PASSWORD RETURN AREA
124	(7C)	CHARACTER	8	PTXSMPRO	ADSM PROFILE OPTION (OR BLANKS)
124	(7C)	A-ADDRESS	4	PTXSMP1	ADSM PARM1 FIELD
128	(80)	A-ADDRESS	4	PTXSMP2	ADSM PARM2 FIELD
132	(84)	A-ADDRESS	4	PTXSMP3	ADSM PARM3 FIELD
136	(88)	A-ADDRESS	4	PTXSMP4	ADSM PARM4 FIELD
140	(8C)	A-ADDRESS	4	PTXSMP5	ADSM PARM5 FIELD

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
144	(90)	A-ADDRESS	4	PTXSMP6	ADSM PARM6 FIELD
148	(94)	A-ADDRESS	4	PTXSMP7	ADSM PARM7 FIELD
152	(98)	SIGNED	4	PTXHILIT	1D,II,00,F0 SF,OUT-ATTR,IN-ATTR,FLAGS HILITE
152	(98)	HEX	1	PTXHISF	START FIELD 3270 ORDER HILITE
153	(99)	HEX	1	PTXHIAOT	OUTPUT ATTRIBUTE BYTE HILITE
154	(9A)	HEX	1	PTXHIIOT	INPUT ATTRIBUTE BYTE HILITE
155	(9B)	HEX	1	PTXHIFLG	FLAGS (OUTPUT,INPUT HILITE) HILITE
		1...		PTXHIOHI	"X'80'" OUTPUT HILITING REQUESTED HILITE
152	(98)	HEX	1	PTXHISF	START FIELD 3270 ORDER HILITE
153	(99)	HEX	1	PTXHIAOT	OUTPUT ATTRIBUTE BYTE HILITE
154	(9A)	HEX	1	PTXHIIOT	INPUT ATTRIBUTE BYTE HILITE
155	(9B)	HEX	1	PTXHIFLG	FLAGS (OUTPUT,INPUT HILITE) HILITE
		1...		PTXHIOHI	"X'80'" OUTPUT HILITING REQUESTED HILITE
		.1..		PTXHIIHI	"X'40'" INPUT HILITING REQUESTED HILITE
156	(9C)	A-ADDRESS	4	PTXHELPQ	ADDRESS OF MESSAGE QUEING RTN
160	(A0)	A-ADDRESS	4	PTXUSRWA	ADDR OF INST EXIT WORK AREA
164	(A4)	SIGNED	4	PTXRSV01	RESERVED
168	(A8)	SIGNED	4	PTXRSV02	RESERVED
172	(AC)	SIGNED	4	PTXRSV03	RESERVED
176	(B0)	SIGNED	4	PTXRSV04	RESERVED
		1.11 .1..		PTXEND	"*" END OF THE PTX
		.11. 11..		PTXLEN	"*-PTX" SET THE LENGTH OF THE PTX

DATA AREA IDENTIFIER

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
184	(B8)	FLOATING	8		
184	(B8)	CHARACTER	8	TSOGLID	

MACRO NAME = APLOPTNS.
DESCRIPTIVE NAME = VSAPL/TSO INSTALLATION OPTIONS.
COPYRIGHT = REFER TO MODULE APLCOIBM.
STATUS = RELEASE 4, MODIFICATION LEVEL 0.
FUNCTION = ALL VARIABLE VSAPL/TSO INSTALLATION OPTIONS
ARE LOCATED HERE FOR EASY REFERENCE.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
192	(C0)	SIGNED	4	APLOPTNS	(DSECT)/ALIGNMENT

THE FOLLOWING VALUES ESTABLISH THE APPROPRIATE
WORKSPACE LIBRARY QUALIFIERS TO BE USED BY
APLYULIB.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
192	(C0)	SIGNED	2	OPTID	SIGNIFICANT LENGTH OF APLID
194	(C2)	CHARACTER	8	APLID	WORKSPACE IDENTIFIER
202	(CA)	SIGNED	2	OPTPQ	SIGNIFICANT LENGTH OF PUBQLFR
204	(CC)	CHARACTER	8	PUBQLFR	PUBLIC LIBRARY IDENTIFIER
212	(D4)	SIGNED	2	OPTLQ	SIGNIFICANT LENGTH OF LIBQLFR
214	(D6)	CHARACTER	8	LIBQLFR	PRIVATE-SHAREABLE LIBRARY INDEX IDENTIFIER

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
THE FOLLOWING VALUES ESTABLISH DEFAULT INFORMATION USED FOR ALLOCATING WORKSPACE DATA SETS.					
224	(E0)	SIGNED	4	OPTBLKSI	DCB=BLKSIZE= VALUE
228	(E4)	CHARACTER	8	LIBUNIT	UNIT= VALUE
236	(EC)	CHARACTER	8	LIBSER	VOL=SER= VALUE
THE FOLLOWING OPTION BITS ARE DEFINED.					
244	(F4)	BITSTRING	1	OPTBITS1	OPTION BITS
		1...		OPTDLTX	"B'10000000'" 1: SHAREABLE LIBRARY OWNERSHIP IS KEPT EVEN WHEN ALL WORKSPACES IN THAT LIBRARY ARE DROPPED. 0: SHAREABLE LIBRARY OWNERSHIP IS DROPPED WHEN ALL WORKSPACES IN THE LIBRARY ARE DROPPED.
		..1.		OPTBCH19	"B'00100000'" 1: WHEN VSAPL/TSO IS BEING EXECUTED IN THE BACKGROUND BATCH, 192 TRANSLATION IS USED FOR APLIN AND APLPRINT. 0: 256 TRANSLATION IS USED.
		...1		OPTMICRO	"B'00010000'" 1: WHEN MICROCODE CHECK IS TO BE PERFORMED TO TEST EXISTENCE OF APL MICROCODE. 0: ASSUME MICROCODE IS NOT AVAILABLE (THE DEFAULT FOR MVS CLASS MACHINES).
		.... 1...		OPTMDY	"B'00001000'" 0: DATE FORMAT DD-MM-YY 1: DATE FORMAT MM/DD/YY
245	(F5)	A-ADDRESS	1	OPTBITS2	OPTION BITS
246	(F6)	A-ADDRESS	1	OPTBITS3	OPTION BITS
247	(F7)	A-ADDRESS	1	OPTBITS4	OPTION BITS
MISCELLANEOUS VALUES.					
248	(F8)	BITSTRING	1	OPTTPUT	TPUT TYPE FOR PROMPTING OUTPUT TPUT=ASIS ==> B'00000001' TPUT=CONTROL ==> B'00000010'
249	(F9)	A-ADDRESS	3	OPTRSV1	RESERVED
252	(FC)	SIGNED	4	OPTFRS	DEFAULT FREESIZE VALUE
256	(100)	SIGNED	4	OPTSMSIZ	DEFAULT GDDM FREESIZE
260	(104)	V-ADDRESS	4	OPTEXIT	"V(APLYUUSR)" INSTALLATION EXIT ROUTINE THIS ADDRESS WILL RESOLVE AT LINKEDIT TIME. IF NO EXIT IS DESIRED, OMIT APLYUUSR.
264	(108)	SIGNED	4	OPTUSR	RESERVED FOR INSTALLATION
280	(118)	SIGNED	4	OPTUSR1	
284	(11C)	SIGNED	4	OPTUSR2	
288	(120)	SIGNED	4	OPTUSR3	
292	(124)	SIGNED	4	OPTUSR4	
296	(128)	SIGNED	4	MINAI	MINIMUM ALT INPUT SIZE

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
300	(12C)	SIGNED	4	STCKLEN	DEFAULT LENGTH OF STACK
304	(130)	A-ADDRESS	4	WSSIZMX	MAX SIZE WS ALLOWED. WORKSPACE.
308	(134)	SIGNED	4	MINWS	MINIMUM ACCEPTABLE WS SIZE.
312	(138)	SIGNED	4	MINSH	MINIMUM SHARED MEM SIZE
316	(13C)	SIGNED	4	DFLTSM	MINIMUM SHARED MEM SIZE.
320	(140)	SIGNED	4	FRSIZMN	AT LEAST 20K FREESPACE
324	(144)	SIGNED	4	MAXDEBUG	ONE BYTE MAXIMUM (8 BITS)
328	(148)	CHARACTER	8	OPTSVPNM	SVP IDENTIFY NAME FOR AP'S
LIST OF RESIDENT AUXILIARY PROCESSORS.					
336	(150)	SIGNED	4	MAINAPS	
		....		MAINAPAD	"0,4" ADDRESS OF THE AP
		.... .1..		MAINAPNM	"MAINAPAD+L'MAINAPAD,8 NAME OF THE AP
		.... 11..		MNAPENT	"MAINAPNM+L'MAINAPNM LIST ENTRY LENGTH
336	(150)	HEX	184	(15)	DEFAULT LIST SPACE
336	(150)	V-ADDRESS	4		"V(APL100)" ADDRESS OF THE AP OR ZERO
340	(154)	CHARACTER	8		NAME OF THE AP
348	(15C)	V-ADDRESS	4		"V(APL101)" ADDRESS OF THE AP OR ZERO
352	(160)	CHARACTER	8		NAME OF THE AP
360	(168)	V-ADDRESS	4		"V(APL102)" ADDRESS OF THE AP OR ZERO
364	(16C)	CHARACTER	8		NAME OF THE AP
372	(174)	V-ADDRESS	4		"V(APL111)" ADDRESS OF THE AP OR ZERO
376	(178)	CHARACTER	8		NAME OF THE AP
384	(180)	V-ADDRESS	4		"V(APL120)" ADDRESS OF THE AP OR ZERO
388	(184)	CHARACTER	8		NAME OF THE AP
396	(18C)	V-ADDRESS	4		"V(APL121)" ADDRESS OF THE AP OR ZERO
400	(190)	CHARACTER	8		NAME OF THE AP
408	(198)	V-ADDRESS	4		"V(APL126)" ADDRESS OF THE AP OR ZERO
412	(19C)	CHARACTER	8		NAME OF THE AP
420	(1A4)	V-ADDRESS	4		"V(APL123)" ADDRESS OF THE AP OR ZERO
424	(1A8)	CHARACTER	8		NAME OF THE AP

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
432	(1B0)	V-ADDRESS	4		"V(APL210)" ADDRESS OF THE AP OR ZERO
436	(1B4)	CHARACTER	8		NAME OF THE AP
444	(1BC)	SIGNED	4		LIST DELIMITER
END OF INSTALLATION OPTIONS					
520	(208)	SIGNED	4	OPTEND	END OF OPTIONS
				OPTLEN	"OPTEND-APLOPTNS" LENGTH OF OPTIONS
PROGRAM MANAGEMENT. MISCELLANEOUS MODULE POINTERS					
520	(208)	A-ADDRESS	4	CMSDAIR	POINTER TO DAIR
EXECUTER SAVE AREA STACK AND OTHER SAVE AREAS					
524	(20C)	SIGNED	4	CMSAVE	SIX SAVE AREAS
				CMSAVEZ	"*" MARKS END OF SAVE AREAS.
956	(3BC)	A-ADDRESS	4	CMSAVEZP	ADDR OF END OF SAVE AREAS.
		.1.. 1...		CMSBMPSV	"18*4" TO BUMP TO NEXT SAVE AREA.
960	(3C0)	A-ADDRESS	4	TERMSAVE(15)	USED BY APLYUTIO
1020	(3FC)	A-ADDRESS	4	STCKSAVE(3)	USED BY APL101
1032	(408)	A-ADDRESS	4	SCANSAVE(16)	USED BY APLYUSCN
THE FOLLOWING IS THE ECB TO BE USED FOR GDDX SUBTASK CONTROL. IT IS MANAGED COMPLETELY BY GDDX.					
1096	(448)	SIGNED	4	TSOGYTBP	GDDX SUBTASK CONTROL BLOCK
ABEND EXIT WORK AREA, INCLUDES 16 WORD REG SAVE AREA. THIS AREA USED FOR SAVE AREA FOR ALL ABENDING TASKS.					
1100	(44C)	SIGNED	4	CMSXBND	ENSURE ON WORD BOUNDARY
1100	(44C)	HEX	1	CMSBTYPE	ABEND TYPE CODE
		1...		CMSBTPRG	"X'80'" PROGRAM CHECK
		.1..		CMSBTSYS	"X'40'" SYSTEM ABEND
1101	(44D)	HEX	3		RESERVED
1104	(450)	SIGNED	4	CMSBCODE	SYSTEM PROVIDED ABEND CODE
THE FOLLOWING TWO FIELDS ONLY VALID WITH PROGRAM CHECKS					
1108	(454)	SIGNED	4	CMSBPSW(2)	PSW AT POINT OF ABEND
1116	(45C)	SIGNED	4	CMSBREGS(16)	REGISTERS AT POINT OF ABEND
STORAGE MANAGEMENT. KEEP ADDRESS AND LENGTH OF AREA WE GOT IN USER PROGRAM AREA FOR WORKSPACE, SHARED MEM AND AP WORK AREAS. WE USE THESE TO FREE THE SPACE AT YYOFF.					
1180	(49C)	SIGNED	4	CMSFRADR	ADDRESS OF WS, ETC. AREA.

TSOGL (TSO, XSYS, AP) continued

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1184	(4A0)	SIGNED	4	CMSFRSIZ	LENGTH OF AREA, IN BYTES.
				CMSGIVBK	"32*1024" GIVE BACK AT LEAST THIS MUCH SPACE TO TSO.
				CMSAPWKL	"512" EACH AP GETS THIS MUCH CORE (IN BYTES) TO USE AS A WORK AREA.
1188	(4A4)	A-ADDRESS	4	TSOLOADL	ADDRESS OF LOADLIB DCB LOADMD
ACTIVE WORK SPACE IDENTIFICATION					
1192	(4A8)	A-ADDRESS	4	CMSWSADR	ADDRESS OF INCORE WORKSPACE
NOTE: STARTING WITH RELEASE 4, PTXWSM MUST ALSO ALWAYS CONTAIN THE ADDRESS OF THE ACTIVE WORKSPACE.					
1196	(4AC)	SIGNED	4	CMSMAXWS	MAXIMUM ALLOWED WS SIZE. (SEE PTHWSLEN FOR CURR SIZE)
				CMSMINWS	"WSMMINWS" MINIMUM ALLOWED WS SIZE. SEE APLWSM
ACTIVE WORKSPACE ID.					
1200	(4B0)	SIGNED	4	CMSALIB	LIB NUMBER.
1204	(4B4)	CHARACTER	12	CMSANAM	WS NAME (Z-CODES).
1216	(4C0)	CHARACTER	8	CMSAPAS	PASSWORD FROM LAST)SAVE OR)LOAD.
				CMSAWSID	"CMSALIB,*-CMSALIB" ACTIVE WORKSPACE IDENTIFICATION
1224	(4C8)	HEX	24	CMSAVACT	)COPY OR)WSID. ALSO FOR)SAVE
1248	(4E0)	FLOATING	8	TSOWSTIM	TIME WS WAS SAVED. WSTIME
1256	(4E8)	FLOATING	8	TSOWSUSR	USERID THAT SAVED WS WSTIME
SHARED VARIABLES.					
1264	(4F0)	BITSTRING	1	CMSSHVFL	SHARED VARIABLE FLAGS.
		1...		SHVAVAIL	"BIT0" =1 IF SHARED VARIABLES CAN BE USED DURING THIS SESSION.
		.... 1...		SHVRPEAT	"BIT4" IF =1, WE REPEAT A REFERENCE OR OFFER REQUEST ONCE TO PREVENT FALSE RESULTS WITH CERTAIN DISTRIBUTED AUX. PROCESSORS. IF =0, REQUEST HAS NOT BEEN REPEATED AND MAY HAVE TO BE FOR CERTAIN RETURN/REASON CODES.
THIS IS THE ECB LIST INFORMATION THAT WE PASS TO THE SVP WHEN WE DO A SHARED VARIABLE WAIT.					
1268	(4F4)	SIGNED	4	CMSECBL	FOLLOWING TWO WORDS ARE ECB LIST.
1268	(4F4)	A-ADDRESS	4	CMSECBL1	ADDR OF ECB.
1272	(4F8)	SIGNED	4	CMSECBL2	WORD OF X'FF' TO MARK END.
1276	(4FC)	SIGNED	4	CMSVPECB	ECB FOR SH VAR WAIT.
1280	(500)	A-ADDRESS	4	CMSSSMAD	ADDR OF SHARED STORAGE MANAGER.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
SHARED VARIABLE INFORMATION THAT IS PASSED TO THE SVP AT APL STARTUP TO INITIALIZE THE SHARED VARIABLE FACILITY.					
1284	(504)	SIGNED	4	CMSSVPIN	THE FOLLOWING 3 WORDS MUST BE CONTIGUOUS.
1284	(504)	SIGNED	4	CMSNUMAP	NUMBER OF AP'S LOADED.
1288	(508)	SIGNED	4	CMSSMSIZ	SIZE OF SHARED MEMORY.
1292	(50C)	A-ADDRESS	4	CMSSMADR	THE ADDRESS OF SHARED MEMORY
1296	(510)	SIGNED	4	CMSSMSZ2	SIZE OF 2ND SHARED MEMORY
1300	(514)	A-ADDRESS	4	CMSSMAD2	ADDRESS OF 2ND SHARED MEMORY
YYDELAY AND YYRWAIT CONTROL DATA					
1304	(518)	SIGNED	4	CMSECB	THE WAIT ECB.
1308	(51C)	BITSTRING	1	CMSWAITF	SHOWS WHY WE ARE WAITING ON CMSECB.
		1...		WAITRPLY	"BIT0" WAITING FOR ATTENTION TO UNLOCK KEYBOARD AFTER SENDING MESSAGE.
		.1..		WAITIMER	"BIT1" WAITING FOR ATTENTION OR TIMER POP FOR YYDELAY.
		..1.		CMSTIMEP	"BIT2" INSPECTED BY YYDELAY AFTER FALLING OUT OF WAIT MACRO. =1 IF TIMER EXIT POSTED ECB. =0 IF ATTN EXIT POSTED ECB.
		...1		CMSVWAIT	"BIT3" WAITING FOR DOUBLE ATTN TO BREAK SHARED VARIABLE DEADLOCK.
		.... 1...		TSOCMDAT	"BIT4" TSO COMMAND ACTIVE UNDER APL100
1309	(51D)	A-ADDRESS	3	TSOCTCB	ADDRESS OF COMMAND TCB
				CMSMINDL	"1000000" MINIMUM WAIT TIME FOR YYDELAY, IN MICROSECONDS.
QUAD-AI DATA					
1312	(520)	FLOATING	8	CMSTRTUP	TIME OF DAY THAT APL WAS STARTED, IN APL STANDARD TIME FORMAT.
1320	(528)	FLOATING	8	CMSCPUST	STARTING CPU TIME FOR APL MVSCPU SESSION, IN MILLISECONDS MVSCPU
1328	(530)	FLOATING	8	CMSHOLDT	HOLD AREA FOR SAVING CPU TIME WHEN INTERP IS DISPATCHED OR TIME OF DAY WHEN KEYBOARD IS UNLOCKED.
MISCELLANEOUS.					
1336	(538)	FLOATING	8		ALIGNMENT
1336	(538)	CHARACTER	16	CMSPSDT	MM/DD/YYHH:MM:SS (EXACTLY).
1336	(538)	CHARACTER	8	CMSPDATE	PSEUDO DATE.
1344	(540)	CHARACTER	8	CMSPTIME	PSEUDO TIME.
1352	(548)	FLOATING	8	CMSPACK	CONVERTS LIB NUMBERS TO EBCD

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1360	(550)	HEX	1	(31)	WORK AREA
		..1. .111		CMSPACKL	"*-CMSPACK" LENGTH OF WORK AREA
1392	(570)	A-ADDRESS	4	LDSNSAVE	SAVE AREA FOR LIBDSN ROUTIN
1396	(574)	SIGNED	4	USRACTNO	USER SUPPLIED ACCOUNT NUMBER
1400	(578)	SIGNED	4	CMSOLDTE	PREVIOUSLY RESOLVED DATE
1404	(57C)	A-ADDRESS	1	CMSMONTH(12)	
MISCELLANEOUS PROGRAM CONTROL AND STATE FLAGS					
1416	(588)	BITSTRING	1	CMSPGMFL	PROGRAM MANAGEMENT FLAGS.
		.1..		CMSINSVP	"BIT1" =1 IF PROGRAM CONTROL HAS BEEN GIVEN TO THE SVP (WHICH IN TURN GIVES CONTROL TO AN AP).
		..1.		CMSCOPER	"BIT2" SYSTEM ERROR OCCURRED WHILE IN COPY STATUS. SET BY YYSYSER, CHECKED BY YYCOPZ.
1417	(589)	BITSTRING	1	CMSPGMF2	FLAG BYTE
		.1..		CMSXEQTR	"BIT1" CONTROL HAS PASSED TO XQTR
		..1.		CMSABND2	"BIT2" ABEND RECURSION HAS OCCURRED
		...1		CMSSTAE	"BIT3" AN (E)STAE HAS BEEN ISSUED
		.... 1...		CMSNSTAE	"BIT4" NO (E)STAE IS TO BE ISSUED
		.... .1..		CMSABEND	"BIT5" A SERIOUS ABEND HAS OCCURRED
		.... ..1.		CMSCANCL	"BIT6" VSAPL HAS BEEN CANCELLED
		1		CMSABORT	"BIT7" EXECUTION ABORTED BY APLYUINI
1418	(58A)	BITSTRING	1	CMSPGMF3	FLAG BYTE
		.1..		CMSWSZVN	"BIT1" WORKSPACE OPERAND GIVEN ON THE INVOCATION COMMAND
		..1.		CMSNOAUT	"BIT2" SUPPRESS LOAD OF CONTINUE
		...1		CMSCONTX	"BIT3" CONTINUE WORKSPACE DOES EXIST
		.... 1...		CMSAPLSM	"BIT4" APLSM ON SPECIFIED OR ASSUMED
1419	(58B)	BITSTRING	1	OSSYSTYP	COPY OF CVTDCB
TSO PROFILE DATA					
1420	(58C)	CHARACTER	1	TSOLDCC	OLD LINE DELETE CHAR
1421	(58D)	CHARACTER	1	TSOCDCC	OLD CHARACTER DELETE CHAR
1422	(58E)	CHARACTER	8	TSOTRAN	STTRAN TRANSLATE NAME
ERROR RECOVERY DATA SPIE DATA					
1432	(598)	A-ADDRESS	4	OLDPICA	POINTER TO PRE-APL PICA
1436	(59C)		0		ALIGN PICA TO FULLWORD BOUNDARY
1436	(59C)	BITSTRING	1	OURPICA	PROGRAM MASKS
1437	(59D)	A-ADDRESS	3		EXIT ROUTINE ADDRESS S
1440	(5A0)	BITSTRING	2		THE INTERRUPT MASK BYTES 1 AND 2
		.... .11.		OURPICAL	"*-OURPICA" LENGTH OF OUR PICA

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION

STAE/ESTAE DATA

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1444	(5A4)	SIGNED	4		
1444	(5A4)	A-ADDRESS	1	OURESTAE	FLAGS FOR TCB,PURGE,ASYNCH
1445	(5A5)	A-ADDRESS	3		STAE EXIT ROUTINE ADDR.
1448	(5A8)	A-ADDRESS	4		STAE EXIT PARM. LIST ADDR.
1452	(5AC)	A-ADDRESS	4		TCB NOT SPECIFIED
1456	(5B0)	A-ADDRESS	1		FLAGS
1457	(5B1)	A-ADDRESS	3		RESERVED
		...1		TSOESTAL	"*-OURESTAE" LENGTH OF THE ESTAE AREA
1460	(5B4)	A-ADDRESS	4	STAEREGS(5)	CRITICAL STAE EXIT REGS (R13 > R1)
1480	(5C8)	A-ADDRESS	4	RTRYREGS(7)	CRITICAL STAE RETRY REGS (R9 > R14)

CMSDMPNO CONTAINS THE DUMP NUMBER THAT IS PUT INTO SYSTEM ERROR MESSAGES BY THE INTERPRETER. IT IS RETURNED BY

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1508	(5E4)	SIGNED	4	CMSDMPNO	SEE ABOVE.

CONSTRUCTION AREA FOR TSO AND OTHER MVS PARAMETER ADDRESSES OF IMPORTANT TSO CONTROL BLOCKS. SEE 'IKJCPPL' FOR A DISCRIPTION OF ITS CONTENTS.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1512	(5E8)	SIGNED	4		
1512	(5E8)	HEX	1	CPPLSTG	SEE IKJCPPL

THE DAPL IS BUILT BY A COMMAND PROCESSOR AS A PARAMETER LIST FOR DYNAMIC ALLOCATION (DAIR). SEE 'IKJDAPL' FOR A DISCRIPTION OF ITS CONTENTS.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1528	(5F8)	SIGNED	4		
1528	(5F8)	HEX	20	DAPLSTG	SEE IKJDAPL
1548	(60C)	SIGNED	4	DFRC	DAIR FUNCTION CODE SAVE AREA

THE FOLLOWING DATA AREA CONTAINS ROOM FOR ANY OF SEVERAL OS OR TSO INTERFACE CONTROL BLOCKS. IT MAY BE USED BY ANY EXECUTOR COMPONENT INTERFACING WITH TSO OR THE OPERATING SYSTEM, BUT CARE SHOULD BE TAKEN TO AVOID ITS USE WHEN PASSING CONTROL TO ANOTHER EXECUTOR COMPONENT WHICH MIGHT REUSE IT BEFORE RETURNING CONTROL. ANY CONTROL BLOCK WHICH MAY OVERLAY THIS AREA SHOULD BE EXPLICITLY DECLARED MAPPED BY THE 'IKJDAPXX' MAPPING MACROS WHERE 'XX' IS REPLACED WITH 04, 08, 0C, 18, AND 2C.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1552	(610)	SIGNED	4	DAPBS	
1552	(610)	HEX	2	DAPBCD	DAIR CALL CODE
1554	(612)	BITSTRING	1	DAPBFLG	FUNCTIONS PERFORMED IF RC=0
1555	(613)	HEX	1		RESERVED
1556	(614)	SIGNED	2	DAPBDARC	DYNAM RC WHEN DAIR RC=8
1558	(616)	SIGNED	2	DAPBCTRC	CATLG RC WHEN DAIR RC=4
1552	(610)	HEX	20		SEE IKJDAP00
1552	(610)	HEX	16		SEE IKJDAP04

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1552	(610)	HEX	84		SEE IKJDAP08
1552	(610)	HEX	16		SEE IKJDAP14
1552	(610)	HEX	40		SEE IKJDAP18
1552	(610)	HEX	16		SEE IKJDAP2C
1552	(610)	SIGNED	4	S26PLIST	ALIGN ON FULL WORD
1552	(610)	A-ADDRESS	1		THREE BYTES OF FLAGS
1553	(611)	A-ADDRESS	1		INDICATING THE FUNC-
1554	(612)	A-ADDRESS	1		TION TO BE PERFORMED
1555	(613)	A-ADDRESS	1		NO OPTION THREE
1556	(614)	A-ADDRESS	4		PARAMETER TWO
1560	(618)	A-ADDRESS	4		PARAM. THREE OMMITTED
1564	(61C)	A-ADDRESS	4		PARAMETER FOUR
1552	(610)	SIGNED	4	APATTACH	
1552	(610)	A-ADDRESS	4		POINTER TO SYMB NAME
1556	(614)	A-ADDRESS	1		
1557	(615)	A-ADDRESS	3		DCB ADDRESS LCS1
1560	(618)	A-ADDRESS	1		FLAGS
1561	(619)	A-ADDRESS	3		ECB ADDRESS
1564	(61C)	A-ADDRESS	4		GSPL OR GSPV
1568	(620)	A-ADDRESS	4		SHSPV VALUE
1572	(624)	A-ADDRESS	1		
1573	(625)	A-ADDRESS	3		EXIT ROUT. ADDRESS RORI
1576	(628)	A-ADDRESS	2		DPMOD VALUE
1578	(62A)	A-ADDRESS	1		LPMOD VALUE
1579	(62B)	A-ADDRESS	1		
1580	(62C)	A-ADDRESS	4	(2)	EP NAME SPACE
1588	(634)	A-ADDRESS	1		NO LSQA
1589	(635)	A-ADDRESS	3		
1592	(638)	A-ADDRESS	1		NO TID
1593	(639)	A-ADDRESS	3		STAI/ESTAI PARAMETER LIST
1596	(63C)	A-ADDRESS	1		STAI/ESTAI FLAGS
1597	(63D)	A-ADDRESS	3		STAI/ESTAI EXIT ROUTINE ADDR
1600	(640)	A-ADDRESS	4		TASKLIB.
1604	(644)	A-ADDRESS	4		FLAGS AND PARM LIST LENGTH
1608	(648)	A-ADDRESS	4		NO NSHSPV OR NSHSPL PARM
		..11 11..		APATCHLN	"*-APATTACH" PLIST LENGTH
1552	(610)	A-ADDRESS	4	TSOTRTBL	STTRAN PARM BLOCK
1556	(614)	A-ADDRESS	4	TSOTRNM	STTRAN TABLE NAME

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1560	(618)	A-ADDRESS	4	TSOTROPT	STTRAN LIST OPTION LIST
1564	(61C)	CHARACTER	8	TRNNAME	TRANSLATE NAME
1572	(624)	A-ADDRESS	4	TSOVTBL	A(TRANSLATE NAME)
1576	(628)	A-ADDRESS	4	TSOVTSB	A(TSB BUFFER = CMSBUFF)
LIBRARY MANAGEMENT DATA BSAM CONTROL BLOCKS					
		.... .1.1		LIBNCP	"5" # ICB'S FOR OPTCD=C
1636	(664)	SIGNED	4	LIBOPEN	ALIGN LIST TO FULLWORD
1636	(664)	A-ADDRESS	1		OPTION BYTE
1637	(665)	A-ADDRESS	3		DCB ADDRESS
1640	(668)	SIGNED	4	LIBDCB	ORIGIN ON WORD BOUNDARY DIRECT ACCESS DEVICE INTERFACE
1640	(668)	BITSTRING	16		FDAD,DVTBL
1656	(678)	A-ADDRESS	4		KEYLE,DEVT,TRBAL COMMON ACCESS METHOD INTERFACE
1660	(67C)	A-ADDRESS	1		BUFNO
1661	(67D)	A-ADDRESS	3		BUFCB
1664	(680)	A-ADDRESS	2		BUFL
1666	(682)	BITSTRING	2		DSORG
1668	(684)	A-ADDRESS	4		IOBAD FOUNDATION EXTENSION
1672	(688)	BITSTRING	1		BFTEK,BFLN,HIARCHY
1673	(689)	A-ADDRESS	3		EODAD
1676	(68C)	BITSTRING	1		RECFM
1677	(68D)	A-ADDRESS	3		EXLST FOUNDATION BLOCK
1680	(690)	CHARACTER	8		DDNAME
1688	(698)	BITSTRING	1		OFLGS
1689	(699)	BITSTRING	1		IFLG
1690	(69A)	BITSTRING	2		MACR BSAM-BPAM-QSAM INTERFACE
1692	(69C)	BITSTRING	1		RER1
1693	(69D)	A-ADDRESS	3		CHECK, GERR, PERR
1696	(6A0)	A-ADDRESS	4		SYNAD
1700	(6A4)	SIGNED	2		CIND1, CIND2
1702	(6A6)	A-ADDRESS	2		BLKSIZE
1704	(6A8)	SIGNED	4		WCPO, WCPL, OFFSR, OFFSW
1708	(6AC)	A-ADDRESS	4		IOBA
1712	(6B0)	A-ADDRESS	1		NCP
1713	(6B1)	A-ADDRESS	3		EOBR, EOBAD BSAM-BPAM INTERFACE
1716	(6B4)	A-ADDRESS	4		EOBW
1720	(6B8)	SIGNED	2		DIRCT
1722	(6BA)	A-ADDRESS	2		LRECL

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1724	(6BC)	A-ADDRESS	4		CNTRL, NOTE, POINT
		.1.1 1...		LIBDCBL	"*-LIBDCB" DCB LENGTH
1728	(6C0)	SIGNED	4	LIBDECB	EVENT CONTROL BLOCK
1732	(6C4)	HEX	1		TYPE FIELD
1733	(6C5)	HEX	1		TYPE FIELD
1734	(6C6)	A-ADDRESS	2		LENGTH
1736	(6C8)	A-ADDRESS	4		DCB ADDRESS
1740	(6CC)	A-ADDRESS	4		AREA ADDRESS
1744	(6D0)	A-ADDRESS	4		RECORD POINTER WORD
1748	(6D4)	A-ADDRESS	4	LIBDECBN	DECB RING CHAIN FIELD
		...1 1...		LIBDECBL	"*-LIBDECB" RING ENTRY LENGTH
1752	(6D8)	HEX	4		REMAINING RING ENTRIES

MISCELLANEOUS OTHER LIBRARY MANAGEMENT DATA
DATE/TIME WORKSPACE WAS SAVED, TO BE RETURNED TO INTERP.
VIA PDSPASS.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1848	(738)	HEX	8	CMSAVDAT	TIME/DATE IN FLOATING PT.
1856	(740)	HEX	4	LIB#LIST	POINTER TO AN INSTALLATION PROVIDED LIST OF LIBRARY NUMBERS WHICH THIS USER IS PERMITTED TO CREATE. NOTE, ACCESS IS NOT HEREBY LIMITED, ONLY CREATION.
		.... 1...		LIB#LEN	"8" LENGTH OF EACH ENTRY OF THE LIB#LIST OF LIBRARIES
1860	(744)	HEX	4	MAXLIBNO	MAX LEGAL LIBRARY NUMBER (THIS VALUE IS DEPENDENT ON THE LENGTH OF APLID)
1864	(748)	HEX	4	LIBDATLN	BYTE LENGTH OF DATA TO BE WRITTEN TO DISK BY EITHER SCSAVE OR SCCOPA.
1868	(74C)	HEX	4	CNTALIB	CONTINUE WORKSPACE'S ORIGINAL LIBRARY NUMBER
1872	(750)	HEX	12	CNTANAM	CONTINUE WORKSPACE'S ORIGINAL NAME (ZCODES)
1884	(75C)	HEX	2		| DSN SIGNIFICANT LENGTH
1886	(75E)	HEX	44	LIBWSDSN	V DSN
1930	(78A)	HEX	4	LIBFNCTN	LIBRARY MGMT FUNCTION CODE
		1		LIBDROP	"1" SCDROP
		.... ..1.		LIBLIB	"2" SCLIB
		11		LIBLOAD	"3" SCLOAD
		.... .1..		LIBSAVE	"4" SCSAVE
		.... .1.1		LIBCONT	"5" SCCONT
		.... .11.		LIBCOPY	"6" SCCOPA, SCCOPO, SCCOPI, OR SCCOPZ

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1931	(78B)	HEX	4	LIBLOCAL	LIBRARY MANAGEMENT FLAG
		1...		LIBOURS	"BIT0" THE CURRENT LIBRARY IS OWNED BY THE CURRENT USER. IT MAY STILL BE EITHER PROJECT OR PRIVATE.
		.1..		LIBSHR	"BIT1" THE CURRENT LIBRARY IS PROJECT. IT MAY OR MAY NOT BE OURS.
		..1.		LIBPUB	"BIT2" THE CURRENT LIBRARY IS PUBLIC. I.E. ITS NUMBER IS LESS THAN 1000. IT MAY OR MAY NOT BE OURS. IT MUST BE SHARABLE.
		...1		LIBCONTU	"BIT3" THE WORKSPACE BEING REFERENCED IS THE PRIVATE CONTINUE WORKSPACE.
		.... 1...		LIBNWLIB	"BIT4")SAVE IS CAUSING A NEW PROJECT LIBRARY TO BE DEFINED.
		.... .1..		LIBNEWDS	"BIT5")SAVE ONLY WORKSPACE DATA SET IS NEW
		.... .1..		LIBLBERR	"BIT5")LIB ONLY A CATALOG MANAGEMENT ERROR HAS OCCURRED. IF THIS IS THE INTERPRETER'S LAST CALL TO SCLIB FOR THIS)LIB COMMAND, THEN QUEUE UP A WARNING MESSAGE FOR THE USER.
		.... .1..		LIBSORSD	"BIT5")COPY ONLY COPA HAS SAVED THE SINK AND LOADED THE SOURCE. I.E. THE SINK IS DESTROYED IN CORE BUT PRESERVED ON DISK.
		.... ..1.		LIBPWDKN	"BIT6")SAVE ONLY THE CURRENT PASSWORD FOR THIS WORKSPACE IS KNOWN FROM THE PREVIOUS)LOAD COMMAND.
		1		LIBAUTH	"BIT7" ALL THE USER IS NOT AUTHORIZED TO)SAVE OR)DROP.
1932	(78C)	HEX	4	LIBGLOBL	GLOBAL LIB MGMNT FLAG
		1...		LIBLBOFL	"BIT0")LIB ONLY WSMBUFF HAS OVERFLOWED DURING PROCESSING OF THE)LIB COMMAND. AS A RESULT, THE INTERPRETER WILL RE-CALL SCLIB FOR THE ADDITIONAL WORKSPACE NAMES.
		.1..		LIBLDCPY	"BIT1")COPY ONLY THIS SIGNALS)LOAD THAT IT IS BEING CALLED BY)COPY INITIALIZATION.
		..1.		LIBABEND	"BIT2" DCB ABEND THE DCB ABEND EXIT HAS BEEN RACF TAKEN, AND AN ERROR NOTED. RACF THE DCB WILL BE CLOSED. RACF
1936	(790)	HEX	0		ALIGNMENT
1936	(790)	HEX	4	WSHSAVE	WS HEADER BUFFER SAVE AREA

THE FOLLOWING DATA MUST BE CONTIGUOUS SINCE IT IS INITIALIZE AS A SINGLE BLOCK

2024	(7E8)	HEX	4	LIBDATAA	START OF BLOCK

DEFAULT LIBRARY NUMBER PERMISSION LIST.

2024	(7E8)	HEX	4	DFLTLIBN	USED IFF APLYUUSR ¬EXIST

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
CATALOG MANAGEMENT LISTS USED TO MANAGE LIBRARY IDENTIFICATION CATALOG ENTRIES					
2032	(7F0)	HEX	0	APLBLDX	ALIGN ON FULL WORD
2032	(7F0)	HEX	1		THREE BYTES OF FLAGS
2033	(7F1)	HEX	1		INDICATING THE FUNC-
2034	(7F2)	HEX	1		TION TO BE PERFORMED
2035	(7F3)	HEX	1		NO OPTION THREE
2036	(7F4)	HEX	4		PARAMETER TWO
2040	(7F8)	HEX	4		PARAM. THREE OMMITTED
2044	(7FC)	HEX	4		PARAMETER FOUR
2048	(800)	HEX	0	APLDLTX	ALIGN ON FULL WORD
2048	(800)	HEX	1		THREE BYTES OF FLAGS
2049	(801)	HEX	1		INDICATING THE FUNC-
2050	(802)	HEX	1		TION TO BE PERFORMED
2051	(803)	HEX	1		NO OPTION THREE
2052	(804)	HEX	4		PARAMETER TWO
2056	(808)	HEX	4		PARAM. THREE OMMITTED
WORKSPACE DATA SET NAME CONSTRUCTION DATA					
2060	(80C)	HEX	2	LIBPREFL	SIGNIFICANT L'LIBPREFX
2062	(80E)	HEX	8	LIBPREFX	1ST LEVEL DSNAME QUALIFIER (USUALLY THE USERID)
2070	(816)	HEX	4	LIBUID	USERID
				LIBDATAZ	"*" END MARKER
		..11 .1.1		LIBDATAL	"LIBDATAZ-LIBDATAA" AREA LENGTH
END OF CONTIGUOUS DATA)COPY MANAGEMENT DATA BSAM CONTROL BLOCKS					
2080	(820)	HEX	0	WORKOPEN	ALIGN LIST TO FULLWORD
2080	(820)	HEX	1		OPTION BYTE
2081	(821)	HEX	4		DCB ADDRESS
2084	(824)	HEX	0	WORKDCB	ORIGIN ON WORD BOUNDARY DIRECT ACCESS DEVICE INTERFACE
2084	(824)	HEX	16		FDAD,DVTBL
2100	(834)	HEX	4		KEYLE,DEVT,TRBAL COMMON ACCESS METHOD INTERFACE
2104	(838)	HEX	1		BUFNO
2105	(839)	HEX	3		BUFCB
2108	(83C)	HEX	2		BUFL
2110	(83E)	HEX	2		DSORG
2112	(840)	HEX	4		IOBAD FOUNDATION EXTENSION
2116	(844)	HEX	1		BFTEK,BFLN,HIARCHY
2117	(845)	HEX	3		EODAD

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
2120	(848)	HEX	1		RECFM
2121	(849)	HEX	3		EXLST FOUNDATION BLOCK
2124	(84C)	HEX	8		DDNAME
2132	(854)	HEX	1		OFLGS
2133	(855)	HEX	1		IFLG
2134	(856)	HEX	2		MACR BSAM-BPAM-QSAM INTERFACE
2136	(858)	HEX	1		RER1
2137	(859)	HEX	3		CHECK, GERR, PERR
2140	(85C)	HEX	4		SYNAD
2144	(860)	HEX	2		CIND1, CIND2
2146	(862)	HEX	2		BLKSIZE
2148	(864)	HEX	4		WCPO, WCPL, OFFSR, OFFSW
2152	(868)	HEX	4		IOBA
2156	(86C)	HEX	1		NCP
2157	(86D)	HEX	3		EOBR, EOBAD BSAM-BPAM INTERFACE
2160	(870)	HEX	4		EOBW
2164	(874)	HEX	2		DIRCT
2166	(876)	HEX	2		LRECL
2168	(878)	HEX	4		CNTRL, NOTE, POINT
		.1.1 1...		WORKDCBL	"*-WORKDCB"
)COPY SINK CONTROL DATA					
2172	(87C)	HEX	4	CPYDATA	DATA START
2172	(87C)	HEX	4	CPYGETMN	ADDR. OF LENGTH LIST
2176	(880)	HEX	4		ADDR. OF ADDR. LIST
2180	(884)	HEX	1		MODE AND OPTION FLAGS
2181	(885)	HEX	3		SUBPOOL VALUE
2184	(888)	HEX	8	CPYGMQTY	VARIABLE GETMAIN LIMITS
2192	(890)	HEX	4	CPYMAXL	MAX NEEDED SAVE SPACE SIZE
2196	(894)	HEX	4	CPYRSDUL	MAX NEEDED SAVE DATA SET SZ
2200	(898)	HEX	4	CPYHEADL	SINK WS HEAD SAVE SIZE
2204	(89C)	HEX	4	CPYTAILL	SINK WS TAIL SAVE SIZE
2208	(8A0)	HEX	8	CPYSLOT1	| 1ST SAVE SLOT DESCRIPTER
2216	(8A8)	HEX	8	CPYSLOT2	| 2ND SAVE SLOT DESCRIPTER
2224	(8B0)	HEX	4	CPYSLOT3	| 3RD SAVE SLOT DESCRIPTER
		11		CPYSLOT#	"(*-CPYSLOT1)/8" V SAVE SLOT COUNT
2232	(8B8)	HEX	0		| ALIGNMENT
2232	(8B8)	HEX	28	CPYAVAIL	AVAIL SPACE DESCRIPTERS

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
2260	(8D4)	HEX	4	CPYNXAVL	> NEXT AVAIL DESCRIPTER
2264	(8D8)	HEX	4	CPYSAVEA	MISC SAVE AREA
				CPYDATZ	"*" DATA END
		.11.		CPYDATL	"CPYDATZ-CPYDATA" DATA LENGTH
MISCELLANEOUS OTHER)COPY DATA					
2268	(8DC)	HEX	4	CMSAVSIZ	SAVE SIZE OF ACTIVE (SINK) WS HERE WHILE SOURCE IS LOADED.
2272	(8E0)	HEX	4	COPARET	RETURN ADDRESS TO THE INTERPRETER AFTER CALL TO YYCOPA. THIS IS ALSO USED AS THE RETURN ADDRESS AFTER THE CALL TO YYCOPZ THAT CONVERTS FROM COPY-SOURCE MODE TO COPY-SINK MODE. IT IS ALSO USED FOR ALL NON-SYSTEM-ERROR TYPE ERROR RECOVERY RETURNS.
2276	(8E4)	HEX	4	CMSICTR	COUNT OF RECORDS READ FROM COPYDATA FILE.
2280	(8E8)	HEX	4	CMSOCTR	COUNT OF RECORDS WRITTEN TO COPYDATA FILE.
TERMINAL MANAGEMENT.					
2284	(8EC)	HEX	1	CMSIDLSW	=ENL IF IDLES REQ'D, ELSE 0.
2285	(8ED)	HEX	1	CMSNLSW	=ENL IF NEW-LINE SEEN, ELSE X'00'. USED FOR WSMPARM2 CHECK ON YYTYO.
2286	(8EE)	HEX	4	CMSFLAGS	TERMINAL MANAGEMENT FLAGS
		1...		CMSSEGZ	"BIT0" SEGMENT IS LAST ONE IN CMSBF
		.1..		TSORFLAG	"BIT1" READING FROM TERMINAL
		..1.		CMSTYOI	"BIT2" TYO CALLED FROM TYOI
		...1		CMSLAST	"BIT3" FINAL TYO OUTPUT
		.... 1...		CMSOUT	"BIT4" O-U-T SIGNALLED ON DISPLAY TERMINAL.
		.... .1..		CMSNLREQ	"BIT5" ON YYTYO FOR DISPLAY TERM, INPUT PARM SAYS NEW-LINE CHAR MUST BE AT END OF OUTPUT.
		.... ..1.		CMSTYOII	"BIT6" TYOI CALLING TYI
		1		CMS4SOUT	"BIT7" OUTPUT THIS MESSAGE IN SPITE OF ATTENTIONS
CMSBUFF POINTERS. USED DURING TYO TO KEEP TRACK OF WHAT HAS BEEN PUT IN CMSBUFF.					
2288	(8F0)	HEX	4	CMSBFLIN	BEGINNING OF CURRENT LINE IN CMSBUFF (TYPEWRITER ONLY)
				CMSHELD	"CMSBFLIN" FOR DISPLAY TERMINALS, CONTAINS LENGTH OF DATA HELD FROM PREVIOUS YYTYO.
2292	(8F4)	HEX	4	CMSBFSEG	BEGINNING OF CURRENT SEGMENT IN CMSBUFF. (FOR TYPEWRITER TERMINAL ONLY.)
				CMSINBUF	"CMSBFSEG" LENGTH OF INPUT BUFFER FOR DISPLAY TERMINAL (=135 IF USING 3270).

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
PARMLISTS FOR RDTERM AND WRTERM MACROS.					
2296	(8F8)	HEX	0	CMSXPLST	WRTERM PLIST USED BY ATTENTION (STAX) EXIT.
2296	(8F8)	HEX	8		
2304	(900)	HEX	1		UNUSED HISTORIC BIT.
2305	(901)	HEX	3	CMSRADDR	OUTPUT ADDRESS.
2308	(904)	HEX	1		BLACK RIBBON.
2309	(905)	HEX	1		LONG WRITE, EDIT=NO.
2310	(906)	HEX	4	CMSRLGTH	OUTPUT LENGTH.
		...1		CMSXPLL	"*-CMSXPLST" LENGTH OF PLIST.
2312	(908)	HEX	0	CMSWPLST	WRTERM PLIST.
2312	(908)	HEX	8		
2320	(910)	HEX	1		UNUSED HISTORIC BIT.
2321	(911)	HEX	3	CMSWADDR	OUTPUT ADDRESS.
2324	(914)	HEX	1		BLACK RIBBON.
2325	(915)	HEX	1		LONG WRITE, EDIT=NO.
2326	(916)	HEX	4	CMSWLGTH	OUTPUT LENGTH.
		...1		CMSWPLL	"*-CMSWPLST" LENGTH WRTERM PLIST.
2328	(918)	HEX	8		
2328	(918)	HEX	0	TSORPLST	RDTERM PLIST.
2336	(920)	HEX	1		UNUSED HISTORIC BIT.
2337	(921)	HEX	3	TSORADDR	INPUT BUFFER ADDRESS.
2340	(924)	HEX	1		ATTREST=NO OPTION.
2341	(925)	HEX	1		UNUSED.
2342	(926)	HEX	4	TSORLGTH	INPUT LENGTH.
		...1		TSORPLL	"*-TSORPLST" LENGTH OF RDTERM PLIST.
TERMINAL DEVICE INFORMATION.					
2344	(928)	HEX	4	TSODTYPE	TERMINAL DEVICE TYPE
		1...		TS03270	"BIT0" 3270 DISPLAY
		.1..		TS03277	"BIT1" 3277 DISPLAY
		..1.		TS03278	"BIT2" 3278 DISPLAY
		...1		TSOAPLFC	"BIT3" APL FEATURED TERMINAL
		.... 1...		TSOBATCH	"BIT4" APL RUNNING IN BACKGROUND
		.... .1..		TSOIDLES	"BIT5" TERMINAL START-STOP W/ IDLES
		.... ..1.		TSOVTAM	"BIT6" LINE IS VTAM
		1		TSOTCAM	"BIT7" LINE IS TCAM
2345	(929)	HEX	1	TS0327CC	327X GRAPHIC ESCAPE X'1D' FOR 3277 X'08' FOR 3278
2346	(92A)	HEX	4	TSODPYFL	FLAGS USED BY APLYUDPY
		1...		TSODPHLD	"BIT0" HELD INPUT IN CMSBUFF2
		.1..		TSODPFUL	"BIT1" BUFFER OVERFLOW ON TGET
		..1.		TSODPECO	"BIT2" ECHO INPUT (MULTILINE)
		...1		TSODSMAV	"BIT3" DISPLAY SESSION MGR AVAILABL R4ADSM
2347	(92B)	HEX	1		RESERVED
2348	(92C)	HEX	4	TSOPFKAD	PFK DEFINITION TABLE ADDR

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
PARMLIST FOR STAX MACRO.					
2352	(930)	HEX	0	CMSTAXPL	
2352	(930)	HEX	4		ADDRESS OF EXIT ROUTINE
2356	(934)	HEX	2		LENGTH OF INPUT BUFFERS
2358	(936)	HEX	2		LENGTH OF OUTPUT BUFFERS
2360	(938)	HEX	4		ADDRESS OF OUTPUT BUFFERS
2364	(93C)	HEX	4		ADDRESS OF INPUT BUFFERS
2368	(940)	HEX	1		REPLACE/NO REPLACE, DEFERRAL IND
2369	(941)	HEX	4		ADDRESS OF USER PARAMETERS
		...1 .1..		CMSTAXL	"*-CMSTAXPL" LENGTH OF STAX PARMLIST.
		....		STXEXIT	"0" OFFSET TO FIELD IN CMSTAXPL CONTAINING ADDR OF STAX EXIT ROUTINE.
2372	(944)	HEX	4	CMSTABS	CURRENT TAB SETTING. (ALL 0 IF NO TABS.)
ADDRESSES OF DEVICE-DEPENDENT SERVICE REQUEST HANDLERS. THE ADDRESSES IN THESE FIELDS DEPEND ON WHETHER THE TERMINAL IS A TYPEWRITER OR A DISPLAY (3270). APLYUINI STORES THE ADDRESSES HERE, APLYUFXI USES THEM.					
2628	(A44)	HEX	0	CMSDDADR	
2628	(A44)	HEX	4	CMSXTYI	ADDRESS OF YYTYI HANDLER. (EITHER SCTYI OR SCDTYI.)
2632	(A48)	HEX	4	CMSXTYO	ADDRESS OF YYTYO HANDLER. (EITHER SCTYO OR SCDTYO.)
2636	(A4C)	HEX	4	CMSXTYOI	ADDRESS OF YYTYOI HANDLER. (EITHER SCTYOI OR SCDTYOI.)
STACK PROCESSING (APL101) SUPPORT.					
2640	(A50)	HEX	4	STCKPURG	> STACK PURGE (IF ANY)
2644	(A54)	HEX	4	STCKPOP	> STACK POP (IF ANY)
				LENSTACK	"512" LENGTH OF STACK
2648	(A58)	HEX	4	NEXTITEM	(STACKTXT+LENSTACK) START OF USED PART
2652	(A5C)	HEX	4	STACKBEG	(STACKTXT) CONSTANT
2656	(A60)	HEX	4	STACKEND	(STACKTXT+LENSTACK) CONSTANT
STCKLEN DS F DEFAULT SIZE IS IN APLOPTNS AISIZE					
2660	(A64)	HEX	4	STCKSTAT	STATUS OF STACK
		1...		PURGING	"X'80'" ITS BEING PURGED
		.1..		NOFENCE	"X'40'"
		..1.		STCKDATA	"X'20'" LAST DATA FROM STACK
2661	(A65)	HEX	3	APL101WK	STACK FENCE ADDRESS
2664	(A68)	HEX	4	TERMTRAC	> TERMINIAL IO TRACE

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
TERMINAL I/O TESTING AND TRANSLATING TABLES					
2668	(A6C)	HEX	256	CMSXOUT	OUTPUT TRANSLATE FOR DEVICE NTO--
2924	(B6C)	HEX	256	CMSXIN	INPUT TRANSLATE FOR DEVICE NTO--
3180	(C6C)	HEX	4	CMSYOUT	A(OUTPUT) (TRT FOR ESCAPE)
3184	(C70)	HEX	4	CMSYIN	A(INPUT) (TR AFTER ESCAPE)
3188	(C74)	HEX	4	TSOTRZE	ZCODE TO EBCDIC TRANSLATE NTO--
3192	(C78)	HEX	4	TSOTREZ	EBCDIC TO ZCODE TRANSLATE NTO--
3196	(C7C)	HEX	1	TSOQTT	TERMINAL TYPE
3197	(C7D)	HEX	3		RESERVED
3200	(C80)	HEX	2	TSODH	TERMINAL HEIGHT
3202	(C82)	HEX	4	TSODW	TERMINAL WIDTH
		.... .1..		TCAMFUDG	"4" ATTRIB OVERHEAD
3204	(C84)	HEX	4	CMSFSAWA	FS EDIT WORK AREA
				CMSFSFLG	"CMSFSAWA" FIRST BYTE IS FLAG.
		1...		FSEDINIT	"X'80'" FS EDITOR INITIALIZED.
		.1..		FSMSGFUL	"X'40'" FS MSG AREA IS FULL.
		..1.		FSMRFLSH	"X'20'" FS MSG AREA HAS BEEN FLUSHED.
		...1		FSMSCSTK	"X'10'" SCREEN 'STACK' IN USE.
		.... 1...		FSEDOPEN	"X'08'" FS EDITOR CURRENTLY OPEN.
		.... .1..		FSMODE	"X'04'" CURRENT SCREEN IS FULL SCREEN
				FSWALEN	"X'4000'" MAXIMUM THAT MIGHT BE NEEDED
3208	(C88)	HEX	4	CMSFSMWK	FULL SCREEN MGR WORK AREA
3212	(C8C)	HEX	4	CMSFSGWK	FULL SCREEN GRAF DRIVER AREA
3216	(C90)	HEX	0	GTTERM	
3216	(C90)	HEX	4		ADDRESS OF PRIMARY PARM ADDR
3220	(C94)	HEX	4		ADDR OF ALTERNATE
3224	(C98)	HEX	4		L-FORM--ATTRIB BYTE
3228	(C9C)	HEX	4	FSEWRC	FULLSCREEN ERASE WRITE
		..1. .111		$ESC	"X'27'" ESCAPE
		1111 .1.1		$EWR	"X'F5'" ERASE WRITE
		.111 111.		$EWRA	"X'7E'" ERASE WRITE ALTERNATE
3230	(C9E)	HEX	6	FSTCAM	TCAM FULLSCREEN EXIT STRING
3236	(CA4)	HEX	4	IBMOPT1	RESERVED FOR IBM USE
3240	(CA8)	HEX	4	IBMOPT2	RESERVED FOR IBM USE
3244	(CAC)	HEX	4	IBMOPT3	RESERVED FOR IBM USE

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
3248	(CB0)	HEX	4	IBMOPT4	RESERVED FOR IBM USE
3252	(CB4)	HEX	4	IBMOPT5	RESERVED FOR IBM USE RSVD
3256	(CB8)	HEX	4	IBMOPT6	RESERVED FOR IBM USE RSVD
3260	(CBC)	HEX	4	IBMOPT7	RESERVED FOR IBM USE RSVD
3264	(CC0)	HEX	4	IBMOPT8	RESERVED FOR IBM USE RSVD
3268	(CC4)	HEX	4	HDSC403	RESERVED FOR IBM USE HDSC

APL BATCH-I/O MANAGEMENT
THE FOLLOWING DATA MUST BE CONTIGUOUS SINCE IT IS
INITIALIZED AS A SINGLE BLOCK.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
3272	(CC8)	HEX	4	BATDATAA	START OF CONTIG DATA ||

QSAM CONTROL BLOCKS

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
3272	(CC8)	HEX	0	BATOPEN	ALIGN LIST TO FULLWORD
3272	(CC8)	HEX	1		OPTION BYTE
3273	(CC9)	HEX	3		DCB ADDRESS
3276	(CCC)	HEX	1		OPTION BYTE
3277	(CCD)	HEX	4		DCB ADDRESS
3280	(CD0)	HEX	0	APLIN	ORIGIN ON WORD BOUNDARY DIRECT ACCESS DEVICE INTERFACE
3280	(CD0)	HEX	16		FDAD,DVTBL
3296	(CE0)	HEX	4		KEYLE,DEVT,TRBAL COMMON ACCESS METHOD INTERFACE
3300	(CE4)	HEX	1		BUFNO
3301	(CE5)	HEX	3		BUFCB
3304	(CE8)	HEX	2		BUFL
3306	(CEA)	HEX	2		DSORG
3308	(CEC)	HEX	4		IOBAD FOUNDATION EXTENSION
3312	(CF0)	HEX	1		BFTEK,BFLN,HIARCHY
3313	(CF1)	HEX	3		EODAD
3316	(CF4)	HEX	1		RECFM
3317	(CF5)	HEX	3		EXLST FOUNDATION BLOCK
3320	(CF8)	HEX	8		DDNAME
3328	(D00)	HEX	1		OFLGS
3329	(D01)	HEX	1		IFLG
3330	(D02)	HEX	2		MACR BSAM-BPAM-QSAM INTERFACE
3332	(D04)	HEX	1		RERI
3333	(D05)	HEX	3		CHECK, GERR, PERR
3336	(D08)	HEX	4		SYNAD
3340	(D0C)	HEX	2		CIND1, CIND2
3342	(D0E)	HEX	2		BLKSIZE
3344	(D10)	HEX	4		WCPO, WCPL, OFFSR, OFFSW

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
3348	(D14)	HEX	4		IOBA
3352	(D18)	HEX	1		NCP
3353	(D19)	HEX	3		EOBR, EOBAD QSAM INTERFACE
3356	(D1C)	HEX	4		RECAD
3360	(D20)	HEX	2		QSWS
3362	(D22)	HEX	2		LRECL
3364	(D24)	HEX	1		EROPT
3365	(D25)	HEX	3		CNTRL
3368	(D28)	HEX	4		PRECL
3372	(D2C)	HEX	4		EOB
3376	(D30)	HEX	0	APLPRINT	ORIGIN ON WORD BOUNDARY DIRECT ACCESS DEVICE INTERFACE
3376	(D30)	HEX	16		FDAD,DVTBL
3392	(D40)	HEX	4		KEYLE,DEVT,TRBAL COMMON ACCESS METHOD INTERFACE
3396	(D44)	HEX	1		BUFNO
3397	(D45)	HEX	3		BUFCB
3400	(D48)	HEX	2		BUFL
3402	(D4A)	HEX	2		DSORG
3404	(D4C)	HEX	4		IOBAD FOUNDATION EXTENSION
3408	(D50)	HEX	1		BFTEK,BFLN,HIARCHY
3409	(D51)	HEX	3		EODAD
3412	(D54)	HEX	1		RECFM
3413	(D55)	HEX	3		EXLST FOUNDATION BLOCK
3416	(D58)	HEX	8		DDNAME
3424	(D60)	HEX	1		OFLGS
3425	(D61)	HEX	1		IFLG
3426	(D62)	HEX	2		MACR BSAM-BPAM-QSAM INTERFACE
3428	(D64)	HEX	1		RER1
3429	(D65)	HEX	3		CHECK, GERR, PERR
3432	(D68)	HEX	4		SYNAD
3436	(D6C)	HEX	2		CIND1, CIND2
3438	(D6E)	HEX	2		BLKSIZE
3440	(D70)	HEX	4		WCPO, WCPL, OFFSR, OFFSW
3444	(D74)	HEX	4		IOBA
3448	(D78)	HEX	1		NCP
3449	(D79)	HEX	3		EOBR, EOBAD QSAM INTERFACE
3452	(D7C)	HEX	4		RECAD
3456	(D80)	HEX	2		QSWS
3458	(D82)	HEX	2		LRECL
3460	(D84)	HEX	1		EROPT
3461	(D85)	HEX	3		CNTRL

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
3464	(D88)	HEX	4		PRECL
3468	(D8C)	HEX	4		EOB
				BATDATAZ	"*" END OF CONTIG DATA
		11.. 1...		BATDATAL	"BATDATAZ-BATDATAA" LEN OF CONTIG DATA

END OF CONTIGUOUSLY INITIALIZED DATA
MISCELLANEOUS OTHER BATCH I/O MANAGEMENT DATE

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
3472	(D90)	HEX	4	BINSAVE	ADDR OF NEXT INPUT BUFFER OR ZERO. THIS IS USED TO SAVE THE NEXT LINE WHEN ATTN IS PRESSED FOR THE PREVIOUS LINE (WHICH CAN HAPPEN WHEN APL IS RUN IN BATCH MODE UNDER TSO).
3476	(D94)	HEX	4	BOUTADDR	| > NEXT BYTE IN OUTPUT RCD
3480	(D98)	HEX	4	BOUTLEN	V REMAINING LEN OF OUTPUT RCD

MESSAGE EDITTING ROUTINE WORK AREA

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
3488	(DA0)	HEX	0	ERDSECT	APLYULNE (APLEDIT) WORK AREA
3488	(DA0)	HEX	8	ERT1	DOUBLE-WORD WORKSPACE
3496	(DA8)	HEX	4	ERT2	TWO DOUBLE-WORDS WORKSPACE

SAVE AREAS

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
3512	(DB8)	HEX	64	ERSAVE	
3576	(DF8)	HEX	4	ERPAS13	PASS THIS SAVE AREA IN REG 13 TO BALR'ED-TO ROUTINES

RECONSTRUCTED PLIST AREA

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
3648	(E40)	HEX	4	ERPF1	FIRST FLAG BYTE
		1...		ERF1TX	"X'80'" TEXT ADDRESS IN PLIST
		.1..		ERF1HD	"X'40'" HEADER IN PLIST
		..1.		ERF1BF	"X'20'" BUFFER ADDRESS IN PLIST
		...1		ERF1SB1	"X'10'" ONE SUBSTITUTION
		.... 1...		ERF1SBN	"X'08'" MULTIPLE SUBSTITUTIONS (> 1)
3649	(E41)	HEX	4	ERPF2	SECOND FLAG BYTE
		1...		ERF2CM	"X'80'" BLANK COMPRESSION WANTED
		.1..		ERF2DT	"X'40'" DOT AT END OF LINE WANTED
		..1.		ERF2DI	"X'20'" 'DIE = YES' WANTED

LAST THREE BITS INDICATE 'DISP' FIELD

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
		....		ERF2ER	"0" ERRMSG
		1		ERF2TY	"1" TYPE
		.... ..1.		ERF2SI	"2" SIO
		11		ERF2NO	"3" NONE
		.... .1..		ERF2PR	"4" PRINT
		.... .1.1		ERF2CP	"5" CPCOMM
3652	(E44)	HEX	4	ERPTXA	TEXT ADDRESS

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
3656	(E48)	HEX	0	ERPHDR	ERROR MESSAGE HEADER
3656	(E48)	HEX	2	ERPNUM	MESSAGE NUMBER
3658	(E4A)	HEX	1	ERPLET	MESSAGE LETTER
3659	(E4B)	HEX	5	ERPCS	CSECT NAME
3664	(E50)	HEX	4	ERPBFA	BUFFER ADDRESS (FOR 'BUFFA')
FIELDS FOR SUBSTITUTION					
3668	(E54)	HEX	4	ERPSBA	POINTER TO FIRST (NEXT) GROUP OF SUB PARAMS IN ORIGINAL PLIST
3672	(E58)	HEX	4	ERSBD	DATA ADDR/VALUE OR CURRENT SUB
3676	(E5C)	HEX	4	ERSBF	SUB FLAG BYTE FOR CURRENT SUB
		1...		ERSFLST	"X'80'" THE LAST SUBSTITUTION PARAM
		.1..		ERSFA	"X'40'" 'A'-TYPE OPTION
		..1.		ERSFL	"X'20'" LENGTH SPECIFIED
LAST THREE BITS GIVE OPTION TYPE					
		....		ERSFH	"0" HEX OR HEXA
		1		ERSFD	"1" DEC OR DECA
		.... ..1.		ERSFC	"2" CHARA
		11		ERSFH4	"3" HEX4A
		.... .1..		ERSFC8	"4" CHAR8A
3677	(E5D)	HEX	3	ERSBL	SUB LENGTH BYTE FOR CURRENT SUB
3680	(E60)	HEX	4	ERSSZ	SIZE OF SUB FIELD (# DOTS 1)
MESSAGE CONSTRUCTION AREA					
3688	(E68)	HEX	8		NEED DOUBLE WORD BEFORE TEXT
3696	(E70)	HEX	3	ERMESS	FIRST LETTERS OF HEADER
3699	(E73)	HEX	3	ERSECT	DSECT NAME
3702	(E76)	HEX	3	ERNUM	MESSAGE NUMBER
3705	(E79)	HEX	1	ERLET	MESSAGE LEVEL LETTER
3706	(E7A)	HEX	4	ERBL	BLANK
		1... ..1.		ERTSIZE	"130" MAX TEXT SIZE
3707	(E7B)	HEX	4	ERTEXT	MESSAGE TEXT AREA
'TYPLIN'/'PRINTR' PLIST CONSTRUCTION AREA					
3840	(F00)	HEX	0		
3840	(F00)	HEX	8	ERTPL	
3848	(F08)	HEX	4	ERTPLA	(ERMESS) MESSAGE TEXT ADDR
3852	(F0C)	HEX	4	ERTPLL	MESSAGE LENGTH
WORK AREA USED FOR EDITING SUPERVISOR MESSAGES WITH APLEDIT MACRO.					
3856	(F10)	HEX	4	CMSLINED	

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
SECOND LEVEL (HELP) MESSAGE CONTROL DATA					
3896	(F38)	HEX	4	ERMOREQ	ADDR OF)MORE MSG QUEUE)MORE
3900	(F3C)	HEX	3	ERMQHEAD	HEAD OF HELP MESSAGE QUEUE
3903	(F3F)	HEX	4	ERMQDLEN	DUMMY LEN FOR FAKE 1ST MSG
		....		ERMNOINF	"0" ZERO LENGTH MEANS NO INFO.)MORE
3904	(F40)	HEX	4	ERMFLAG1	FLAG BYTE
		1...		ERMWANTD	"BIT0" DISPLAY OF HELP MESSAGES HAS BEEN REQUESTED BY THE USER.
		.1..		ERM4CED	"BIT1" FORCE DISPLAY OF WHOLE Q
		..1.		ERMPLSD	"BIT2" MESSAGE WAS PLUSED)MORE
3908	(F44)	HEX	4	ERMGMNL	LENGTH
3912	(F48)	HEX	4		ADDR. OF ADDR. LIST
3916	(F4C)	HEX	1		MODE AND OPTION FLAGS
3917	(F4D)	HEX	4		SUBPOOL VALUE
MESSAGE BUFFER FIELD DISPLACEMENTS					
		....		ERMCHAIN	"0" QUEUE CHAIN FIELD
		11		ERMLEN	"ERMCHAIN+3" MSG TEXT LENGTH
		.... .1..		ERMTEXT	"ERMLEN+1" MSG TEXT BUFFER
		.... .1..		ERMPFXLN	"ERMTEXT" MSG BUFFER PREFIX LENGTH
THE FOLLOWING IDENTIFIES THE END OF THE MISCELLANEOUS DATA AREA AND SETS THE START OF THE BUFFER AREA AND ENSURES THAT THE TWO DO NOT					
				TAILSTAR	"TSOGL+X'1000'-X'20'-6"B MINUS 6 MINUS SOME MORE
3918	(F4E)	HEX	4		
DEFINE THE TERMINAL BUFFER HERE AND OVERLAY IT WITH VARIOUS TRANSIENT DATA. ENSURE THAT THE BUFFER STARTS SUFFICIENTLY SHORT OF TSOGL+X'1000' SO THAT THE TRANSIENT DATA REMAINS ADDRESSABLE. TERMINAL BUFFER					
4058	(FDA)	HEX	4	BUFPRFX	ROOM FOR AID,ADDR,SBA,ADDR
4064	(FE0)	HEX	4		
4064	(FE0)	HEX	4	CMSBUFF	
				CMSBUFFZ	"*"
				CMSBUFFL	"CMSBUFFZ-CMSBUFF" BUFFER LENGTH
6112	(17E0)	HEX	4	CMSBUFF2	BUFFER FOR KEYBOARD TRANSLATIONS BY APLYUS93 OF TPUT MESSAGES TO NON-DISPLAY TERMINALS.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
INVOCATION OPERAND PARSING DATA					
4064	(FE0)	HEX	4	IOPODEFI	PTR 2 1ST OPND DSCPTR NTRY
4068	(FE4)	HEX	4	IOPODELA	PTR 2 LAST ODE
4072	(FE8)	HEX	4	CMSAPADA	SVP PARMS
4076	(FEC)	HEX	4	IOPERMBU	PTR 2 ERR MSG BUF
4080	(FF0)	HEX	8	IOPSAVE	MISC SAVE AREA
4088	(FF8)	HEX	4	IOPFLAG	FLAG BYTE
		1...		IOPLOADL	"BIT0" A LOADLIB DDNAME HAS BEEN CREATED VIA THE LOADLIBS OPERAND.
4090	(FFA)	HEX	4	IOPBUFF	PARSING BUFFER
4094	(FFE)	HEX	0		
GLOBAL EQUATES.					
		.... 1.1.		TSOR	"R10" GLOBAL TABLE BASE REGISTER.
				TSOML	"*-TSOGL" GLOBAL TABLE LENGTH

CROSS REFERENCE

$ESC	3228 X'27'
$EWR	3228 X'F5'
$EWRA	3228 X'7E'
APATCHLN	1608 X'3C'
APATTACH	1552(610)
APLBLDX	2032(7F0)
APLDLTX	2048(800)
APLDTLX	0 (0)
APLID	194 (C2)
APLIN	3280(CD0)
APLOPTNS	192 (C0)
APLPRINT	3376(D30)
APL101WK	2661(A65)
BATDATAA	3272(CC8)
BATDATAL	3468 X'C8'
BATDATAZ	= 3472
BATOPEN	3272(CC8)
BINSAVE	3472(D90)
BOUTADDR	3476(D94)
BOUTLEN	3480(D98)
BUFPRFX	4058(FDA)
CMSABEND	1417 X'04'
CMSABND2	1417 X'20'
CMSABORT	1417 X'01'
CMSALIB	1200(4B0)
CMSANAM	1204(4B4)
CMSAPADA	4072(FE8)
CMSAPAS	1216(4C0)
CMSAPLSM	1418 X'08'
CMSAPWKL	= 512
CMSAVACT	1224(4C8)
CMSAVDAT	1848(738)
CMSAVE	524(20C)
CMSAVEZ	= 956
CMSAVEZP	956(3BC)
CMSAVSIZ	2268(8DC)
CMSAWSID	= 1200
CMSBCODE	1104(450)
CMSBFLIN	2288(8F0)
CMSBFSEG	2292(8F4)
CMSBMPSV	956 X'48'
CMSBPSW	1108(454)
CMSBREGS	1116(45C)
CMSBTPRG	1100 X'80'
CMSBTSYS	1100 X'40'
CMSBTYPE	1100(44C)
CMSBUFF	4064(FE0)
CMSBUFFL	= 2048
CMSBUFFZ	= 6112
CMSBUFF2	611(17E0)
CMSCANCL	1417 X'02'
CMSCONTX	1418 X'10'
CMSCOPER	1416 X'20'
CMSCPUST	1320(528)
CMSDAIR	520(208)
CMSDDADR	2628(A44)
CMSDMPNO	1508(5E4)
CMSECB	1304(518)
CMSECBL	1268(4F4)
CMSECBL1	1268(4F4)
CMSECBL2	1272(4F8)
CMSFLAGS	2286(8EE)
CMSFRADR	1180(49C)
CMSFRSIZ	1184(4A0)
CMSFSAWA	3204(C84)
CMSFSFLG	= 3204
CMSFSGWK	3212(C8C)
CMSFSMWK	3208(C88)
CMSGIVBK	= 32768
CMSHELD	= 2288
CMSHOLDT	1328(530)
CMSICTR	2276(8E4)
CMSIDLSW	2284(8EC)
CMSINBUF	= 2292
CMSINSVP	1416 X'40'
CMSLAST	2286 X'10'
CMSLINED	3856(F10)
CMSMAXWS	1196(4AC)
CMSMINDL	= 1000000
CMSMINWS	= 20480
CMSMONTH	1404(57C)
CMSNLREQ	2286 X'04'
CMSNLSW	2285(8ED)
CMSNOAUT	1418 X'20'
CMSNSTAE	1417 X'08'
CMSNUMAP	1284(504)
CMSOCTR	2280(8E8)
CMSOLDTE	1400(578)
CMSOUT	2286 X'08'
CMSPACK	1352(548)
CMSPACKL	1360 X'27'
CMSPDATE	1336(538)
CMSPGMFL	1416(588)
CMSPGMF2	1417(589)
CMSPGMF3	1418(58A)
CMSPSDT	1336(538)
CMSPTIME	1344(540)
CMSRADDR	2305(901)
CMSRLGTH	2310(906)
CMSSEGZ	2286 X'80'
CMSSHVFL	1264(4F0)
CMSSMADR	1292(50C)
CMSSMAD2	1300(514)
CMSSMSIZ	1288(508)
CMSSMSZ2	1296(510)
CMSSSMAD	1280(500)
CMSSTAE	1417 X'10'
CMSSVPIN	1284(504)
CMSTABS	2372(944)
CMSTAXL	2369 X'14'
CMSTAXPL	2352(930)
CMSTIMEP	1308 X'20'
CMSTRTUP	1312(520)
CMSTYOI	2286 X'20'
CMSTYOII	2286 X'02'
CMSVPECB	1276(4FC)
CMSVWAIT	1308 X'10'
CMSWADDR	2321(911)
CMSWAITF	1308(51C)
CMSWLGTH	2326(916)
CMSWPLL	2326 X'10'
CMSWPLST	2312(908)
CMSWSADR	1192(4A8)
CMSWSZVN	1418 X'40'
CMSXBND	1100(44C)
CMSXEQTR	1417 X'40'

CROSS REFERENCE

CMSXIN	2924(B6C)	ERM4CED	3904 X'40'	LIBDATLN	1864(748)
CMSXOUT	2668(A6C)	ERNUM	3702(E76)	LIBDCB	1640(668)
CMSXPLL	2310 X'10'	ERPAS13	3576(DF8)	LIBDCBL	1724 X'58'
CMSXPLST	2296(8F8)	ERPBFA	3664(E50)	LIBDECB	1728(6C0)
CMSXTYI	2628(A44)	ERPCS	3659(E4B)	LIBDECBL	1748 X'18'
CMSXTYO	2632(A48)	ERPF1	3648(E40)	LIBDECBN	1748(6D4)
CMSXTYOI	2636(A4C)	ERPF2	3649(E41)	LIBDROP	1930 X'01'
CMSYIN	3184(C70)	ERPHDR	3656(E48)	LIBFNCTN	1930(78A)
CMSYOUT	3180(C6C)	ERPLET	3658(E4A)	LIBGLOBL	1932(78C)
CMS4SOUT	2286 X'01'	ERPNUM	3656(E48)	LIBLBERR	1931 X'04'
CNTALIB	1868(74C)	ERPSBA	3668(E54)	LIBLBOFL	1932 X'80'
CNTANAM	1872(750)	ERPTXA	3652(E44)	LIBLDCPY	1932 X'40'
COPARET	2272(8E0)	ERSAVE	3512(DB8)	LIBLIB	1930 X'02'
CPPLSTG	1512(5E8)	ERSBD	3672(E58)	LIBLOAD	1930 X'03'
CPYAVAIL	2232(8B8)	ERSBF	3676(E5C)	LIBLOCAL	1931(78B)
CPYDATA	2172(87C)	ERSBL	3677(E5D)	LIBNCP	1576 X'05'
CPYDATL	2264 X'60'	ERSECT	3699(E73)	LIBNEWDS	1931 X'04'
CPYDATZ	= 2268	ERSFA	3676 X'40'	LIBNMLIB	1931 X'08'
CPYGETMN	2172(87C)	ERSFC	3676 X'02'	LIBOPEN	1636(664)
CPYGMQTY	2184(888)	ERSFC8	3676 X'04'	LIBOURS	1931 X'80'
CPYHEADL	2200(898)	ERSFD	3676 X'01'	LIBPREFL	2060(80C)
CPYMAXL	2192(890)	ERSFH	3676 X'00'	LIBPREFX	2062(80E)
CPYNXAVL	2260(8D4)	ERSFH4	3676 X'03'	LIBPUB	1931 X'20'
CPYRSDUL	2196(894)	ERSFL	3676 X'20'	LIBPWDKN	1931 X'02'
CPYSAVEA	2264(8D8)	ERSFLST	3676 X'80'	LIBQLFR	214 (D6)
CPYSLOT#	2224 X'03'	ERSSZ	3680(E60)	LIBSAVE	1930 X'04'
CPYSLOT1	2208(8A0)	ERTEXT	3707(E7B)	LIBSER	236 (EC)
CPYSLOT2	2216(8A8)	ERTPL	3840(F00)	LIBSHR	1931 X'40'
CPYSLOT3	2224(8B0)	ERTPLA	3848(F08)	LIBSORSD	1931 X'04'
CPYTAILL	2204(89C)	ERTPLL	3852(F0C)	LIBUID	2070(816)
DAPBCD	1552(610)	ERTSIZE	3706 X'82'	LIBUNIT	228 (E4)
DAPBCTRC	1558(616)	ERT1	3488(DA0)	LIBWSDSN	1886(75E)
DAPBDARC	1556(614)	ERT2	3496(DA8)	MAINAPAD	336 X'00'
DAPBFLG	1554(612)	FRSIZMN	320(140)	MAINAPNM	336 X'04'
DAPBS	1552(610)	FSEDINIT	3204 X'80'	MAINAPS	336(150)
DAPLSTG	1528(5F8)	FSEDOPEN	3204 X'08'	MAXDEBUG	324(144)
DBGECHO	109 X'02'	FSEWRC	3228(C9C)	MAXLIBNO	1860(744)
DBGMICRO	109 X'80'	FSMBFLSH	3204 X'20'	MINAI	296(128)
DBGMSG	109 X'01'	FSMODE	3204 X'04'	MINSH	312(138)
DBGNSTAE	109 X'40'	FSMSCSTK	3204 X'10'	MINWS	308(134)
DFLTLIBN	2024(7E8)	FSMSGFUL	3204 X'40'	MNAPENT	336 X'0C'
DFLTSM	316(13C)	FSTCAM	3230(C9E)	NEXTITEM	2648(A58)
DFRC	1548(60C)	FSWALEN	= 16384	NOFENCE	2660 X'40'
ERBL	3706(E7A)	GTTERM	3216(C90)	OLDPICA	1432(598)
ERDSECT	3488(DA0)	HDSC403	3268(CC4)	OPTBCH19	244 X'20'
ERF1BF	3648 X'20'	HEADEND	= 3918	OPTBITS1	244 (F4)
ERF1HD	3648 X'40'	IBMOPT1	3236(CA4)	OPTBITS2	245 (F5)
ERF1SBN	3648 X'08'	IBMOPT2	3240(CA8)	OPTBITS3	246 (F6)
ERF1SB1	3648 X'10'	IBMOPT3	3244(CAC)	OPTBITS4	247 (F7)
ERF1TX	3648 X'80'	IBMOPT4	3248(CB0)	OPTBLKSI	224 (E0)
ERF2CM	3649 X'80'	IBMOPT5	3252(CB4)	OPTDLTX	244 X'80'
ERF2CP	3649 X'05'	IBMOPT6	3256(CB8)	OPTEND	520(208)
ERF2DI	3649 X'20'	IBMOPT7	3260(CBC)	OPTEXIT	260(104)
ERF2DT	3649 X'40'	IBMOPT8	3264(CC0)	OPTFRS	252 (FC)
ERF2ER	3649 X'00'	IOPBUFF	4090(FFA)	OPTID	192 (C0)
ERF2NO	3649 X'03'	IOPERMBU	4076(FEC)	OPTLEN	= 328
ERF2PR	3649 X'04'	IOPFLAG	4088(FF8)	OPTLQ	212 (D4)
ERF2SI	3649 X'02'	IOPLOADL	4088 X'80'	OPTMDY	244 X'08'
ERF2TY	3649 X'01'	IOPODEFI	4064(FE0)	OPTMICRO	244 X'10'
ERLET	3705(E79)	IOPODELA	4068(FE4)	OPTPQ	202 (CA)
ERMCHAIN	3917 X'00'	IOPSAVE	4080(FF0)	OPTRSV1	249 (F9)
ERMESS	3696(E70)	LDSNSAVE	1392(570)	OPTSMSIZ	256(100)
ERMFLAG1	3904(F40)	LENSTACK	= 512	OPTSVPNM	328(148)
ERMGMNL	3908(F44)	LIB#LEN	1856 X'08'	OPTTPUT	248 (F8)
ERMLEN	3917 X'03'	LIB#LIST	1856(740)	OPTUSR	264(108)
ERMNOINF	3903 X'00'	LIBABEND	1932 X'20'	OPTUSR1	280(118)
ERMOREQ	3896(F38)	LIBAUTH	1931 X'01'	OPTUSR2	284(11C)
ERMPFXLN	3917 X'04'	LIBCONT	1930 X'05'	OPTUSR3	288(120)
ERMPLSD	3904 X'20'	LIBCONTU	1931 X'10'	OPTUSR4	292(124)
ERMQDLEN	3903(F3F)	LIBCOPY	1930 X'06'	OSSYSTYP	1419(58B)
ERMQHEAD	3900(F3C)	LIBDATAA	2024(7E8)	OURESTAE	1444(5A4)
ERMTEXT	3917 X'04'	LIBDATAL	2070 X'35'	OURPICA	1436(59C)
ERMWANTD	3904 X'80'	LIBDATAZ	= 2077	OURPICAL	1440 X'06'

PTH	0 (0)
PTHACCNO	36 (24)
PTHASYNC	0 (0)
PTHATTN	0 X'01'
PTHCNCTM	64 (40)
PTHCPULM	0 X'20'
PTHCPUTM	48 (30)
PTHCURSR	18 (12)
PTHCWBIT	3 X'80'
PTHDATTN	0 X'80'
PTHFOFF	0 X'02'
PTHFSAVL	5 X'08'
PTHKEYTM	56 (38)
PTHLOCAL	40 (28)
PTHLOCKB	5 X'80'
PTHMDY	5 X'40'
PTHMICRO	5 X'10'
PTHMSBLK	5 X'20'
PTHNOOUT	0 X'04'
PTHPARM1	24 (18)
PTHPARM2	28 (1C)
PTHQEND	0 X'40'
PTHQSIZE	20 (14)
PTHQVAR	6 (6)
PTHSINK	4 X'02'
PTHSIZE	64 X'48'
PTHSORS	4 X'01'
PTHSPCLY	8 X'80'
PTHSRCOD	10 (A)
PTHSUSP1	3 (3)
PTHSVBIT	3 X'20'
PTHSVON	4 X'80'
PTHUEXTN	5 X'04'
PTHUSTAT	5 (5)
PTHWABIT	3 X'40'
PTHWIDTH	14 (E)
PTHWORD1	0 (0)
PTHWSLEN	32 (20)
PTHWSTAT	4 (4)
PTHYYCOD	8 (8)
PTHYYRC	8 (8)
PTX	72 (48)
PTXADSM	110 X'08'
PTXAIFUR	110 X'80'
PTXATTN	104 (68)
PTXCICS	108 X'20'
PTXCMS	108 X'40'
PTXCODE	120 (78)
PTXDEBUG	109 (6D)
PTXDXTBP	112 (70)
PTXEND	176 X'B4'
PTXFLAG	108 (6C)
PTXFLAGS	110 (6E)
PTXFSRST	110 X'40'
PTXFSTBP	100 (64)
PTXGXGDM	92 (5C)
PTXGXTBP	88 (58)
PTXHELPQ	156 (9C)
PTXHIADT	153 (99)
PTXHIFLG	155 (9B)
PTXHIIHI	155 X'40'
PTXHIIOT	154 (9A)
PTXHILIT	152 (98)
PTXHIOHI	155 X'80'
PTXHISF	152 (98)
PTXLEN	176 X'6C'
PTXLEVEL	116 (74)
PTXPRIBP	96 (60)
PTXRSV01	164 (A4)
PTXRSV02	168 (A8)
PTXRSV03	172 (AC)
PTXRSV04	176 (B0)
PTXSCRTH	124 (7C)
PTXSMPRO	124 (7C)
PTXSMPSD	124 (7C)
PTXSMP1	124 (7C)
PTXSMP2	128 (80)
PTXSMP3	132 (84)
PTXSMP4	136 (88)
PTXSMP5	140 (8C)
PTXSMP6	144 (90)
PTXSMP7	148 (94)
PTXSMTBP	84 (54)
PTXSTACK	80 (50)
PTXSUBSY	108 (6C)
PTXTSO	108 X'80'
PTXUSRWA	160 (A0)
PTXVCT	76 (4C)
PTXVSPC	108 X'10'
PTXWSM	72 (48)
PUBQLFR	204 (CC)
PURGING	2660 X'80'
RTRYREGS	1480(5C8)
SCANSAVE	1032(408)
SHVAVAIL	1264 X'80'
SHVRFEAT	1264 X'08'
STACKBEG	2652(A5C)
STACKEND	2656(A60)
STAEREGS	1460(5B4)
STCKDATA	2660 X'20'
STCKLEN	300(12C)
STCKPOP	2644(A54)
STCKPURG	2640(A50)
STCKSAVE	1020(3FC)
STCKSTAT	2660(A64)
STXEXIT	2369 X'00'
S26PLIST	1552(610)
TAILSTAR	= 4058
TCAMFUDG	3202 X'04'
TERMSAVE	960(3C0)
TERMIRAC	2664(A68)
TRNNAME	1564(61C)
TSOAPLFC	2344 X'10'
TSOBATCH	2344 X'08'
TSOCDCC	1421(58D)
TSOCMDAT	1308 X'08'
TSOCTCB	1309(51D)
TSODH	3200(C80)
TSODPECO	2346 X'20'
TSODPFUL	2346 X'40'
TSODPHLD	2346 X'80'
TSODPYFL	2346(92A)
TSODSMAV	2346 X'10'
TSODTYPE	2344(928)
TSODW	3202(C82)
TSOFSTAL	1457 X'10'
TSOGL	0 (0)
TSOGLID	184 (B8)
TSOGYTBP	1096(448)
TSOIDLES	2344 X'04'
TSOLDCC	1420(58C)
TSOLOADL	1188(4A4)
TSOML	= 8160
TSOPFKAD	2348(92C)
TSOQTT	3196(C7C)
TSOR	4094 X'0A'
TSORADDR	2337(921)
TSORFLGS	2286 X'40'
TSORLGTH	2342(926)
TSORFLL	2342 X'10'
TSORFLST	2328(918)
TSOTCAM	2344 X'01'
TSOTRAN	1422(58E)
TSOTREZ	3192(C78)
TSOTRNM	1556(614)
TSOTROPT	1560(618)
TSOTRTBL	1552(610)
TSOTRZE	3188(C74)
TSOVTAM	2344 X'02'
TSOVTBL	1572(624)
TSOVTSB	1576(628)
TSOWSTIM	1248(4E0)
TSOWSUSR	1256(4E8)
TSO327CC	2345(929)
TSO3270	2344 X'80'
TSO3277	2344 X'40'
TSO3278	2344 X'20'
USRACTNO	1396(574)
WAITIMER	1308 X'40'
WAITRPLY	1308 X'80'
WORKDCB	2084(824)
WORKDCBL	2168 X'58'
WORKOPEN	2080(820)
WSHSAVE	1936(790)
WSSIZMX	304(130)

VCT (ALL)

This is the executor common services vector table, and contains addresses of service routines available with the executor being used (CICS/VS, CMS, or TSO). (The format of this layout is the one used in publications titled "Data Areas and Symbolic Names Cross-Reference Table," usually distributed on microfiche.) This control block is mapped by the APLXXVCT macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	92	VCT	COMMON VECTOR TABLE
0	(0)	A-ADDRESS	4	VCTMAINS	ADDR OF MAIN STORAGE SERVICES
4	(4)	A-ADDRESS	4	VCTFILES	ADDR OF FILE SERVICES
8	(8)	A-ADDRESS	4	VCTGDDXS	ADDR OF GDDX SERVICES
12	(C)	A-ADDRESS	4	VCTGDDXE	ADDR OF GDDX ENVIRONMENT DEP MOD
16	(10)	A-ADDRESS	4	VCTSTKIN	ADDR OF STACK INITIALIZATION
20	(14)	A-ADDRESS	4	VCTCAPS	ADDR OF COMMON AP SERVICES
24	(18)	A-ADDRESS	4	VCTCAPON	ADDR OF COMMON AP SERVICES SIGNO
28	(1C)	A-ADDRESS	4	VCTXTRAN	ADDR OF TRANSLATE SERVICES
32	(20)	A-ADDRESS	4	VCTXTRZE	ADDR OF TRANSLATE ZCODE->EBCDIC
36	(24)	A-ADDRESS	4	VCTXTREZ	ADDR OF TRANSLATE EBCDIC->ZCODE
40	(28)	A-ADDRESS	4	VCTXBXIT	ADDR OF ABEND EXIT ROUTINE
44	(2C)	A-ADDRESS	4	VCTDUMPX	ADDR OF DUMP SERVICE ROUTINE
48	(30)	A-ADDRESS	4	VCTWTPST	ADDR OF WAIT/POST ROUTINE
52	(34)	A-ADDRESS	4	VCTSTKAB	ADDR OF STACK ABEND EXIT SERVICE
56	(38)	A-ADDRESS	4	VCTXBEND	ENTRY FOR APL ABEND REQUEST
60	(3C)	A-ADDRESS	4	VCTPRTX	ENTRY FOR PRINT SERVICES
64	(40)	A-ADDRESS	4	VCTLEDIT	ENTRY POINT FOR APLEDIT ROUTINE
68	(44)	A-ADDRESS	4	VCTXVERS	ENTRY PT FOR CONVERSION SERVICES
72	(48)	A-ADDRESS	4	VCTMSGNQ	ENTRY POINT FOR HELPENQ ROUTINE
76	(4C)	A-ADDRESS	4	VCTRSV01	RESERVED
80	(50)	A-ADDRESS	4	VCTRSV02	RESERVED
84	(54)	A-ADDRESS	4	VCTRSV03	RESERVED
88	(58)	A-ADDRESS	4	VCTRSV04	RESERVED

CROSS REFERENCE

VCT	0	(0)
VCTCAPON	24	(18)
VCTCAPS	20	(14)
VCTDUMPX	44	(2C)
VCTFILES	4	(4)
VCTGDDXE	12	(C)
VCTGDDXS	8	(8)
VCTLEDIT	64	(40)
VCTMAINS	0	(0)
VCTMSGNQ	72	(48)
VCTPRTX	60	(3C)
VCTRSV01	76	(4C)
VCTRSV02	80	(50)
VCTRSV03	84	(54)
VCTRSV04	88	(58)
VCTSTKAB	52	(34)
VCTSTKIN	16	(10)
VCTWTPST	48	(30)
VCTXBEND	56	(38)
VCTXBXIT	40	(28)
VCTXTRAN	28	(1C)
VCTXTREZ	36	(24)
VCTXTRZE	32	(20)
VCTXVERS	68	(44)

VRD (XSYS, AP)

This is the common system services conversion services request block. It is mapped by the APLXVRD macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	24	VRD	CONVERSION REQUEST DESCRIPTOR
0	(0)	UNSIGNED	1	VRDRC	RETURN CODE
1	(1)	UNSIGNED	1	VRDRES	RESERVED
2	(2)	UNSIGNED	1	VRDTIN	TYPE OF INPUT PROVIDED
3	(3)	UNSIGNED	1	VRDTOU	TYPE OF OUTPUT DESIRED
4	(4)	A-ADDRESS	4	VRDPIN	POINTER TO INPUT
8	(8)	SIGNED	4	VRDXIN	INPUT INDEX (ORIGIN 0)
12	(C)	SIGNED	4	VRDKELM	COUNT OF ELEMENTS IN/OUT
16	(10)	A-ADDRESS	4	VRDPOU	POINTER TO OUTPUT AREA
20	(14)	A-ADDRESS	4	VRDLOU	LENGTH OF OUTPUT AREA
24	(18)	CHARACTER	0	VRDEND	END OF VRD

CROSS REFERENCE

VRD	0	(0)
VRDEND	24	(18)
VRDKELM	12	(C)
VRDLOU	20	(14)
VRDPIN	4	(4)
VRDPOU	16	(10)
VRDRC	0	(0)
VRDRES	1	(1)
VRDTIN	2	(2)
VRDTOU	3	(3)
VRDXIN	8	(8)

WSM (ALL)

The VS APL workspace (WSM) represents the state of one user's VS APL machine. It contains all data, functions, processor transient memory, and the interpreter side of all interpreter/executor communications. This control block is mapped by the APLWSM macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
0	(0)	STRUCTURE	2049	WSM	
0	(0)	CHARACTER	1956	WSMSIZDF	PREFIX NOT SAVED IN LIB
0	(0)	CHARACTER	1024	WSMBUFF	GENERAL BUFFER
1024	(400)	CHARACTER	72		SAVE AREA FOR EXECUTOR USE
1024	(400)	BITSTRING	8	WSMSUPSW	
1032	(408)	UNSIGNED	64	WSMSURGS	
1096	(448)	CHARACTER	8		POSITION INFORMATION
1096	(448)	UNSIGNED	2	WSMRSV02	
1098	(44A)	SIGNED	2	WSMCURSR	PRINT ELEMENT INDEX
1100	(44C)	SIGNED	2	WSMBFLIM	LEN OF WSMBUFF LESS)WIDTH
1102	(44E)	SIGNED	2	WSMBFPTR	LEN OF DATA IN WSMBUFF
1104	(450)	CHARACTER	8		PARAMETER WORDS
1104	(450)	A-ADDRESS	4	WSMPARM1	
1108	(454)	A-ADDRESS	4	WSMPARM2	
1112	(458)	CHARACTER	64	WSMSVLRQ	AREA FOR TRANSMITTING BLOCKS OF DATA TO AND FROM THE EXECUTOR ALSO USED FOR SCV TO SSM
1112	(458)	CHARACTER	36	WSMPDSD	FOR PDSD (SEE BELOW) SEE MORE DEFNS BELOW
1176	(498)	CHARACTER	96	WSMREGSV	
1176	(498)	UNSIGNED	4	WSMREG00	GENERAL REGISTERS
1180	(49C)	UNSIGNED	4	WSMREG01	
1184	(4A0)	UNSIGNED	4	WSMREG02	
1188	(4A4)	UNSIGNED	4	WSMREG03	
1192	(4A8)	UNSIGNED	4	WSMREG04	
1196	(4AC)	UNSIGNED	4	WSMREG05	
1200	(4B0)	UNSIGNED	4	WSMREG06	
1204	(4B4)	UNSIGNED	4	WSMREG07	
1208	(4B8)	UNSIGNED	4	WSMREG08	
1212	(4BC)	UNSIGNED	4	WSMREG09	
1216	(4C0)	UNSIGNED	4	WSMREG10	
1220	(4C4)	UNSIGNED	4	WSMREG11	

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1224	(4C8)	UNSIGNED	4	WSMREG12	
1228	(4CC)	UNSIGNED	4	WSMREG13	
1232	(4D0)	UNSIGNED	4	WSMREG14	
1236	(4D4)	UNSIGNED	4	WSMREG15	
1240	(4D8)	BITSTRING	8	WSMREGF0	FLOATING REGS
1248	(4E0)	BITSTRING	8	WSMREGF2	
1256	(4E8)	BITSTRING	8	WSMREGF4	
1264	(4F0)	BITSTRING	8	WSMREGF6	
1272	(4F8)	CHARACTER	8	WSMPCPSW	BC MODE RESUME PSW
1272	(4F8)	BITSTRING	4		HIGH-ORDER HALF
1276	(4FC)	A-ADDRESS	4	WSMNSI	CC/PMASK/RESUME ADDR
1280	(500)	A-ADDRESS	4	WSMPTHPT	ADDR OF PERTERM HEADER
1284	(504)	SIGNED	4	WSMWORK	M-REL OFFSET TO WSM STACK
1288	(508)	CHARACTER	664		(INTERPRETER USE)
1952	(7A0)	SIGNED	4	WSMFREEZ	OFFSET TO HIGH UNUSED
1956	(7A4)	CHARACTER	93	WSMLIBA	START OF WS IN LIBRARY
1956	(7A4)	SIGNED	4	WSMFREEA	OFFSET TO LOW UNUSED
1960	(7A8)	CHARACTER	4	WSMCHECK	WORKSPACE FORMAT
1964	(7AC)	SIGNED	4	WSMSYMX	USED IN SYMTAB SEARCH
1968	(7B0)	SIGNED	4	WSMSYMBL	CURRENT)SYMBOLS VALUE
1972	(7B4)	A-ADDRESS	4	WSMOLDMR	ADDR OF WS BEFORE YY-REQ
1976	(7B8)	CHARACTER	64	WSMLIBRS	RESERVED IN LIB RECORD
2040	(7F8)	BITSTRING	8		INTERPRETER TOGGLES
2048	(800)	CHARACTER	1	WSMASYNC	INTERRUPT FLAGS
		1...		WSMASIST	MICROCODE ASSIST
		.1..		WSMLNHLT	HALT AT LINE END
		..1.		WSMNOOUT	SUPPRESS OUTPUT
		...1			
		.... 1...		WSMIMHLT	IMMEDIATE HALT
		.... 1...		WSMDATTN	SAME AS IMHLT
1112	(458)	STRUCTURE	36	PDSD	LIB OP CONTROL BLOCK
1112	(458)	SIGNED	4	PDSLIBNO	LIB NUMBER
1116	(45C)	CHARACTER	11	PDSNAME	WS NAME
1127	(467)	CHARACTER	1		
1128	(468)	CHARACTER	8	PDSPASS	WS PASSWORD

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
1136	(470)	CHARACTER	1	PDSMOD	FLAG BYTE
		1...		PDSAUTO	SAVE OF CONTINUE
		.1..		PDSDFSIZ	USE ACTIVE WS SIZE
		..11			
		.... 1...		PDSDFNAM	USE ACTIVE WS NAME
		.... .1..		PDSDFPAS	USE ACTIVE WS PASS
		.... ..1.		PDSDFLIB	USE LIB AS PER DFLIB
		1		PDSDFACL	LIB FROM ACTIVE WS
1137	(471)	CHARACTER	3		RESERVED
1140	(474)	SIGNED	4	PDSSIZE	SIZE OF WORKSPACE
1144	(478)	SIGNED	4	PDSSLOP	MIN FOR STACK AND FREE
1112	(458)	STRUCTURE	28		YYQAI
1112	(458)	BITSTRING	8	WSMYYCTM	COMPUTE TIME
1120	(460)	BITSTRING	8	WSMYYTTM	TERMINAL TIME
1128	(468)	BITSTRING	8	WSMYYKTM	KEYING TIME
1136	(470)	SIGNED	4	WSMYYUNO	SIGN-ON ID
1112	(458)	STRUCTURE	16		YYDELAY
1112	(458)	BITSTRING	8	WSMYYDLA	TOD AT DELAY START
1120	(460)	BITSTRING	8	WSMYYDLZ	TOD AT DELAY END
1112	(458)	STRUCTURE	4		YYDUMP
1112	(458)	SIGNED	4	WSMYYDNM	NUMBER OF LAST DUMP
1112	(458)	STRUCTURE	24		YYQUOTA
1112	(458)	SIGNED	4	WSMYYQLS	TOTAL USER LIB SPACE
1116	(45C)	SIGNED	4	WSMYYQSR	REMAIN UNUSED SPACE
1120	(460)	SIGNED	4	WSMYYQDW	DEFAULT WS SIZE
1124	(464)	SIGNED	4	WSMYYQMW	MAXIMUM WS SIZE
1128	(468)	SIGNED	4	WSMYYQVM	SHARED MEMORY SPACE
1132	(46C)	SIGNED	4	WSMYYQVV	NUMBER OF SHARED VARS

CROSS REFERENCE

PDSAUTO	1136 X'80'	WSMREG09	1212(4BC)
PDSD	1112(458)	WSMREG10	1216(4C0)
PDSDFACL	1136 X'01'	WSMREG11	1220(4C4)
PDSDFLIB	1136 X'02'	WSMREG12	1224(4C8)
PDSDFNAM	1136 X'08'	WSMREG13	1228(4CC)
PDSDFPAS	1136 X'04'	WSMREG14	1232(4D0)
PDSDFSIZ	1136 X'40'	WSMREG15	1236(4D4)
PDSLIBNO	1112(458)	WSMRSV02	1096(448)
PDSMOD	1136(470)	WSMSIZDF	0 (0)
PDSNAME	1116(45C)	WSMSUPSW	1024(400)
PDSPASS	1128(468)	WSMSURGS	1032(408)
PDSSIZE	1140(474)	WSMSVLRQ	1112(458)
PDSSLOP	1144(478)	WSMSYMBL	1968(7B0)
WSM	0 (0)	WSMSYMX	1964(7AC)
WSMASIST	2048 X'80'	WSMWORK	1284(504)
WSMASYNC	2048(800)	WSMYYCTM	1112(458)
WSMBFLIM	1100(44C)	WSMYYDLA	1112(458)
WSMBFPTR	1102(44E)	WSMYYDLZ	1120(460)
WSMBUFF	0 (0)	WSMYYDNM	1112(458)
WSMCHECK	1960(7A8)	WSMYYKTM	1128(468)
WSMCURSR	1098(44A)	WSMYYQDW	1120(460)
WSMDATTN	2048 X'08'	WSMYYQLS	1112(458)
WSMFREEA	1956(7A4)	WSMYYQMW	1124(464)
WSMFREEZ	1952(7A0)	WSMYYQSR	1116(45C)
WSMIMHLT	2048 X'08'	WSMYYQVM	1128(468)
WSMLIBA	1956(7A4)	WSMYYQVV	1132(46C)
WSMLIBRS	1976(7B8)	WSMYYTTM	1120(460)
WSMLNHLT	2048 X'40'	WSMYYUNO	1136(470)
WSMNOOUT	2048 X'20'		
WSMNSI	1276(4FC)		
WSMOLDMR	1972(7B4)		
WSMPARM1	1104(450)		
WSMPARM2	1108(454)		
WSMPCPSW	1272(4F8)		
WSMPDSD	1112(458)		
WSMPTHPT	1280(500)		
WSMREGF0	1240(4D8)		
WSMREGF2	1248(4E0)		
WSMREGF4	1256(4E8)		
WSMREGF6	1264(4F0)		
WSMREGSV	1176(498)		
WSMREG00	1176(498)		
WSMREG01	1180(49C)		
WSMREG02	1184(4A0)		
WSMREG03	1188(4A4)		
WSMREG04	1192(4A8)		
WSMREG05	1196(4AC)		
WSMREG06	1200(4B0)		
WSMREG07	1204(4B4)		
WSMREG08	1208(4B8)		

WSX (ALL)

This control block defines the format of the 80-byte header record for an exported workspace in all systems. It is mapped by APLWSX macro.

OFFSETS		TYPE	LENGTH	NAME	DESCRIPTION
WSXDR DSECT					
0	(0)	SIGNED	4	WSXAR13	OFFSET TO END OF WS SLOT FROM WSMFREEA.
4	(4)	SIGNED	4	WSXAX	OFFSET TO END OF WRITTEN DATA FROM WSMFREEA.
8	(8)	SIGNED	4	WSXAOFFS	OFFSET OF WSMFREE FROM BEGINNING OF WS.
12	(C)	SIGNED	4		UNUSED.
16	(10)	FLOATING	8	WSXDATE	DATE AND TIME WS WAS SAVED IN APL STANDARD TIME FORMAT.
24	(18)	SIGNED	4	WSXLIB	LIBRARY NUMBER.
28	(1C)	CHARACTER	8	WSXENAME	WS NAME IN EBCDIC. 8 BYTES.
36	(24)	CHARACTER	11	WSXZNAME	WS NAME IN Z-CODES.
47	(2F)	HEX	3,3		RESERVED

SECTION 6. DIAGNOSTIC AIDS

This section contains information useful in determining the causes of VS APL processor errors. It contains the following items:

- Descriptions of linkage conventions
- Expansions of calling macros
- A list of executor service calls for CICS/VS, CMS, TSO, and VSPC
- Lists of error messages generated by the processor
- Directions for reading and analyzing dumps issued after system errors
- Descriptions of miscellaneous diagnostic aids

COMPONENT LINKAGE CONVENTIONS

The components of the VS APL processor use several linkage conventions. The types of linkage conventions used between and within components are summarized in Figure 31. The remainder of this subsection describes the component linkage conventions listed there.

FROM	TO A	B	C	D	E	F	G	H	I	J
A. Host	-	13	-	-	-	-	13	13	13	-
B. Executor	13	4,5 6	7	-	-	12	12	-	-	11
C. Translator	-	7	1	1	1	-	-	-	-	-
D. Exarch	-	-	-	3	1	-	-	-	-	-
E. Appendage	-	7	1	3	1	-	-	-	-	-
F. Shared storage mgr.	13	-	-	-	-	9,10	-	-	-	-
G. Auxiliary processor	13	-	-	-	-	12	-	-	-	11
H. Conversion program	13	-	-	-	-	-	-	8	-	-
I. Library service prog.	13	-	-	-	-	-	-	-	2	-
J. Stack protocol exec.	13	4,5 6	-	-	-	12	12	8	-	11

Notes	Component Linkage Convention Used
1	VS APL interpreter linkage
2	APL library service program linkage
3	Nonstandard linkage to and within exarch
4	CMS/TSO executor linkage
5	VSPC executor linkage
6	CICS/VS executor linkage
7	Service request calls
8	Conversion program linkage
9	CMS/TSO shared storage manager
10	CICS/VS shared storage manager
11	Common executor linkage
12	Auxiliary processor linkage: see the applicable manual on writing auxiliary processors
13	Standard linkage using host system macros
-	Does not occur or does not apply

Figure 31. Component Linkage Conventions

1. VS APL INTERPRETER LINKAGE

Register Usage

The register usage implemented by the VS APL processor is shown in Figure 32.

Register	Use	
1	Reason code	
	Value	Meaning
	0	good
	4	error
	8	severe error
11	Address of active workspace	
12	Program base register	
13	Address of current R13 stack level	
14	Address of end of current R13 stack level	
15	Return register	

Figure 32. VS APL Processor Register Usage

Since VS APL standard linkage does not use parameter lists, other registers are used to pass parameters between calling and called routines. These conventions are documented in the listings of the routines or their prologs.

Save Areas

VS APL standard linkage uses a pushdown stack of contiguous save areas that are kept in the user's workspace. The current save area is always pointed to by register 13. See "R13 Stack" in "Data Areas" for a complete discussion of how the R13 stack functions.

Calling Macros

There are several macros used by VS APL standard linkage in making calls and returns. These macros are expanded and described below.

The APLCALL macro instruction, which calls a routine, is written as follows:

[label]	APLCALL	routine [,ERROR1=lbl1[,ERROR2=lbl2]]

label
: is an optional statement label.

routine
: is the name of the global routine defined by use of APLENTRY.

ERROR1= lbl1
causes assembly of a branch instruction to this label immediately following the call of the routine. lbl1 receives control if the called routine returns reason-code 4.

ERROR2= lbl2
causes assembly of a second branch instruction immediately following the call of the routine. lbl2 receives control if the called routine returns reason-code 8.

The APLENTRY macro instruction, which defines an entry point, is written as follows:

[entry-pt]	**APLENTRY**	lcl-begin,lcl-end [,reg-list][,**NOCALL**]

entry-pt
is the entry point name.

lcl-begin
is the name of the DSECT that describes this routine's R13 stack work area; 0 if no work area.

lcl-end
is the name of the address immediately following the last field in the DSECT; 0 if no work area.

reg-list
is the list of caller's registers to be saved. A single value causes a store instruction to be generated; multiple values cause a store-multiple instruction.

NOCALL
defines a terminal routine; that is, no subsequent APLCALLs may be used. If this option is specified, no registers are saved. The called routine may explicitly save one or more registers at register 14 plus 0.

The APLEXIT macro instruction, which returns to the caller, is written as follows:

[label]	**APLEXIT**	[,**REASON=**regno]

label
is an optional statement label.

REASON=regno
causes an exit to the point of invocation plus a displacement equal to the contents of register regno. Register regno should contain only 0, 4, or 8.

Code generated for APLCALL and APLEXIT is described in Figure 33. In each example, the APLEXIT is from the routine called by the APLCALL.

Code generated for APLENTRY with various parameter options is described in Figure 34.

With no error return:

```
                APLCALL    IAHTSPEC
+               L          R15,=V(IAHTSPEC)
+               BALR       R15,R15

+               APLEXIT
+               DS         0H
+               LM         R12,R15,0(R13)
+               STM        R13,R14,WSMREG13
+               LM         R0,R7,16(R14)
                Registers 0-7 specified in APLENTRY macro.
+               BR         R15
```

With error return:

```
 SPQPW20        APLCALL    IARTOI,ERROR1=SPQPW30
+SPQPW20        L          R15,=V(IARTOI)
+               BALR       R15,R15
+               B          SPQPW30 FIRST ERROR RETURN (R15+0)
 CVEXIT         APLEXIT    REASON=1
+CVEXIT         DS         0H
+               LM         R12,R15,0(R13)
+               STM        R13,R14,WSMREG13
                No registers specified in APLENTRY macro.
+               B          0(R1,R15) RETURN + 0, 4, OR 8
```

Figure 33. APLCALL and APLEXIT: Generated Code

```
IASYSPST       APLENTRY  0,0
+              DS        0D
+              DC        CL8'IASYSPST'
+              ENTRY     IASYSPST
+IASYSPST      EQU       *
+              STM       R12,R15,0(R14)
+              LR        R13,R14
+              LA        R14,(0-0+((16+7)/8*8)+7)/8*8(0,R13)
+              STM       R13,R14,WSMREG13
+              BALR      R12,0
+              USING     *,R12

 IARABREF      APLENTRY  0,0,(2,4)
+              DS        0D
+              DC        CL8'IATABREF'
+              ENTRY     IATABREF
+IATABREF      EQU       *
+              STM       R12,R15,0(R14)
+              STM       R2,R4,16(R14)
+              LR        R13,R14
+              LA        R14,(0-0+((28+7)/8*8)+7)/8*8(0,R13)
+              STM       R13,R14,WSMREG13
+              BALR      R12,0
+              USING     *,R12

 IASYSREF      APLENTRY    SRWORK,SRWORKZ
+              DS        0D
+              DC        CL8'IASYSREF'
+              ENTRY     IASYSREF
+IASYSREF      EQU       *
+              STM       R12,R15,0(R14)
+              LR        R13,R14
+              LA        R14,(SRWORKZ-SRWORK+((16+7)/8*8)+7)/8*8(0,R13)
+              STM       R13,R14,WSMREG13
+              USING     SRWORK-(16+7)/8*8,R13
+              BALR      R12,0
+              USING     *,R12

 IAFACT        APLENTRY  TRSWORK,TRSWORKZ,(1,3)
+              DS        0D
+              DC        CL8'IAFACT'
+              ENTRY     IAFACT
+IAFACT        EQU       *
+              STM       R12,R15,0(R14)
+              STM       R1,R3,16(R14)
+              LR        R13,R14
+              LA        R14,(TRSWORKZ-TRSWORK+((28+7)/8*8)+7)/8*8(0,R13)
+              STM       R13,R14,WSMREG13
+              USING     TRSWORK-(28+7)/8*8,R13
+              BALR      R12,0
+              USING     *,R12
```

Figure 34. APLENTRY: Generated Code

2. APL LIBRARY SERVICE PROGRAM LINKAGE

Register Usage

The register usage implemented by the APL library service program is shown in Figure 35.

Register	Use
0	Reason code returned to the calling routine
11	Address of the APL library service program global table
13	Address of the register save area
14	Return address to the calling routine
15	Entry point address of called routine or return code returned to calling routine

Figure 35. APL Library Service Program Register Usage

Save Areas

Registers are saved in an 18-word area pointed to by register 13.

3. NONSTANDARD LINKAGE TO AND WITHIN EXARCH

Register Usage

The register usage implemented by nonstandard linkage to and within exarch is shown in Figure 36.

Register	Use
11	Address of workspace
12	Base register
13	Address of exarch's R13 stack level
14	Address of end of exarch's R13 stack level
15	Subroutine linkage

Figure 36. Nonstandard Exarch Register Usage

Save Areas

None

Calling Macros

Calls to subroutines within the same module as the calling routine use a branch-and-link instruction using register 15.

The APLXCALL macro instruction, which calls a subroutine outside of the module of the calling routine, is written as follows:

[label]	APLXCALL	routine

label
    is an optional statement label.

routine
    is the name of the called routine defined by use of APLXNTRY.

Code generated for APLXCALL is described in Figure 37.

```
 GOGOMN4       APLXCALL  IEGINITI              GET 1ST ELEMENT INTO RVALUE
+GOGOMN4       L         BASEREG,=A(IEGINITI)  LOAD ADDRESS OF SUBROUTINE
+              BALR      R15,BASEREG           CALL SUBROUTINE
+              USING     *,R15                 RETURN
+              L         BASEREG,=A(APLIEFNM)  RELOAD BASE REGISTER
+              DROP      R15
+              USING     APLIEFNM,BASEREG
```

Figure 37. APLXCALL: Generated Code

The APLXNTRY macro instruction, which defines an entry point, is written as follows:

[entrypt]	APLXNTRY	entry-address

entry-pt
    is the entry point name.

entry-address
    is the optional entry-address register; if omitted, register 12 (equated to BASEREG) is assumed.

Code generated for APLXNTRY is described in Figure 38.

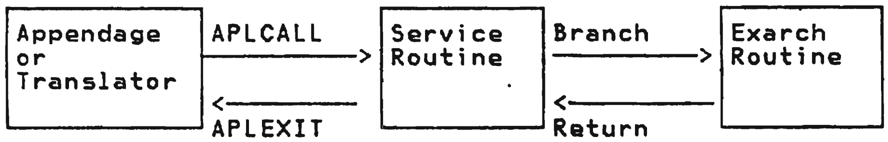
```
 IEGOGOMN      APLXNTRY
+              USING     *,BASEREG
+IEGOGOMN      L         BASEREG,=A(APLIEFNM) LOAD BASE REGISTER
+              USING     APLIEFNM,BASEREG

 IEGOGOSC      APLXNTRY  R15
+              USING     *,R15
+IEGOGOSC      L         BASEREG,=A(APLIEFNM) LOAD BASE REGISTER
+              DROP      R15
+              USING     APLIEFNM,BASEREG
```

Figure 38. APLXNTRY: Generated Code

Transfer of control when return is not expected uses no set convention. The address of the routine is loaded in some register and a branch is taken.

Some of the exarch subroutines may also be called by appendage and translator routines. A two-level linkage is used to effect these calls as described below.



A service routine contains three basic statements:

entry-point	APLENTRY	Save caller's registers
	BALR15,routine	Call exarch routine
	APLEXIT	Register caller's registers and return

The names of all service routines begin with IES.

4. CMS/TSO EXECUTOR LINKAGE

The CMS and TSO executors use similar linkage conventions. Unless otherwise noted, the following information applies to both.

Register Usage

The register usage implemented by the CMS or TSO executor is shown in Figure 39.

Register	Use
10	CMS/TSO: address of global area (PERTERM control block followed by executor work area)
11	Address of VS APL workspace
12	Base register
13	Address of register save area
14	Return address
15	Entry point address

Figure 39. Executor Linkage Register Usage (CMS or TSO)

Parameters, when required, are passed in the workspace and, in CMS and TSO, the global area.

A return code is passed in PTHSRCOD (two-byte field in the PERTERM). Return codes are described under "Service Request Calls."

Save Areas

Registers are saved in an 18-word area pointed to by register 13. There are three such areas or levels reserved in the CMS or TSO executor work area. When a routine is called, the calling routine's registers are saved in the current level. Register 13 is then set to the address of the next level.

Calling Macros (CMS and TSO)

There are three macros used by the CMS or TSO executor in calling and returning to routines within itself: APLCENTR, APLCCALL, and APLCEXIT.

The APLCENTR macro instruction, which defines an entry point, is written as follows:

[entry-pt]	**APLCENTR**	

entry-pt
is the entry point name.

Code generated for APLCENTR is described in Figure 40.

```
 SCTYOI         APLCENTR
+               ENTRY      SCTYOI
+SCTYOI         DS         0H
+               USING      SCTYOI,R15           R15+IS ENTRY POINT REG.
+               STM        R14,R12,12(R13)      SAVE CALLER'S REGISTERS.
+               LA         R14,CMSDMPSV(,R13)   BUMP TO NEXT AREA IN STACK.
+               C          R14,CMSAVEZP         IF END OF STACK REACHED,
+               BL         ENT0040              (WE DIDN'T REACH IT.)
+               L          R1,=V(SCSAVOFL)      IT'S A SUPERVISOR BUG.
+               BALR       R0,R1                BRANCH TO ERROR ROUTINE.
+ENT0040        ST         R14,8(,R13)          LINK FORWARD TO NEW SAVE.
+               ST         R13,4(,R14)          AND BACKWARDS TO CALLERS SAV
+               LR         R13,R14              SET NEW SAVE AREA ADDR.
+               L          R12,=A(APLSCTYP)     SET PROGRAM BASE
+               USING      APLSCTYP,R12
+               DROP       R15
```

Figure 40. APLCENTR: Generated Code

The APLCCALL macro instruction, which calls a routine, is written as follows:

[label]	APLCCALL	routine

label
is an optional statement label.

routine
name of called routine; must be defined by APLCENTR macro.

Code generated for APLCCALL is described in Figure 41.

```
 TYOI2     APLCCALL   SCTYI
+TYOI2     L          R15,=A(SCTYI)  GET ADDR OF CALLED ROUTINE.
+          BALR       R14,R15        BRANCH TO ROUTINE.
```

Figure 41. APLCCALL: Generated Code

The APLCEXIT macro instruction, which returns to the caller, is written as follows:

[label]	APLCEXIT	

label
is an optional statement label.

Code generated for APLCEXIT is described in Figure 42.

```
               APLCEXIT
+              L          R13,4(,R13)         ADDR OF CALLER'S SAVE AREA.
+              LM         R14,R12,12(R13)     RESTORE CALLER'S REGISTERS.
+              BR         R14                 RETURN TO CALLER.
```

Figure 42. APLCEXIT: Generated Code

5. VSPC EXECUTOR LINKAGE

Register Usage

The register usage implemented by the VSPC executor is shown in Figure 43.

Register	Use
9	Address of PTC—VSPC control block
10	Address of workspace defined by VSPC (includes PERTERM and executor work area)
11	Address of VS APL workspace
12	Base register
13	Address of register save area
14	Return address
15	Entry point address

Figure 43. Executor Linkage Register Usage (VSPC)

Parameters, when required, are passed in the workspace.

A return code is passed in PTHSRCOD (two-byte field in the PERTERM). Return codes are described under "Service Request Calls."

Save Areas

Registers are saved in an 18-word area printed to by register 13. There are five such areas or levels reserved in the executor work area. When a routine is called, the calling routine's registers are saved in the current level. Register 13 is then set to the address of the next level. The pointer to the save area is saved as a relative address to the workspace.

Calling Macros (VSPC)

There are two macros used by the VSPC executor in calling and returning to routines within itself: APLPENTR and APLPEXIT.

The APLPENTR macro instruction, which defines an entry point, is written as follows:

[entry-pt]	**APLPENTR**	

entry-pt
    is the entry point name.

Code generated for APLPENTR is described in Figure 44.

```
 PCTYI         APLPENTR
+              DS         0D
+              DC         CL8'PCTYI'
+PCTYI         EQU         *
+              ENTRY      PCTYI
+              USING      PCTYI,R15              ADDRESSABILITY
+              USING      PTC,R9                 R9 HAS ADDR OF PTC
+              USING      WSH,R10                R10 HAS ADDR OF TOTAL WS
+*                                               PROVIDES ADDRESSABILITY TO
+*                                               WSH,SFN,PTH,ECA
+              STM        R14,R12,12(R13)        SAVE CALLERS REGISTERS
+              LA         R2,ECASVLN(R13)        BUMP TO NEXT SAVE AREA
+              LA         R1,ECASVEND            COMPARE TO END OF SAVE AREA
+              CR         R2,R1                  ANY ROOM LEFT ?
+              BL         APLP0062               YES,SET UP
+              L          R15,=V(ERSAVEAR)       NO,GO TO ERROR
+              BALR       R14,R15                ROUTINE
+APLP0062      EQU        *
+              LR         R1,R13                 CALLER'S SAVE ADDR
+              SR         R1,R10                 RELATIVIZE ADDR TO THE WS
+              ST         R1,4(R2)               SAVE IN OUR SAVE AREA
+              LR         R1,R2                  OUR SAVE ADDR
+              SR         R1,R10                 RELATIVIZE ADDR TO THE WS
+              ST         R1,8(,R13)             LINK SAVE AREAS
+              LR         R13,R2                 SAVE ADDR IN R13
+              L          R12,=A(APLPTYIO)       ADDRESSABILITY IN CSECT
+              DROP       R15
+              USING      APLPTYIO,R12
```

Figure 44. APLPENTR: Generated Code

The APLPEXIT macro instruction, which returns to the caller, is written as follows:

[label]	**APLPEXIT**	

label
    is an optional statement label.

Code generated for APLPEXIT is described in Figure 45.

```
             APLPEXIT
+            L          R13,4(,R13)          CALLER'S SAVE AREA
+            AR         R13,R10              OBTAIN ABSOLUTE ADDR PTR
+            LM         R14,R8,12(13)        RESTORE R14-R8
+            L          R12,12+14*4(,R13)    AND R12
+*                                           R10,R11 HAVE BEEN RELOCATED
+            BR         R14                  RETURN TO CALLER
```

Figure 45. APLPEXIT: Generated Code

6. CICS/VS EXECUTOR LINKAGE

Register Usage

The register usage implemented by the CICS/VS executor is shown in Figure 46.

Register	Use
7	Address of VS APL workspace
8	Address of global table (GBL)
9	Base Register
10	Address of perterm (PTH, PTK, and PRO control blocks)
11	Address of stack
12	Address of CICS/VS TCA
13	Address of CICS/VS CSA
14	Return address
15	Entry point address or return code

Figure 46. Executor Register Usage (CICS/VS)

Parameters are passed in either register 1 or the user perterm. Register 0 is destroyed by the linkage.

A return code is passed to PTHSRCOD (two-byte field in the PERTERM). Return codes are described under "Service Request Calls."

Save Areas

The CICS/VS executor uses a processing stack as a save area. See "VS APL Executor Stack for CICS/VS" in "Section 5. Data Areas" for a complete discussion of how the processing stack functions.

Calling Macros

CICS/VS executor modules use three macros to generate entry and exit code: APLKPOP, APLKSTAK, and APLKPROC. In addition, the following macros are used in calling routines: APLKG, APLKHIST, APLKT, APLKMAIN, APLKMILA, APLKPOST, APLKTERM, and APLKTRCE.

The APLKPOP macro, which defines an exit point from a routine, is written as follows:

[label]	**APLKPOP**	entry-pt[,retcode]

label
: is an optional statement label.

entry-pt
: is the entry point name; must match the label on the most recently issued APLKSTAK or APLKPROC macro; multiple exits from a routine may be used.

retcode
: is a return code to be placed in register 15; if omitted, register 15 is not modified.

The APLKSTAK macro defines the beginning of a work area to be used by the routine that follows it. The work area is described by a series of DS statements placed between the APLKSTAK macro and the APLKPROC macro following it. If no work area (other than a register save area) is required, the APLKSTAK macro can be omitted. The APLKPROC macro (or, if present, the APLKSTAK macro) defines an internal or external routine entry point and what registers are to be saved. If the APLKSTAK macro is used, the APLKPROC macro will have no operands.

The APLKSTAK AND APLKPROC macros are written as follows:

[entry-pt]	**APLKSTAK** or **APLKPROC**	[regname] [TYPE=ENTRY][,SAVE=(reg,reg)]

entry-pt
: is the entry point name; it will be an external entry point if TYPE=ENTRY is specified. The CSECT name may be used for the first routine in a CSECT.

regname
: can be used to provide unique names for registers used within this routine; register equates are provided by the macro; and registers are saved and restored on entry to and exit from this routine; registers are assigned from a pool defined as RLOCALA ... RLOCALG at the beginning of the module; pool registers must be consecutive and named in order, the numbering normally beginning with register 2.

TYPE=ENTRY
: indicates that all registers are to be saved at the beginning of this routine and that module addressability is to be established; register 10, the base register, is always set to the address at the beginning of the CSECT.

SAVE=(reg,reg)
: indicates the range of registers to be saved on entry to and restored at exit from this routine; if this operand is omitted and if TYPE=ENTRY was specified, registers 2 through 10 are saved; otherwise, as many registers as are listed in the regname parameter are saved; in either case, register 14 is always saved and restored.

An example of code generated for the APLKSTAK and APLKPROC macro when an external routine has been specified is described in Figure 47.

```
 APLKAHST      APLKSTAK RDIFF,TYPE=ENTRY
+*
+              USING    *,15
+              B        ENT0003
+              DC       AL1(ENT0003-*-1),C'APLKAHST',C'05/11/78'
+ENT0003       DS       0H
+              DROP     15
+STKD0003      DSECT
+STKP0003      DS       A                PTR to TOP OF STACK AREA
+STKR0003      DS       CL4              PROCEDURE ID
+STKR0003      DS       A                REG 14
+STKS0003      DS       9F               OTHER SAVED REGS
+RDIFF         EQU      RLOCALA
 CLOCKNEW      DS       D
 CLOCK0        DS       F                HIGH ORDER CLOCK WORD
 CLOCK1        DS       F                LOW ORDER CLOCK WORD
               APLKPROC
+*
+STKD0003      DSECT
+              ORG
+              DS       0F               ROUND UP TO WORK BOUNDARY
+STKL0003      EQU      *-STKD0003
+APLKAHST      CSECT
+              USING    STKD0003,11
+              L        15, STKP0003     PTR TO END OF STACK
+              LA       0,STKL0003       BACK UP STACK PTR
+              SR       11,0             BY LENGTH OF STACK ENTRY
+              LR       0,15             SAVE END-OF-STACK PTR
+              L        15,0(15)         PTR TO ERROR ROUTINE
+              CR       11,0             IS THERE ENOUGH SPACE?
+              BNHR     15               (NO - GOTO ERROR ROUTINE)
+              ST       0,STKP0003       IF SO, SAVE ENT PTR
+              ST       14,STKR003       SAVE REGISTER(S)
+              STM      2,10,STKS0003
+              BALR     10,0             MODULE ADDRESSABILITY
+              USING    *,10
+              LA       0,*-APLKAHST
+              SR       10,0
+              USING    APLKAHST,10
+              MVC      STKI0003,=C'AHST' SET ID FOR DEBUG
```

Figure 47. APLKSTAK and APLKPROC: Generated Code for External Routines

An example of code generated for the APLKPOP macro is described in Figure 48.

```
 CALC8         APLKPOP  APLKAHST         RETURN TO CALLER
+*
+CALC8         DS       0H
+              NI       4(11),X'BF'      OVERLAY ID WITH EXIT FLAG
+              L        14,8(11)         RESTORE REGS
+              LM       2,10,12(11)
+              LA       11,STLK0003(11)      POP STACK
+              BR       14               RETURN  TO CALLER
```

Figure 48. APLKPOP: Generated Code

Code generated for the APLKPROC macro when an internal routine has been specified is described in Figure 49.

```
 HISTUP         APLKPROC
+*
+*
+HISTUP        DS         ON
+STKD0009      DSECT
+STKP0009      DS         A               PTR TO TOP OF STACK AREA
+STKI0009      DS         CL4             PROCEDURE ID
+STKR0009      DS         A               REG 14
+STKS0009      DS         0F              OTHER SAVED REGS
+STKD0009      DSECT
+              ORG
+              DS         0F              ROUND UP TO WORD BOUNDARY
+STKL0009      EQU        *-STKD0009
+APLKAHST      CSECT
+              USING      STKD0009,11
+              L          15,STKP0009     PTR TO END OF STACK
+              LA         0,STKL0009      BACK UP STACK PTR
+              SR         11,0            BY LENGTH OF STACK ENTRY
+              LR         0,15            SAVE END-OF-STACK PTR
+              L          15,0(15)        PTR TO ERROR ROUTINE
+              CR         11,0            IS THERE ENOUGH SPACE?
+              BNHR       15              (NO - GOTO ERROR ROUTINE)
+              ST         0,STKP0009      PTR TO END OF STACK
+              ST         14,STKR0009     PTR TO END OF STACK
+              MVC        STKI0009,=C'HIST' SET ID FOR DEBUG
```

Figure 49. APLKPROC: Generated Code for Internal Routines

The APLKEXIT macro defines an exit routine to be entered if a program check or abend interrupt is encountered. It is written as follows:

[APLKEXIT]	ENTRY=<u>name</u>,PARM=,<u>value</u>

ENTRY=

gives the name of the exit routine entered. Only one exit routine may be in effect for a given process. On return, if a previous exit definition has been overridden, R0 contains its entry point, and R1 contains its parameter. Exits may be "stacked" by saving this information, and using it later as APLKEXIT ENTRY = (0), PARM = (1) to restore the previous exit.

PARM=

specifies a value to be passed to the exit routine in R1 if it is entered. It is an optional field. Frequently, the address of the current stack entry (R11) is employed.

The Exit Routine

On entry to the abend exit routine, R1 contains the parameter specified on the APLKEXIT macro, and R0 points to an abend exit block described below. R15 contains the entry point address, and R14 is an abend return address. All other registers are set as they existed at the last time an APLKEXIT or APLKWAIT was issued by the process. (Note that an APLKWAIT may have been issued by a different module from that issuing the APLKEXIT.)

The routine is actually dispatched as a retry routine, and does not normally return to its caller. If it should return, the abend condition is raised again. Since the exit is implicitly cancelled when it is invoked, a return from the retry routine will terminate the process unless the retry routine has set a new APLKEXIT or restored a previous APLKEXIT.

A pointer to an abend exit block is passed to the routine in R0. The data in that block is valid only until the next program check or abend in the APLU task. The block contains space for a PSW, registers 0-15, and floating point registers 0-8, in that order (there is no mapping macro for this block). The high order bit of the PSW should first be checked. If it is off, a program check has occurred, and the PSW and register save areas contain valid information. If the high order bit is on, an abend has occurred, and the first four bytes contain an EBCDIC CICS/VS abend code. In this case, no additional information (PSW or register) is available.

The APLKG macro, which invokes library services by queueing a request for the library task, is written as follows:

[APLKG]	libserve,LISTA=area,TYPE=,code

libserve
: states the library service requested.

area
: is a pointer to a global request element (GRE). GREs contain four words which are formatted by the APLKG macro expansion, and a two-word extension which must be set up by the caller before issuing the request.

code
: Codes and their meanings follow:

Code	Meaning
LOAD	Load a workspace
SAVE	Save a workspace
DROP	Drop a workspace
WDIR	Add a directory entry to the APL directory
WLIB	Write a control interval in the APL library
UDIR	Update an entry in the APL directory
CFILE	Create an auxiliary processor 121 file
DFILE	Drop an auxiliary processor 121 file
UFILE	Extend an auxiliary processor 121 file
RLIB	Read a control interval from the APL library

The APLKHIST macro, which is used to invoke the histogram data recorder routine, is written as follows:

[label]	**APLKHIST**	type,oldclock

label
: is an optional statement label.

type
: is one of the following names:

- INPUT1—the time between APL issuing a terminal read request to CICS/VS and notification to APL of completion of the request.
- INPUT2—the interevent time between two read operations
- REACT1—the elapsed time between first recognition of the input by APL and the time when the input is passed to the interpreter.
- INTERP1—interpreter processing time per input operation. This time is not precisely processor time, because it may include time, of which VS APL or CICS/VS has not been notified, that was used by the operating system or higher priority partitions.
- INTERP2—the length of time that processing is interrupted for a user when the interpreter is stopped at the end of a time slice.
- DEPEND1—the elapsed time between auxiliary processor dispatch and the next auxiliary processor wait operation.
- DEPEND2—the time between two auxiliary processor dispatch operations.
- CALC—the difference in time between current time and oldclock time. This time is returned without making any histogram update.
- LOAD1—the time it takes for a)LOAD operation to be processed.
- LOAD2—the time elapsed between two successive)LOAD requests for the same user.
- SAVE1—the time it takes for a)SAVE operation to be processed.
- SAVE2—the time elapsed between two successive)SAVE requests for the same user.
- COPY1—the time it takes for a)COPY operation to be processed.
- COPY2—the time elapsed between two successive)COPY requests for the same user.
- LOGON1—the elapsed time between logon and when interpreter code is first entered.
- LOGON2—the time between the beginning and end of the user session.

oldclock
is the name of a doubleword containing output from the STCK hardware instruction. Register notation may be used.

When control is returned from the APLKHIST macro, floating point register 0 contains the _newclock_ value (as it was provided by the STCK hardware instruction, not as a floating point number). The caller may use the STD hardware instruction to save this value to use when issuing a subsequent APLKHIST macro. Register 15 will contain the difference in time between _oldclock_ time and current time in milliseconds. If, however, register 15 contains a negative value, it will be one of the following return codes:

-4 The _oldclock_ value is zero. No histogram update is made in this case, but the newclock value is returned.

-8 The clock is not running. Note that a clock that is running but not set is acceptable.

The APLKMAIN macro invokes CICS/VS DFHSC GETMAIN and FREEMAIN routines. If CLASS=USER is specified or is the default, the macro gets enough storage to allow for the SSA (storage accounting area) affixed by CICS/VS; that is, the address returned by the GETMAIN routine and passed to the FREEMAIN routine points beyond the SAA.

The APLKMAIN macro is written as follows:

[_label_]	APLKMAIN	GET,_length_[CLASS=,COND=,INITIMG=] FREE,_addr_

label
is an optional statement label.

GET, _length_
requests storage, the _length_ being the number of bytes of storage requested. If register notation is used, it indicates that the length is in the specified register. The storage address is returned in register 1.

FREE, _addr_
requests that the storage acquired by the GET parameter be freed, _addr_ being the address of that storage.

CLASS=,COND=,INITIMG=
These parameters are defined in the _CICS/VS Application Programmer's Reference Manual (Macro Level)_.

The APLKT macro, which provides linkage between terminal manager routines, is written as follows:

[_label_]	APLKT	TARGET[,G][,G=_gblptr_] [,P=_parm_]

label
is an optional statement label.

TARGET
is either the label (within the module currently executing) of the routine to be called or a keyword identifying that routine. Valid keywords and the entry points to which control is routed are:

Keyword	**Entry Point**
CLEAR	KTSCLEAR
FCHECK	KTSFCHK
FINDF	KTSFNDF
HLINE	KTSLINE
LOCID	KTSLOCID
LOCREQ	KTSLOCR
SCHED	KTSSCHED
SETCUR	KTRCU
TRAN	KTRTRAN

G
indicates that the **TARGET** operand specifies a keyword rather than a label. In this case, the entry point to which control is to be passed will be located through the GBL (global) table. If the GBL table is not addressable, see the G=gblptr operand.

P=parm
the address of a fullword parameter to be passed in register 1 to the entry point getting control. If register notation is used, it indicates that the parameter is in the identified register.

The APLKTERM macro, which passes control to terminal manager entry points, is written as follows:

APLKTERM	trqdaddr,TYPE=tlist,OPT=olist

trqdaddr
gives the address of the TRQD (terminal request descriptor) parameter list. This parameter is required and positional.

TYPE=tlist
Indicates the type of request. A single type keyword may be used, or a list of type keywords may be enclosed in parentheses. Type keywords and their meanings follow:

- FORMAT—define the screen format.
- WRITE—write or erase data by field.
- READ—wait for input (if necessary) and summarize it.
- GETDATA—get data by field.
- FLDATTR—set field attributes.
- GETFORM—get the current screen format.
- HCOPY—request hardcopy output of the screen image.
- ALARM—ring the alarm.
- SETCUR—set the cursor.

- RESTORE—restore the screen.
- INIT—indicates this is the initial service request.
- FINAL—indicates this is the final service request.
- YES—indicates that type bits have been set in TRQ fields TRQTYP1 and TRQTYP2.

The following type keywords are mutually exclusive: INIT, FINAL, FORMAT, WRITE, READ, GETDATA, FLDATTR, and GETFORM. They are considered major requests and result in control being passed to one of the following entry points:

Keyword	Entry Point
INIT	KTRIN
FINAL	KTRFI
FORMAT	KTRFM
WRITE	KTRWR
READ	KTRRD
GETDATA	KTRGD
FLDATTR	KTRFA
GETFORM	KTRGA

The following type keywords may be specified alone or with others: HCOPY, ALARM, SETCUR, and RESTORE. (HCOPY and RESTORE may be used only with the WRITE keyword.) These keywords are considered minor requests. If they are specified with major request keywords, control is passed to the entry point listed previously for major request keywords. Otherwise, control is passed to the KTRRT entry point.

OPT=olist
indicates what options are desired on requests. A single **OPT** keyword may be used, or a list of **OPT** keywords may be enclosed in parentheses and separated by commas. **OPT** keywords and their meanings follow:

- ALT—use the alternate screen.
- REFORM—is specified with the type keyword FORMAT. Requests a reformat of the screen.
- FCHECK—is specified with the type keyword FORMAT. Requests verification of the screen format given on the FORMAT request.
- WAIT—is specified with the type keyword WRITE. Requests that the system wait for the results of a write operation.
- NULL—is specified with the type keyword WRITE. Requests that the trailing blanks be treated as null.
- NODAT—is specified with READ or GETDATA. If the keyword is READ, requests that no data information be returned. If the keyword is GETDATA, requests that field lengths be returned instead of data.
- WILLWR—is specified with the type keyword READ. Specifies that a write operation will immediately follow the read operation.
- YES—means that option bits have been set in the TRQOPT field in the TRQD. If the **OPT** keyword is not specified, the TRQOPT field will be cleared.

The APLKTRCE macro, which invokes the CICS/VS DFHTR trace routine for a user 193 trace, is written as follows:

[label]	**APLKTRCE**	id,dddddd, [,BYTE=bb[],R14=rrrrrr]

label
: is an optional statement label.

id
: is two hexadecimal digits identifying what was traced. An, Bn, and Cn indicate the traced data was associated with executor routines, operating system calls, and shared storage manager services respectively.

ddddd
: is the address of three (or if the BYTE operand is omitted, four) bytes of traced data. If register notation is used, the traced data is in the specified register.

BYTE=bb
: allows the high-order byte of the traced data word (ddddd) to be overlaid with the data specified in bb. If register notation is used, the data to be overlaid is in the low-order byte of the register.

R14=rrrrr
: is the caller's address. If this parameter is specified, rrrrr overrides the default caller's address that's specified in the register save area in the stack.

The trace records generated by the APLKTRCE macro have the following format:

bb	dddddd	id	rrrrrr

7. SERVICE REQUEST CALLS (CICS/VS, CMS, TSO, OR VSPC)

Service request calls are made from the interpreter or translator to the executor (CICS/VS, CMS, TSO, or VSPC).

Register Usage

Register	Use
0	Address of parameter (service request code)
1	Entry point address
11	Address of workspace

Parameters other than service request codes are passed in the workspace.

Return codes are passed in PTHSRCOD.

Save Areas

The interpreter's or translator's registers are saved in the workspace at location WSMREGSV.

Calling Macros

The interpreter or translator passes control to the executor by issuing a service request using the APLSVCC macro.

The APLSVCC macro instruction is written as follows:

[label]	APLSVCC	id,yycode,

The yycode is implicitly the first parameter of any service request. It specifies exactly what service is desired. The yycode is a 16-bit constant aligned on a halfword boundary. The high-order bit of the yycode is a flag used by the interpreter to determine if the last request issued marked the end of one work session and the start of another. The remaining 15 bits represent a positive numeric value that specifies the service requested. The value assigned to a particular service is arbitrary. In the interpreter, yycode values are always referred to by name; the numeric value of the low-order 15 bits of the yycode has meaning only to the executor.

Code generated for APLSVCC is described in Figure 50.

```
                APLSVCC  YYQAI
+               L        R1,=V(APLFXIIM)
+               BALR     R0,R1
+               DC       AL2(YYQAI)
+SVCC0020       EQU      *
```

Figure 50. APLSVCC: Generated Code

Values, Parameters, and Return Codes for Service Requests

REGULAR SERVICE REQUESTS: Regular service requests are those that do not invoke library operations, and are not concerned with shared variables.

Value 0001 - YYQZ

Service: Relinquish control at end of quantum.
Parameters: None

Return Codes:

0000 Continue

8000 Illegal: issued when executor had not requested quantum end.

CICS/VS Routine: APLKMSCB, KCQZ
CMS Routine: APLSCMSC, SCQZ
TSO Routine: APLYUMSC, SCQZ
VSPC Routine: APLPMISC, PCQZ

Value 0002 - YYTYO

Service: Send output to terminal.

Parameters:

WSMPARM1 Absolute address of data string

WSMPARM2 Length of data string (negative value if last byte of string not a new-line character)

Return Codes:

0000 Completed

8000 WSMPARM1 (or sum of WSMPARM1 and WSMPARM2) addresses an area not in the workspace or interpreter; WSMPARM2 positive and last character not a new line; absolute value of WSMPARM2 greater than 1024.

CICS/VS Routine: APLKIFIX, KYYTYOI
CMS Routine: APLSCTYP, SCTYO, APLSCDPY, SCDTYO
TSO Routine: APLYUTYF, SCTYO, APLYUDPY, SCDTYO
VSPC Routine: APLPTYIO, PCTYO, PCDTYO
Session manager routine: APLACPRO

Value 0003 - YYTYI

Service: Get input from terminal.

Parameters:

WSMBUFF String of prepared input

WSMBFPTR Length of prepared input in bytes at this time

Return Codes:

0000 Completed

0001 Buffer overflow

0002 Character error

0003 O-U-T discovered

8000 WSMBFPTR or WSMCURSR less than 0; WSMCURSR greater than WSMBFPTR; issuing workspace is in copy status; WSMCURSR not equal to PTHCURSR when request issued.

CICS/VS Routine: APLKIFIX, KYYTYOI
CMS Routine: APLSCTYP, SCTY1, APLSCDPY, SCDTYI
TSO Routine: APLYUTYP, SCTYI, APLYUDPY, SCDTYI
VSPC Routine: APLPTYIO, PCTYI, PCDTYO
Session manager routine: APLACPRO

Value 0004 - YYTYOI

Service: Output prompt, then get input from terminal.

Parameters:

WSMPARM1 Addresses WSMBUFF

WSMPARM2 Contains length of string in WSMBUFF

WSMBUFF Contains short character string containing no terminal-control Z-codes

WSMBFPTR Contains length of string in WSMBUFF

WSMCURSR Equal to WSMBFPTR

Return Codes: See "YYTYI."
CICS/VS Routine: APLKIFIX, KYYTYOI
CMS Routine: APLSCTYP, SCTYOI, APLSCDPY, SCDTYOI
TSO Routine: APLYUTYP, SCTYOI

VSPC Routine: APLPTYIO, PCTYOI, PCDTYOI
Session manager routine: APLACPRO

Value 0005 - YYATOFF

Service: Turn off attention and cancel bits in PERTERM.
Parameters: None
Return Codes: Always 0000
CICS/VS Routine: APLKMSCB, KCATOFF
CMS Routine: APLSCTYP, SCATOFF
TSO Routine: APLYUMSC, SCATOFF
VSPC Routine: APLPMISC, PCATOFF

Value 0006 - YYTIME

Service: Send date and time of day in VS APL standard time format.
Parameters: None

Return Codes:

0000 Completed

8000 Workspace in copy status

CICS/VS Routine: APLKMSCA, KCATOFF
CMS Routine: APLSCMSC, SCTIME
TSO Routine: APLYUMSC, SCTIME
VSPC Routine: APLPMISC, PCTIME

Value 0007 - YYQAI

Service: Send account information.
Parameters: None

Return Codes:

0000 Completed

8000 Workspace in copy status

CICS/VS Routine: APLKMSCA, KCQAI
CMS Routine: APLSCMSC, SCQAI
TSO Routine: APLYUMSC, SCQAI
VSPC Routine: APLPMISC, PCQAI

Value 0008 - YYRWAIT

Service: Wait for response to message.
Parameters: None

Return Codes:

0000 Completed

8000 Illegal: workspace in copy status; PTHCURSR not equal to 0.

CICS/VS Routine: not supported
CMS Routine: APLSCMSG, SCRWAIT
TSO Routine: APLYUMSC, SCRWAIT
VSPC Routine: APLPTYIO, PCRWAIT

Value 0009 - YYDELAY

Service: Set delay interval.

Parameters:

WSMPARM1 The desired delay interval in VS APL

WSMPARM2 standard time format.

Return Codes:

0000 Completed

8000 Workspace in copy status

CICS/VS Routine: APLKMSCA, KCDELAY
CMS Routine: APLSCMSC, SCDELAY
TSO Routine: APLYUMSC, SCDELAY
VSPC Routine: APLPTYIO, PCDELAY

Value 000A - YYTABS

Service: Set or send tab settings.

Parameters:

WSMBUFF Parameter string

WSMBFPTR 0 = reference tabs; non-0 = set tabs

Return Codes:

0000 Completed

0001 Tabs not supported

8000 Illegal: invalid parameter string; WSMBFPTR less than 0 or greater than 255; workspace in copy status.

CICS/VS Routine: APLKMSCB, KCTABS
CMS Routine: APLSCTYP, SCTABS
TSO Routine: APLYUMSC, SCTABS
VSPC Routine: APLPTYIO, PCTABS

Value 000B - YYWIDTH

Service: Accept specified line width.

Parameters:

WSMPARM1 New width setting

Return Codes:

0000 Accepted

8000 Illegal: WSMPARM1 greater than 255 or less than 30; workspace in copy status.

CICS/VS Routine: APLKMSCB, KCWIDTH
CMS Routine: APLSCTYP, SCWIDTH
TSO Routine: APLYUTYP, SCWIDTH
VSPC Routine: APLPTYIO, PCWIDTH

Value 000C - YYMBL

Service: Block or unblock messages.

Parameters:

WSMPARM1	Contains signal value; 0 = no change; 1 = block; 2 = unblock

Parameters:

0000	Accepted
0001	Not supported
0002	Invalid for this user
8000	Illegal: WSMPARM1 is not 0, 1, or 2; this workspace is in copy status.

CICS/VS Routine: APLKMSCB, KCMBL
CMS Routine: APLSCMSG, SCMBL
TSO Routine: APLYUMSG, SCMBL
VSPC Routine: APLPTYIO, PCMBL

Value 000D - YYTRAN

Service: Transmit message.

Parameters:

WSMPARM1	Absolute address of the first nonblank character following the verb of an OPR or MSG command, the text of which is in WSMBUFF.
WSMPARM2	Bit 0=1 if command verb was OPR or OPRN Bits 1-31 = length of string addressed by WSMPARM1

Return Codes:

0000

0000	Sent
0001	Message lost
0002	User not receiving
0003	Target not signed on
0004	Target undecipherable
8000	Illegal: WSMPARM1, or WSMPARM1 plus WSMPARM2 (1-31) do not fall in WSMBUFF; WSMPARM2 (1-31) greater than 1024; this workspace in copy status.

CICS/VS Routine: APLKMSCB, KCTRAN
CMS Routine: APLSCMSG, SCTRAN
TSO Routine: APLYUMSG, SCTRAN
VSPC Routine: APLPTYIO, PCTRAN

Value 0010 - YYCOPI

Service: Accept a buffer of copy data from copy source workspace to copy sink workspace.

Parameters: None

Return Codes:

0000	Data supplied

0001 No data available

8000 Illegal: this workspace is not copy sink.

CICS/VS Routine: APLKLIBU, KCOPI
CMS Routine: APLSCOPY, SCCOPI
TSO Routine: APLYUOPY, SCCOPI
VSPC Routine: APLPMISC, PCCOPI

Value 0011 - YYCOPO

Service: Offer a buffer of copy data to copy sink workspace.

Parameters:

WSMBUFF Block of copy data

WSMBFPTR Length of data in WSMBUFF

Return Codes:

0000 Completed

0001 Partner has terminated

8000 Illegal: WSMBFPTR less than or equal to zero, or greater than the maximum length of WSMBUFF; this workspace not copy source.

CICS/VS Routine: APLKLIBU, KCOPU
CMS Routine: APLSCOPY, SCCOPO
TSO Routine: APLYUOPY, SCCOPO
VSPC Routine: APLPMISC, PCCOPO

Value 0012 - YYCOPZ

Service: Take the issuing workspace out of copy status.

Parameters: None

Return Codes:

0000 Completed

0001 This workspace is copy sink and not all copy data has been consumed.

8000 Illegal: workspace not in copy status.

CICS/VS Routine: APLKLIBU, KCOPZ
CMS Routine: APLSCOPY, SCCOPZ
TSO Routine: APLYUDPY, SCCOPZ
VSPC Routine: APLPMISC, PCCOPZ

Value 0013 - YYDUMP

Service: Dump active workspace and PERTERM.
Parameters: None
Return Code: Always 0000
CICS/VS Routine: APLKMSCA, KCDUMP
CMS Routine: APLSCERR, SCDUMP
TSO Routine: APLYUERR, SCDUMP
VSPC Routine: APLPSERR, PCDUMP

Value 0014 - YYOFF

Service: Terminate session.

Parameters:

WSMPARM1 Bit 31; 0 = no hold, 1 = hold

Return Codes: None
CICS/VS Routine: APLKMSCB, KYYOFF
CMS Routine: APLSCMSC, SCOFF
TSO Routine: APLYUMSC, SCOFF
VSPC Routine: APLPMISC, PCOFF

Value 0015 - YYSYSER

Service: Transmit system error message.

Parameters:

WSMBUFF Character string (error message)

WSMBFPTR Length of string

Return Codes: Always 0000
CICS/VS Routine: APLKMSCB, KCSYSER
CMS Routine: APLSCERR, SCSYSER
TSO Routine: APLYUERR, SCSYSER
VSPC Routine: APLPSERR, PCSYSER

Value 0016 - YYQUOTA

Service: Print information concerning workspace, library, and shared variable quotas.
Parameters: None

Return Codes:

0000 Completed

8000 Illegal: workspace in copy status.

CICS/VS Routine: APLKMSCB, KCQUOTA
CMS Routine: APLSCMSC, SCQUOTA
TSO Routine: APLYUMSC, SCQUOTA
VSPC Routine: APLPMISC, PCQUOTA

Value 0017 - YYCMD

Service: Pass all commands to supervisor before they are processed by the interpreter.

Parameters:

WSMPARM1 Points to command verb block

WSMPARM2 Points to command operand block

Both blocks are of the form

```
DC   X(LEN-1), CL(LEN) '....'
```

On return, WSMBUFF may contain z-code output data with embedded new-line characters; WSMBFPTR must contain either the length of this data, or, if no data is to be displayed, 0.

Return Codes:

0000 Completed

0001 WSMBUFF full: clear and restart

0002 Provide standard error message

0003 Let the interpreter process the command

CICS/VS Routine: APLKMSCB, KYYCMD
CMS Routine: APLSCMSC, SCCMD
TSO Routine: APLYUMSC, SCCMD
VSPC Routine: APLPMISC, PCMD

LIBRARY REQUESTS: Library requests are those that entail a reference to the workspace library or the user directory. All the library requests share a common parameter list, called the PDSD, and a common set of return codes.

Parameters:

PDSLIBNO Library number

PDSNAME Workspace name

PDSPASS Password

PDSMOD Flag bits

PDSSIZE Workspace size

Return Codes:

0000 Good completion

0002 Library not found

0004 Workspace not found

0006 Incorrect password

0008 Improper library request

000A Name already exists (SAVE)

000C Workspace too large

000E Library is full

0010 Name in use (non-VS APL workspace)s:

0012 System resources full

0014 Library locked

0016 Space not available

0018 Size quota exceeded

0020 Library not available

0022 Filesize maximum exceeded

0024 Not supported

The following codes indicate hardware or software failure:

Return Codes:

0081 Directory read I/O failure

0082 Library read I/O failure

0085 Directory write I/O failure

0086 Library write I/O failure

8000 Illegal: bad PDSD, or this workspace in copy status.

Value 8020 - YYLOAD

Service: Load the specified workspace.
CICS/VS Routine: APLKLIBU, KLOAD
CMS Routine: APLSCLIB, SCLOAD
TSO Routine: APLYULIB, SCLOAD
VSPC Routine: APLPLIBS, PCLOAD

Value 8021 - YYCOPA

Service: Load copy source workspace and dispatch it to generate copy output.
CICS/VS Routine: APLKLIBU, KCOPA
CMS Routine: APLSCOPY, SCCOPA
TSO Routine: APLYUOPY, SCCOPA
VSPC Routine: APLPLIBS, PCCOPA

Value 0022 - YYSAVE

Service: Save a workspace.
CICS/VS Routine: APLKLIBU, KSAVE
CMS Routine: APLSCLIB, SCSAVE
TSO Routine: APLYULIB, SCSAVE
VSPC Routine: APLPLIBS, PCSAVE

Value 0023 - YYDROP

Service: Drop a workspace.
CICS/VS Routine: APLKLIBU, KDROP
CMS Routine: APLSCLIB, SCDROP
TSO Routine: APLYULIB, SCDROP
VSPC Routine: APLPLIBS, PCDROP

Value 0024 - YYLIB

Service: Send library and workspace information.
CICS/VS Routine: APLKLIBU, KLIB
CMS Routine: APLSCLIB, SCLIB
TSO Routine: APLYULIB, SCLIB
VSPC Routine: APLPLIBS, PCLIB

Value 8025 - YYCLEAR

Service: Clear the workspace.
CICS/VS Routine: APLKLIBU, KCLEAR
CMS Routine: APLSCMSC, SCCLEAR
TSO Routine: APLYUMSC, SCCLEAR
VSPC Routine: APLPLIBS, PCCLEAR

Value 0026 - YYWSID

Service: Change workspace identification.
CICS/VS Routine: APLKLIBU, KWSID
CMS Routine: APLSCMSC, SCWSID
TSO Routine: APLYUMSC, SCWSID
VSPC Routine: APLPLIBS, PCWSID

Value 0028 - YYPASS

Service: Change signon password.
CICS/VS Routine: APLKLIBU, KPASS
CMS Routine: APLSCMSC, SCPASS
TSO Routine: APLYUMSC, SCPASS
VSPC Routine: APLPLIBS, PCPASS

SHARED VARIABLE ORIENTED REQUESTS: Shared variable services are not performed by the executor but by the shared storage manager. The executor, however, provides the communication medium between the interpreter and the shared storage manager.

The parameter for each shared variable request is a control block, the processor control vector (PCV) for YYSON and YYSOFF, the share control vector (SCV) for all others. The interpreter builds the appropriate control block in WSMSVLRQ in the workspace. The executor places the shared storage manager return code in byte 0 of PTHSRCOD and the reason code in byte 1 of PTHSRCOD. In addition, the executor may generate this return code:

Return Code:

8000 Illegal: this workspace in copy status.

Value 0030 - YYSON

Service: Sign on to shared storage manager.
CICS/VS Routine: APLKISUI
CMS Routine: APLSCSHV, SCSVON
TSO Routine: APLYUSHV, SCSVON
VSPC Routine: APLPSHVR, PCSON

Value 0031 - YYSOFFER

Service: Offer to share a variable.
CICS/VS Routine: APLKISUI
CMS Routine: APLSCSHV, SCSVOFFR
TSO Routine: APLYUSHV, SCVOFFR
VSPC Routine: APLPSHVR, PCSOFFER

Value 0032 - YYSRET

Service: Retract a variable.
CICS/VS Routine: APLYUSHV, SCURETR
CMS Routine: APLSCSHV, SCSVRETR
TSO Routine: APLYUSHV, SCURETR
VSPC Routine: APLPSHVR, PCSRET

Value 0033 - YYSQUERY

Service: Send information about shared variables and partners.
CICS/VS Routine: APLYUSHV, SCSVQUER
CMS Routine: APLSCSHV, SCSVQUER
TSO Routine: APLYUSHV, SCSVQUER
VSPC Routine: APLPSHVR, PCSQUERY

Value 0034 - YYSACC

Service: Change access control vector.
CICS/VS Routine: APLYUSHV, SCSVACC
CMS Routine: APLSCSHV, SCSVACC
TSO Routine: APLYUSHV, SCSVACC
VSPC Routine: APLPSHVR, PCSACC

Value 0035 - YYSSPEC

Service: Accept new value for shared variable.
CICS/VS Routine: APLYUSHV, SCSVSPEC
CMS Routine: APLSCSHV, SCSVSPEC

TSO Routine: APLYUSHV, SCSVSPEC
VSPC Routine: APLPSHVR, PCSSPEC

Value 0036 - YYSREF

Service: Send current value of shared variable.
CICS/VS Routine: APLYUSHV, SCSVREF
CMS Routine: APLSCSHV, SCSVREF
TSO Routine: APLYUSHV, SCSVREF
VSPC Routine: APLPSHVR, PCSREF

Value 0037 - YYSCOPY

Service: Send current value of shared variable, regardless of access state.
CICS/VS Routine: APLYUSHV, SCSVCOPY
CMS Routine: APLSCSHV, SCSVCOPY
TSO Routine: APLYUSHV, SCSVCOPY
VSPC Routine: APLPSHVR, PCSCOPY

Value 0038 - YYSOFF

Service: Signoff of shared storage manager.
CICS/VS Routine: APLYUSHV, SCSVOFF
CMS Routine: APLSCSHV, SCSVOFF
TSO Routine: APLYUSHV, SCSVOFF
VSPC Routine: APLPSHVR, PCSOFF

IMPLIED SERVICE REQUESTS: Implied service requests are the means by which control is passed to the interpreter when an unpredictable event has occurred.

Value 80F0 - YYON

Event: User has signed on.

Return Codes:

0000 No continue

0001 Continue needed

Value 80F1 - YYPRGX

Event: Program check has occurred.
Return Codes: None

8. CONVERSION PROGRAM LINKAGE

The conversion program uses the conventions shown in Figure 51 for calls within itself.

Register	Use
0	Parameter (word of flag bits defined by macro APLFLAGS)
10	Address of source workspace (VS APL communication)
11	Address of sink (converted) workspace (VS APL communication)
	NOTE: Registers 10 and 11 are reversed for OS/VS1, OS/VS2, or DOS/VS communication.
12	Base register
13	Address of register save area
14	Return address
15	Entry point address and return code register

Figure 51. Conversion Program Register Usage

Save Areas

Registers are saved in an 18-word area pointed to by register 13.

Calling Macros

Two sets of calling macros are used in the conversion programs. The first consists of the ACENTRY macro and the OS/VS1, OS/VS2, DOS/VS CALL and RETURN macros. With these, APLFLAGS is used for communication between modules, but is not passed as a parameter. The second consists of the ACENTRY2, ACCALL, and ACEXIT macros. With these, APLFLAGS is passed as a parameter.

The ACENTRY macro instruction, which defines an entry point, is written as follows:

[<u>entry-pt</u>]	ACENTRY	<u>save-area</u>

<u>entry-pt</u>
is the name of the entry point.

<u>save-area</u>
is the name of the called routine's 18-word register save area.

Code generated for ACENTRY is described in Figure 52.

```
 CVINIT         ACENTRY   SAVEINIT
+CVINIT         CSECT
+               B         12(0,15)          BRANCH AROUND ID
+               DC        AL1(6)
+               DC        CL6'CVINIT'       IDENTIFIER
+               STM       14,12,12(13)      SAVE REGISTERS
+               BALR      R12,0             PROGRAM ADDRESSABILITY
+               USING     *,R12
+               ST        R13,SAVEINIT+4    SAVE CALLER SAVE AREA PTR
+               LR        R15,R13
+               LA        R13,SAVEINIT      PROGRAM SAVE AREA PTR
+               ST        R13,8(,R15)       BACK CHAIN NEW SAVE PTR
```

Figure 52. ACENTRY: Generated Code

The ACENTRY2 macro instruction, which defines an entry point, is written as follows:

[label]	**ACENTRY2**	save-area

entry-pt
is the name of the entry point.

save-area
is the name of the called routine's 18-word register save area.

Code generated for ACENTRY2 is described in Figure 53.

```
 CVDIRE         ACENTRY2  SAVDIRE
+CVDIRE         CSECT
+               B         12(0,15)          BRANCH AROUND ID
+               DC        AL1(6)
+               DC        CL6'CVDIRE'       IDENTIFIER
+               STM       14,12,12(13)      SAVE REGISTERS
+               BALR      R12,0             PROGRAM ADDRESSABILITY
+               USING     *,R12
+               ST        R13,SAVDIRER+4    SAVE CALLER SAVE AREA PTR
+               LR        R15,R13
+               LA        R13,SAVDIRER      PROGRAM SAVE AREA PTR
+               ST        R13,8(,R15)       BACK CHAIN NEW SAVE PTR
+               ST        R0,APLFLAGS       SAVE COMMUNICATION FLAGS
```

Figure 53. ACENTRY2: Generated Code

The ACCALL macro instruction, which calls a routine, is written as follows:

[label]	**ACCALL**	routine

label
is the optional statement label.

routine
is the name of the called routine as defined by ACENTRY or ACENTRY2.

Register 13 contains the address of the calling routine's save area.

Code generated for ACCALL is described in Figure 54.

```
.
           ACCALL    CVPARM
+          L         R0,APLFLAGS          SEND COMMUNICATIONS FLAGS
+          L         R15,=V(CVPARM)       GET ENTRY POINT
+          BALR      R14,R15              GO TO ROUTINE
+          ST        R0,APLFLAGS          SAVE RETURNED FLAGS
```

Figure 54. ACCALL: Generated Code

The ACEXIT macro instruction, which returns to the caller, is written as follows:

[label]	ACEXIT	RC=return-code

label
is the optional statement label.

return-code

Return Codes:

0 Normal return

4 or 8 Abnormal return; occurs only in communication between conversion program routines; specific meanings defined by each routine.

4 Workspace could not be converted; used in communication with OS/VS1, OS/VS2, or DOS/VS.

32 System error routine; used in communication with OS/VS1, OS/VS2, or DOS/VS.

Code generated for ACEXIT is described in Figure 55.

The conversion program also uses the APLCALL macro to call exarch and translator routines that are contained within it.

```
.
           ACEXIT    RC=0
+          L         R0,APLFLAGS          RETURN COMMUNICATION FLAGS
+          L         R13,4(,R13)          RESTORE CALLER'S SAVE PTR
+          L         R14,12(,R13)         RESTORE RETURN
+          LM        2,12,28(R13)         RESTORE CALLER REGS
+          LA        R15,0                RETURN CODE
+          BR        R14                  RETURN TO CALLER
```

Figure 55. ACEXIT: Generated Code

Control is passed from the interpreter to the shared storage manager as follows:

1. Interpreter issues a service request (APLSVCC) which passes control to the executor.

2. Under CMS or TSO, the executor passes control to the shared storage manager using registers 0, 14, and 15 as shown in Figure 56.

3. Under VSPC, the executor passes control to the shared storage manager using the host system linkage conventions.

   Under CMS or TSO, each auxiliary processor is an independent program. They initially receive control when VS APL is initialized. They subsequently receive control when the interpreter issues a shared variable request that satifies an auxiliary processor's wait. The auxiliary processor passes control to the shared storage manager using the ASVPxxxx macros. In all these cases, the linkage uses registers 0, 14, and 15 as shown in Figure 56.

   Under VSPC, the auxiliary processors are a collection of executor routines. They receive control, and pass control to each other, using the linkage conventions described in section "Executor Linkage." They pass control to the shared storage manager using the host system linkage conventions.

   Shared storage manager routines pass control to each other with register usage as shown in Figure 56.

Register	Use
0	Parameter and reason code (return)
12	Base register (all shared storage manager routines)
13	Address of shared memory
14	Return address
15	Entry point address and return code

Figure 56. Shared Storage Manager and Auxiliary Processor Register Usage

Save Areas

The shared storage manager has four save areas or levels reserved in the beginning of shared memory. Each shared storage manager routine operates on one of these levels in that, when it is called, the calling routine's registers are saved in the appropriate save area.

10. CICS/VS SHARED STORAGE MANAGER

Register Usage

The register usage implemented by the shared storage manager is shown in Figure 57.

Register	Use
9	Address of shared storage manager storage
10	Base register
11	Pointer to stack entry
12	Address of TCA
13	Address of CSA
14	Return address
15	Entry point address and return code

Figure 57. Shared Storage Manager Register Usage

Save Areas

The shared storage manager has a save area for saving caller registers 2 through 13 and a stack area for use by the shared storage manager subroutines. The APLKPROC, APLKSTAK, and APLKPOP macros are used to manage the stack area.

11. COMMON EXECUTOR LINKAGE

There is a set of service routines that produce the same effects in any operating system, but that vary their method of execution according to the operating system. For example, storage management services is called to get more virtual storage. Under TSO, it issues an OS GETMAIN; under CMS, it issues a DMSFREE; and under CICS/VS, it initiates a DFHSC request. These service routines are called cross-sytem executor services routines.

Cross-system executor services routines can use a special entry and exit logic for obtaining and returning dynamic work areas. The logic is called a stack protocol linkage, and an entry point requiring this linkage is a stack protocol entry point. A module calling a stack protocol entry point does not call the entry point directly, but transfers control to a stack processing routine in module APLXSTAK, which in turn calls the entry point.

Register Usage

Register	Use
0	Entry point to a stack protocol module
1	Standard parameter register
10	Pointer to PTX and PTH
13	Pointer to the stack work area (any unused register may be specified by the stack protocol module)
14	Pointer to the stack processor routine
15	Entry point of the stack routine

Save Areas

An 18-word save area is generated in the stack, and is used for other linkages as well as stack protocol linkages.

Calling Macros

There are three macros used by VS APL in stack protocol linkage: APLCALLS, APLXEND, and APLXPROC. These macros are expanded and described below.

The APLCALLS macro instruction, which calls a module that uses the stack protocol linkage, is written as follows:

[label]	APLCALLS	entry point [,PARM=parmlist][,STKPTR=reg]

label
    is an optional statement label.

entry point
    is the name of the entry point to be called. Entries from the cross-system vector table (VCT) will cause an assembler load from the VCT; otherwise, a literal VCON is loaded to fetch the target address.

PARM= parm-list
    specifies the parameter to be passed to the entry point. An LA instruction is generated for this parameter.

STKPTR= reg
    identifies the stack register for this stack call. If specified, this parameter sets the stack register for all future APLCALLS invocations.

Code generated for APLCALLS with various parameter options is described in Figure 58.

```
When entry point begins with VCT:
              APLCALLS  VCTENTRY
+             L         14,PTXVCT
+             L         0,VCTENTRY-VCT(,14)
+             LA        1,PARMLIST          IF PARM= IS USED

When entry point does not begin with VCT:
              APLCALLS  ENTRYPT
+             LR        15,RSTK             (OR REG, IF USED)
+             L         0=V(ENTRYPT)
+             LA        1,PARMLIST          IF PARM= IS USED
+             BALR      14,15
```

Figure 58. APLCALLS: Generated Code

The APLXEND macro instruction, which generates exit code for a module that uses the stack protocol linkage, is written as follows:

[*label*]	APLXEND	[ENTRY|FINAL|SUB] [CODE=*code-parm*]

label
is an optional statement label.

ENTRY
causes assembly of a branch instruction to the address in STKPREV.

FINAL
causes assembly of the exit code for the end of a module.

SUB
causes assembly of a branch instruction to the address in SUB14SV.

If neither ENTRY, FINAL, nor SUB is specified, the function performed is the function of both ENTRY and FINAL.

CODE=*code-parm*
causes assembly of an instruction to load a return code in register 15.

Code generated for APLXEND with various parameter options is described in Figure 59.

```
When ENTRY is specified:
               APLXEND    ENTRY CODE=CDPRM
+              L          R14,STKPREV
+              L          R14,STKPOP-STKENTRY(R14)
+              L          R15,CDPRM          IF CODE= IS USED
+              BR         R14

When FINAL is specified:
               APLXEND    FINAL
+STK           DSECT
+STKLEN        EQU        *-STKENTRY
+SYSECT        CSECT

When SUB is specified:
               APLXEND    SUB CODE=CDPRM
+              L          R14,SUB14SV
+              L          R15,CDPRM          IF CODE= IS USED
+              BR         R14
```

Figure 59. APLXEND: Generated Code

The APLXPROC macro instruction, which defines a stack protocol entry point, is written as follows:

```
[label]   APLXPROC   [ENTRY|SUB]
                      [,PASSB=(15,prgnm)]
                      [,DATAREG(drgnm)][,CODEREG(crgnm)]
```

label
: is an optional statement label.

ENTRY
: causes assembly of code for a stack protocol entry point in a module.

SUB
: is used for internal subroutine linkage for the assembler.

If neither ENTRY nor SUB is specified, the function performed is that of ENTRY.

PASSBK=(15,prgnm**)**
: specifies a list of registers to be returned unchanged. The default list is 15,0.

DATAREG=(drgnm**)**
: specifies a register to contain data. The default register is 13.

CODEREG=(crgnm**)**
: specifies a list of base registers. In assembler, there may be no more than five of these.

Code generated for APLENTRY with various parameter options is described in Figure 60.

```
When ENTRY is specified (first entry point):
LABEL          APLXPROC  ENTRY
+              CSECT     0D
+              USING     *,15
+              B         @PROLOG
+              DC        AL1(length of id)
+              DC        AL1(16*DRGNM+PRGNM)
+              DC        AL2(length of stack needed)
+              DC        CL8'LABEL'
+              DC        CL8'MM/DD/YY'
+              DROP      15
+@PROLOG       DS        0H                    LAST STMT IF CODEREG=0
+              BALR      CRGNM(1),0
+@PSTART       DS        0H
+              USING     @PSTART,CRGNM
+              LA        CRGNM(N),4095(,CRGNM(N-1))    IF CODEREG=
+              USING     @PSTART+4095*(N-1),CRGNM(N)   WAS USED

When ENTRY is specified (subsequent entry point):
LABEL          APLXPROC  SUB
+              DS        0H
+              USING     LABEL,@15
+              B         XPRINDX               XPRINDX HAS UNIQUE SUFFIX
+              DC        AL1(length of id)
+              DC        AL1(16*DRGNM+PRGNM)
+              DC        AL2(length of stack needed)
+              DC        CL8'LABEL'
+              DROP      15
+XPRINDX       DS        0H                    LAST STMT IF CODEREG=0
+              LR        CRGNM(1),15
+              LA        LABEL-@PSTART
+              SR        CRGNM(1),15
+              LA        CRGNM(N),4095(,CRGNM(N-1))

When SUB is specified:
               APLXPROC  SUB
+SUB14SV       SETC      unique save-area name
+STK           DSECT
+SUB14SV       DS        F
+SYSECT        CSECT
+<LABEL>       ST        R14,SUB14SV
```

Figure 60. APLXPROC: Generated Code

ERROR MESSAGE TO MODULE CROSS-REFERENCE INFORMATION

Figure 62 lists the message number, issuing module, and text for messages produced by the executor for error conditions arising within the processor itself. Under CICS/VS, CMS, and TSO, the error messages are displayed at the user's terminal. Under VSPC the errors are printed on the VSPC online log, and the following message is displayed at the terminal:

date time SYSTEM ERROR n

Here n corresponds to the online log error message number.

The broad source of the error message can be determined from the message identifier, according to the list in Figure 61.

Message Identifiers	Source of Messages
APLA000-APLA049	Auxiliary processors
APLC050-APLC073	TSO workspace conversion messages
APLK300-APLK349	CICS/VS executor
APLL350-APLL399	CICS/VS service program
APLM400-APLM499	Session manager
APLP500-APLP549	VSPC executor
APLS600-APLS699	CMS executor
APLW700-APLW749	GRAPHPAK workspace
APLW750-APLW799	TSO Service program workspace
APLY800-APLY949	TSO executor

Figure 61. Message Identifiers and Sources

```
************************************************************
*        AUXILIARY PROCESSOR MESSAGES                      *
************************************************************
```

Message ID	Issued by	Text
APLA000E	APLXAC	INVALID SIGNON PARAMETER LIST - APxxx.
APLA001S	APLXAC APLXAK	INSUFFICIENT STORAGE FOR SIGNON - APxxx.
APLA002S	APLXAC APLXAK	UNRECOVERABLE ERROR FROM SSM. TERMINATION OF APxxx. RC=xx RS=yy
APLA003E	APLXAC	ERROR IN TRANSLATE SERVICES - APxxx.
APLA004W	APLXAC	INVOCATION PARAMETERS EXCEEDED MAXIMUM - APxxx.
APLA005S	APLXAC APLXAK	ABEND. SHARED VARIABLE SET RETRACTED FOR APxxx.
APLA006E	APLYU100 APLYU101 APLYU111 APL100 APL101 APL110 APL111 APL123 APLXAK	AUXILIARY PROCESSOR APxxx ALREADY SIGNED ON.
APLA007I	APLYU100 APLYU101 APLYU111 APL100 APL101 APL110 APL111 APL123	APLxxx ABENDED AT xxxxxx, RETURN CODE IS xx, REASON CODE IS yy.
APLA020E	APL100KO	ICP GET FAILED: TCAICTR=X' '.
APLA021E	APL100KO	TRANSACTION NOT FOUND
APLA022E	APL100KO	INVALID TRANSACTION NAME
APLA023E	APL100KO	UNSUPPORTED TERMINAL TYPE
APLA024E	APL100KO	AP100 TWA SIZE IS TOO MSALL TO RUN TRANSACTION
APLA030I	APL100	CP/CMS COMMAND xxxxxxxx ABENDED, CODE=xxxx.
APLA040S	APLYUERR	ERROR OCCURRED IN AN AUXILIARY PROCESSOR.
APLA041E	APLYU101	AP101 STACK OVERFLOW, APL STACK PURGED.
APLA042E	APLYU101	AP101 STACK CLEARED DUE TO INVALID DATA.
APLA043S	APLYU111 APLYU210	APxxx IS BEING MISUSED, AP RC = xxxxxx.
APLA044E	APLYU210	UNUSUAL END TO FORMAT, xxxxxx RECORDS FORMATTED, DCB ABEND xxxxxx

Figure 62 (Part 1 of 13). Message-to-Module Cross-Reference

Message ID	Issued by	Text
APLA045S	APLYU100	TSO COMMAND xxxxxxxx ABEND SYSTEM CODE xxx, USER CODE xxx ISSUED BY CICS/VS SERVICE PROGRAM

```
**************************************************************
*        TSO WORKSPACE CONVERSION MESSAGES                   *
**************************************************************
```

Message ID	Issued by	Text
APLC050E	APLYUCNV	OPEN FAILED FOR FILE APLIN
APLC051I	APLYUCNV	OPEN FAILED FOR FILE SYSPRINT
APLC052E	APLYUCNV	INPUT PARAMETER INVALID
APLC053E	APLYUCNV	NO DDNAMES SUITABLE FOR OUTPUT WORKSPACE ALLOCATION
APLC054E	APLYUCNV	WORKSPACE HEADER SPECIFIES NO DATA RECORDS
APLC055E	APLYUCNV	HEADER SPECIFIES WORKSPACE SIZE OF ZERO
APLC056I	APLYUCNV	WILL TRY ALLOCATION ON NEXT VOLUME
APLC057E	APLYUCNV	CATALOG ERROR xxxx
APLC058E	APLYUCNV	SCRATCH DATA SET FAILED xxxx
APLC059I	APLYUCNV	DATA SET SCRATCHED
APLC060E	APLYUCNV	OPEN FAILED FOR WORKSPACE
APLC061I	APLYUCNV	WORKSPACE PROTECTION NOT IMPLEMENTED IN APLYUCNV
APLC062E	APLYUCNV	PREMATURE END OF DATA ON WORKSPACE INPUT FILE APLIN
APLC063E	APLYUCNV	UNCATALOG DATA SET FAILED xxxx
APLC064I	APLYUCNV	DATA SET UNCATALOGED
APLC065I	APLYUCNV	END OF WORKSPACE CONVERSION PROGRAM
APLC066I	APLYUCNV	SKIP TO NEXT WORKSPACE
APLC067I	APLYUCNV	WORKSPACE DATA SET NAME -- xxxx ON VOLUME xxxx
APLC068E	APLYUCNV	XXXX VOLUME DATA SET ALLOCATION FAILED
APLC069I	APLYUCNV	WORKSPACE CONVERTED. xxxx INPUT RECORDS. xxxx OUTPUT BLOCKS.
APLC070I	APLYUCNV	WORKSPACE NAME IS xxxx
APLC071W	APLYUCNV	OPEN FAILED FOR SYSIN. ONLY PUBLIC LIBRARIES WILL BE ALLOCATED.
APLC072E	APLYUCNV	SYNTAX ERROR. CARD IGNORED -- xxxx
APLC073E	APLYUCNV	MEMORY SHORTAGE. CARD IGNORED -- xxxx

Figure 62 (Part 2 of 13). Message-to-Module Cross-Reference

```
***********************************************************
*          CICS/VS EXECUTOR MESSAGES                      *
***********************************************************
```

Message ID	Issued by	Text
APLK300S	APLKASON	MAX USERS SIGNED ON
APLK301S	APLKASON	INCORRECT SIGNON
APLK302S	APLKASON	NUMBER NOT IN SYSTEM
APLK303S	APLKASON	NUMBER LOCKED OUT
APLK304S	APLKASON	NUMBER IN USER
APLK305S	APLKASON	FORCING OFF USER. TRY AGAIN
APLK306S	APLKASON	NO SIGNON MESSAGE AVAILABLE
APLK307S	APLKASON	SIGNON TERMINATED BY SYSTEM
APLK308S	APLKASON	CANNOT INITIALIZE APL

The following error message will occur:

4 - APLLIB CLOSE FAILED

8 - APLLIB OPEN FAILED

12 - APLDIR OR APLLIB READ FAILED

Message ID	Issued by	Text
APLK309S	APLKASON	TERMINAL NOT SUPPORTED BY APL
APLK310S	APLKLIBC	NO WS STORAGE AVAILABLE--SESSION TERMINATED

```
***********************************************************
*          CICS/VS SERVICE PROGRAM MESSAGES               *
***********************************************************
```

Message ID	Issued by	Text
APLL350S	APLKVMSG	INSUFFICIENT REAL OR VIRTUAL STORAGE AVAILABLE.
APLL351E	APLKVMSG	UNABLE TO OPEN APL LIBRARY.
APLL352E	APLKVMSG	UNKNOWN CONTROL STATEMENT TYPE.
APLL353E	APLKVMSG	INVALID REQUEST.
APLL354E	APLKVMSG	INVALID OPERAND.
APLL355E	APLKVMSG	DUPLICATE OPERAND.
APLL356E	APLKVMSG	CONFLICTING OPERANDS.
APLL357E	APLKVMSG	REQUIRED OPERAND NOT SPECIFIED.
APLL358E	APLKVMSG	INPUT APL LIBRARY REQUIRED BUT NOT SPECIFIED.

Figure 62 (Part 3 of 13). Message-to-Module Cross-Reference

Message ID	Issued by	Text
APLL359E	APLKVMSG	OUTPUT APL LIBRARY REQUIRED BUT NOT SPECIFIED.
APLL360E	APLKVMSG	UNABLE TO OPEN 'DDNAME'.
APLL361E	APLKVMSG	I/O ERROR IN 'DDNAME'; RETURN CODE=xx, REASON CODE=xxx.
APLL362E	APLKVMSG	OUTPUT APL LIBRARY 'DDNAME' FULL.
APLL363E	APLKVMSG	USER 'USERNAM' NOT FOUND IN INPUT APL LIBRARY.
APLL364E	APLKVMSG	USER 'USERNUM' NOT FOUND IN OUTPUT APL LIBRARY.
APLL365E	APLKVMSG	LIBRARY 'LIBNUM' NOT AVAILABLE.
APLL366E	APLKVMSG	LIBRARY 'LIBNUM' FULL.
APLL367W	APLKVMSG	USER 'USERNUM' REMOVED (NOT COPIED).
APLL368E	APLKVMSG	UNABLE TO IMPORT FILE 'FILENAME. SIZE IS xxxxxxxx K-BYTES.
APLL369E	APLKVMSG	WORKSPACE | FILE 'WORKSPACENAME' | 'FILENAME' NOT FOUND IN LIBRARY 'USERNUM'.
APLL370E	APLKVMSG	INVALID PASSWORD FOR FILE | WORKSPACE 'FILENAME' | 'WORKSPACENAME' IN LIBRARY 'LIBNUM'.
APLL371E	APLKVMSG	FILE NOT TRANSFERRED; INTERACTIVE PROGRAM IN USE.
APLL372I	APLKVMSG	FILE | WORKSPACE 'FILENAME' | 'WORKSPACENAME' REPLACED IN LIBRARY 'LIBNUM'.
APLL373W	APLKVMSG	FILE | WORKSPACE 'FILENAME' | 'WORKSPACENAME' EXISTS, TEMP NAME 'TEMPNAME' ASSIGNED FOR LIBRARY 'LIBNUM'.
APLL374E	APLKVMSG	FILE | WORKSPACE 'FILENAME' | 'WORKSPACENAME' EXISTS IN LIBRARY 'LIBNUM', TEMP NAME ASSIGNMENTS EXHAUSTED.
APLL375E	APLKVMSG	FILE | WORKSPACE 'FILENAME' | 'WORKSPACENAME' ALREADY EXISTS IN LIBRARY 'LIBNUM'.
APLL376E	APLKVMSG	FILE TYPE OF 'FILENAME | 'WORKSPACENAME' CONFLICTS WITH TYPE IN LIBRARY 'LIBNUM'.
APLL377I	APLKVMSG	COPIED FILE | WORKSPACE 'FILENAME' | 'WORKSPACENAME' TO LIBRARY 'LIBNUM'.
APLL378E	APLKVMSG	INPUT DATA SET FOR IMPORT HAS INVALID FORMAT.
APLL379E	APLKVMSG	OUTPUT DATA SET FOR EXPORT HAS INVALID FORMAT.
APLL380S	APLKVMSG	AUTH CONTROL STATEMENT ERROR.
APLL381E	APLKVMSG	AUTHORIZATION MISSING OR INVALID FOR THIS REQUEST.
APLL382S	APLKVMSG	AUTHORIZATION MISSING FOR LIBRARY FORMAT.
APLL383S	APLKVMSG	MODULE IKQVDTPE COULD NOT BE LOADED. PROGRAM TERMINATED.

Figure 62 (Part 4 of 13). Message-to-Module Cross-Reference

Message ID	Issued by	Text
APLL384I	APLKVMSG	RETURN CODE = xxxx.
APLL385I	APLKVMSG	END OF SERVICE PROGRAM JOB STEP.
APLL386I	APLKVMSG	HIGHEST RETURN CODE ENCOUNTERED = xxxx.
APLL387E	APLKVMSG	ERROR IN LIBRARY SERVICE PROGRAM.

```
*******************************************************
*         VS APL SESSION MANAGER MESSAGES             *
*******************************************************
```

Message ID	Issued by	Text
APLM401E	APLACXCM	COMMAND REJECTED BY EXIT
APLM402E	APLACOPY	COPY DESTINATION NOT AUTHORIZED
APLM403E	APLACOPY	COPY DESTINATION NOT FREE
APLM404E	APLACOPY	COPY DESTINATION NOT IN SERVICE
APLM405E	APLACOPY	COPY DESTINATION NOT SUPPORTED
APLM406E	APLACOPY	COPY DESTINATION NOT KNOWN
APLM407W	APLACOPY	COPY LIMIT EXCEEDED
APLM408W	APLACOPY	COPY QUEUE FULL, REQUEST ENDED
APLM409E	APLACOPY	COPY CODE UNKNOWN
APLM410E	APLACXCM	DUPLICATE OR CONFLICTING OPERANDS
APLM411I	APLACXCM APLALINE	END OF DATA REACHED
APLM412A	APLACPRO	ENTER PASSWORD
APLM413I	APLACPRO APLACXCM APLALINE APLACOPY	IN-STORAGE LOG FILE IN USE
APLM414E	APLACXCM	INVALID COMMAND NAME
APLM415E	APLACXCM APLALINE APLACOPY	INVALID, MISSING, OR EXTRA OPERANDSa
APLM416E	APLACOPY	I/O ERROR ON COPY DESTINATION
APLM417E	APLACPRO	INVALID PASSWORD. ENTER PASSWORD
APLM418E	APLACPRO	LINE EXCEEDS DISPLAY SIZE
APLM419E	APLACPRO	LINE NUMBERS EXHAUSTED
APLM420E	APLALINE APLACOPY	LINE NOT IN LOG FILE
APLM421E	APLACPRO	LOG FILE FULL

Figure 62 (Part 5 of 13). Message-to-Module Cross-Reference

Message ID	Issued by	Text
APLM422E	APLACOPY	NO COPY ID SPECIFIED
APLM423E	APLACXCM	NOT ENOUGH FREESPACE
APLM424E	APLACXCM	PROFILE FILE NOT FOUND
APLM425E	APLACXCM	DISPLAY CODE UNKNOWN
APLM426E	APLACQRY	PROFILE FILE I/O ERROR
APLM427E	APLACQRY APLACXCM	PROFILE FILE ATTRIBUTES INVALID
APLM428E	APLACQRY APLACXCM	PROFILE FILE NOT AVAILABLE
APLM429E	APLACOPY	COPY ID ALREADY EXISTS
APLM430E	APLACOPY	NOT WITH COPY ON
APLM431W	APLACPRO	LOG SIZE REDUCED
APLM432W	APLACXCM	LOG SIZE NOT AVAILABLE
APLM433W	APLACPRO	SESSION MANAGER RESTARTING DUE TO ERROR hh:mm:ss mm/dd/yy
APLM434W	APLACPRO	BEGGINING OF SESSION hh:mm:ss mm/dd/yy
APLM435S	APLACXCM	PROFILE RECORD ATTRIBUTES INVALID
APLM436S	APLACPRY	INTERNAL PROFILE PROCESSING ERROR

```
************************************************************
*          VSPC EXECUTOR MESSAGES                          *
************************************************************
```

Message ID	Issued by	Text
APLP500I	APLPCOEX	PGM CHECK LOOP IN APL PROCESSOR.
APLP501I	APLPCOEX	PGM CHECK IN EXECUTOR.
APLP502I	APLPCOEX	EXECUTOR SAVE AREA BLOCK FULL.
APLP503I	APLPCOEX	INVALID ASSIST CHECK CONDITION CODE.
APLP504I	APLPSERR	UNEXPECTED SYSTEM ERROR CODE RECEIVED. RET: xxx, REAS: yyy, SYS: www, APL:zzzzz.
APLP505I	APLPCOEX	NO SYSTEM INDICATOR ON ASYNCH ENTRY.
APLP506I	APLPCOEX	MULTIPLE UNEXPECTED ERROR CODES FROM SYSTEM.
APLP507I	APLPCOEX	APL ASSIST INCOMPATIBLE WITH APL PROCESSOR.
APLP508I	APLPCOEX	INSUFFICIENT STORAGE FOR MINIMUM WS.

Figure 62 (Part 6 of 13). Message-to-Module Cross-Reference

Message ID	Issued by	Text
APLP509I	APLPCOEX	PSW: xxxxxxxx xxxxxxxx, REGS:
APLP510I	APLPCOEX	UNEXPECTED ERROR IN APL INTERNAL AP.

```
*****************************************************************
*         CMS EXECUTOR MESSAGES                                 *
*****************************************************************
```

Message ID	Issued by	Text
APLS600I	APLSCINI	ERROR X INITIALIZING APL ASSIST. ASSIST NOT IN USE.
APLS601S	APLSCINI	ERROR X FROM SSM INITIALIZATION. SESSION TERMINATED.
APLS602I	APLSCINI	APL ASSIST INCOMPATIBLE WITH APL PROCESSOR. ASSIST NOT USED.
APLS603S	APLSCINI	ERROR WHILE GETTING SPACE FOR GLOBAL TABLE. OP=X,RC=Y.
APLS604W	APLSCINI	LIBRARY TABLE FILE NOT FOUND. NO PUBLIC OR PROJECT LIBS.
APLS605W	APLSCINI	ERROR X ON FSREAD OF LIB TABLE FILE. NO PUBLIC/PROJECT LIBS.
APLS606W	APLSCINI	SYNTAX ERROR X ON CARD Y OF LIB TABLE FILE. CARD IGNORED.
APLS607E	APLSCINI	AP NAME 'xxxxxx' INVALID OR TEXT FILE NOT FOUND.
APLS608I	APLSCINI	ERROR X IN DMSFRET DURING YYOFF,RC=Y.
APLS609I	APLSCINI	ERROR X FROM DMSFRET RETURNING STORAGE.
APLS610S	APLSCINI	INSUFFICIENT STORAGE TO INITIALIZE
APLS611W	APLSCINI	CANNOT LOAD AP--A-DISK IS NOT READ/WRITE
APLS612E	APLSCINI	UNKNOWN OPTION - xxxxxxxx.
APLS613E	APLSCINI	SYNTAX ERROR AT xxxxxxxx DURING xxxx.
APLS614E	APLSCINI	LENGTH ERROR AT xxxxxxxx DURING xxxx.
APLS615E	APLSCINI	INVALID VALUE - xxxxxxxx DURING xxxx.
APLS620E	APLSCSVI	PROCESSOR xxxx ABENDED WITH CODE(xxxx).
APLS630S	APLSCERR	PGM INTERRUPT LOOP IN APL PROCESSOR.
APLS631S	APLSCERR	PGM INTERRUPT IN EXECUTOR.
APLS632D	APLSCERR	TYPE 'DUMP 0-END' FOR DUMP, OR 'BEGIN' TO CONTINUE
APLS633I	APLSCERR	PSW=xxxxxxxx xxxxxxxx
APLS634I	APLSCERR	R0-7= xxx

Figure 62 (Part 7 of 13). Message-to-Module Cross-Reference

Message ID	Issued by	Text
APLS635I	APLSCERR	R8-15= xxx
APLS636S	APLSCERR	SAVE AREA OVERFLOW. CALLEE=xxxxxx, CALLER=yyyyyy.
APLS637S	APLSCERR	SYSTEM ERROR IN APL PROCESSOR.
APLS638I	APLSCERR	TYPE 'BEGIN xxxxxxx' TO TAKE WORKSPACE DUMP ON PRINTER.
APLS639A	APLSCERR	TYPE 'BEGIN' TO SKIP WORKSPACE DUMP.
APLS640I	APLSCERR	WORKSPACE AND PERTERM DUMP NUMBER X
APLS641I	APLSCERR	THIS IS DUMP OF ACTIVE WORKSPACE AREA
APLS642I	APLSCERR	THIS IS A DUMP OF THE PERTERM HEADER
APLS643W	APLSCERR	PROGRAM INTERRUPT IN SSM OR AUXILIARY PROCESSOR
APLS644E	APLSCERR	APL HAS ABENDED.
APLS650I	APLSCLIB	UNKNOWN RETURN CODE FROM CMS, OP=x, RC=y.
APLS651I	APLSCLIB	CMS FILE ERROR, OP=x, RC=y.
APLS652I	APLSCLIB	LIBRARY xxxx UNAVAILABLE, RC=y.
APLS653I	APLSCLIB	ERROR x FROM FST LOOKUP DURING YYLIB (RC=x).
APLS654I	APLSCLIB	ERROR x FROM DMSFRET FOR)LIB NAME TABLE (RC=x).
APLS655I	APLSCLIB	INTERNAL ERROR x LOADING WS FILE (OP=x).
APLS656I	APLSCLIB	FILE 'x APLTMPWS' ALREADY EXISTS. WS NOT SAVED.
APLS660I	APLSCOPY	ERROR DURING COPY, OP=x, RC=y.
APLS670E	APLSCSSI	APL MODULE (xxxxxxxx) NOT FOUND
APLS671E	APLSCSSI	DIAGNOSE 64 ERROR (CODE = xxx) WHILE LOADING APL SEGMENT - yyyy
APLS672E	APLSCSSI	APL CANNOT INITIALIZE - GLOBAL TABLE POINTER IN USE

```
************************************************************
*         WORKSPACE MESSAGES                               *
************************************************************
```

Message ID	Issued by	Text
APLW701E	GRAPHPAK	"AND" LENGTH ERROR
APLW702E	GRAPHPAK	RIGHT ARGUMENT OF "AND" MUST BE A HOMOGENEOUS GROUP
APLW703E	GRAPHPAK	IF RT ARG OF "AND" IS A "VS" GRP, LEFT CANNOT BE A MATRIX
APLW704E	GRAPHPAK	0 IS AN INVALID ATTRIBUTE PARAMETER
APLW705E	GRAPHPAK	ATTRIBUTE LENGTH ERROR

Figure 62 (Part 8 of 13). Message-to-Module Cross-Reference

Message ID	Issued by	Text
APLW706E	GRAPHPAK	NOT ABLE TO PRODUCE FUNCTION "FITFUN"
APLW707E	GRAPHPAK	CURSOR OUTSIDE OF WINDOW
APLW708W	GRAPHPAK	SYMBOL SETS DO NOT EXIST ON AUXILIARY STORAGE
APLW709E	GRAPHPAK	DEVICE NOT SUPPORTED
APLW710E	GRAPHPAK	ARGUMENTS OF "WITH" MUST BE OF OPPOSITE TYPE
APLW711E	GRAPHPAK	LEFT ARGUMENT OF "USE" MUST HAVE RANK LESS THAN 3
APLW712E	GRAPHPAK	LEFT ARGUMENT OF "USING" MUST BE A MATRIX
APLW713E	GRAPHPAK	"VS" ERROR
APLW714E	GRAPHPAK	CANNOT DO LOG PLOT OF NON-POSITIVE NUMBERS
APLW715E	GRAPHPAK	"B" ON LEFT ONLY POSSIBLE IN "STEP WITH 3 COLUMNS ON RIGHT
APLW721E	GRAPHPAK	GDDM AP RETURN CODE ERROR
APLW722E	GRAPHPAK	GDDM RETURN CODE ERROR:
APLW723W	GRAPHPAK	GDDM RETURN CODE WARNING
APLW724E	GRAPHPAK	SESSION MANAGER AP RETURN CODE ERROR
APLW725E	GRAPHPAK	GDDM AP NOT SHARING
APLW750I	SERVICE	'CONTROL STATEMENT'
APLW751E	SERVICE	UNABLE TO OPEN SYSIN
APLW752E	SERVICE	UNKNOWN CONTROL STATEMENT TYPE 'STMT'
APLW753E	SERVICE	INVALID OPERAND 'OPERAND'
APLW754E	SERVICE	REQUIRED OPERAND NOT SPECIFIED
APLW755E	SERVICE	UNABLE TO OPEN LIBRARY 'LIBRARY'
APLW756E	SERVICE	FILE ALREADY EXISTS
APLW757E	SERVICE	FILE DOES NOT EXIST
APLW758E	SERVICE	DATASET 'DDNAME' HAS INVALID FORMAT
APLW759I	SERVICE	LIBRARY 'LIBRARY' IS FULL
APLW760E	SERVICE	UNABLE TO OPEN DDNAME 'DDNAME'
APLW761W	SERVICE	FILE IS EMPTY
APLW762E	SERVICE	I/O ERROR IN DDNAME; RETURN CODE 'RC'
APLW763I	SERVICE	RECORD NUMBER 'RECNO'; SEGMENT NUMBER 'SEGNO'
APLW764I	SERVICE	FILE 'FILENAME' IMPORTED | EXPORTED | REPLACED LIBRARY 'LIB'
APLW765I	SERVICE	END OF SERVICE PROGRAM

Figure 62 (Part 9 of 13). Message-to-Module Cross-Reference

```
*****************************************************
*         TSO EXECUTOR MESSAGES                     *
*****************************************************
```

Message ID	Issued by	Text
APLY800I	APLYUINI	ERROR 1 INITIALIZING APL ASSIST. ASSIST NOT IN USE.
APLY801S	APLYUINI	ERROR CODE xx FROM SSM INITIALIZATON. SESSION TERMINATED.
APLY802I	APLYUINI	APL ASSIST INCOMPATIBLE WITH APL PROCESSOR. ASSIST NOT USED.
APLY803S	APLYUINI	YOU ARE NOT AUTHORIZED TO USE VSAPL FOR TSO.
APLY804I	APLYUINI	USING THE APL CHARACTER SET, ENTER OVERBAR - I.E. SHIFT-6
APLY805I	APLYUINI	NULL LINE, UNRECOGNIZED, OR TOO MANY CHARACTERS ENTERED
APLY806E	APLYUINI	UNRECOGNIZED TERMINAL CODE - x.
APLY807S	APLYUINI	UNRECOGNIED OPERAND - x.
APLY808S	APLYUINI	AMBIGUOUS OPERAND - x.
APLY809S	APLYUINI	REDUNDANT OPERAND - x.
APLY810S	APLYUINI	INVALID DSNAME - x.
APLY811S	APLYUINI	INVALID PASSWORD - x.
APLY812S	APLYUINI	ALLOCATION ERROR - RC=rc, DARC=darc, CTRC=ctrc - x.
APLY813S	APLYUINI	CONCATENATION FAILURE - RC=rc, DARC=darc - x.
APLY814S	APLYUINI	OPEN FAILURE - x.
APLY815S	APLYUINI	INVALID AUxILIARY PROCESSOR NAME - X.
APLY816S	APLYUINI	AUxILIARY PROCESSOR NOT FOUND - X.
APLY817S	APLYUINI	INVALID SIZE OPERAND - x.
APLY818W	APLYUINI	NO AUxILIARY PROCESSORS LOADED. VALUE IGNORED - X.
APLY819W	APLYUINI	LESS THAN MINIMUM SHRSIZE. VALUE IGNORED - x.
APLY820W	APLYUINI	LESS THAN MINIMUM WSSIZE. VALUE IGNORED - x.
APLY821W	APLYUINI	LESS THAN MINIMUM AISIZE. VALUE IGNORED - x.
APLY822S	APLYUINI	VIRTUAL STORAGE ALLOCATION ERROR.
APLY824W	APLYUINI	INVALID DEBUG OPTION - IGNORED.
APLY825W	APLYUINI	INVALID INPUT OPTION INVALID, ALL ENTRIES MUST BE IN QUOTES.
APLY826W	APLYUINI	INVALID PROFILE NAME - IGNORED.
APLY827W	APLYUINI	CONTINUE WS EXISTS BUT WILL NOT BE LOADED

Figure 62 (Part 10 of 13). Message-to-Module Cross-Reference

Message ID	Issued by	Text
APLY828W	APLYUINI	INVALID SMAPL OPTION - IGNORED.
APLY829W	APLYUINI	INPUT OPTION INTERNAL ERROR - SINK GETMAIN.
APLY830W	APLYUINI	APL101 ENCOUNTERED ERROR TRYING TO STACK INPUT DATA.
APLY831W	APLYUINI	INVALID HILIGHT OPTION. IGNORED.
APLY836S	APLYUERR	SAVE AREA OVERFLOW. CALLEE=xxxxxx, CALLER=xxxxxx.
APLY837S	APLYUERR	SYSTEM ERROR IN APL PROCESSOR.
APLY850E	APLYULIB	AN I/O ERROR HAS OCCURRED WHILE READING (WRITING) THE WORK DATA SET.
APLY851E	APLYULIB	ERROR DATA - CCHHR cccchhhhrr, CCW cc-aaaaaa-ffff-nnnn, CSWSTAT/COUNT nnnnnnnn, SENSE xxyy.
APLY852I	APLYULIB	YOU MAY DROP FROM OR SAVE INTO ONLY YOUR OWN LIBRARIES
APLY853I	APLYULIB	LIBRARY NOT FOUND
APLY854I	APLYULIB	THE LIBRARY IS EMPTY
APLY855W	APLYULIB	MORE WORKSPACE DATA SETS EXIST IN THIS LIBRARY THAN CAN BE LISTED.
APLY856E	APLYULIB	SYSTEM CATALOG SEARCH ERROR - DISPLAY ABORTED.
APLY857E	APLYULIB	THE WORKSPACE DATA SET DOES NOT CONTAIN A VALID APL WORKSPACE
APLY858E	APLYULIB	WORKSPACE SAVE DURING ABEND, USE)COPY
APLY859E	APLYULIB	WORKSPACE DATA SET OPEN FAILURE.
APLY860I	APLYULIB	THE WORKSPACE WAS SAVED BY userid
APLY861I	APLYULIB	THE WORKSPACE SIZE IS LARGER THAN THE AVAILABLE SPACE IN YOUR REGION.
APLY862E	APLYULIB	THE SIZE OF THE WORKSPACE TO BE LOADED IS SMALLER THAN THE SYSTEM DEFINED MINIMUM.
APLY863I	APLYULIB	THE USED PORTION OF THE WORKSPACE TO BE LOADED IS TOO LARGE.
APLY864E	APLYULIB	THE WORKSPACE DATA SET IS EMPTY.
APLY865E	APLYULIB	THE WORKSPACE SIZE IS LARGER THAN THE WORKSPACE DATA SET.
APLY866I	APLYULIB	YOU ARE NOT AUTHORIZED TO SAVE WORKSPACES IN THIS LIBRARY.
APLY867I	APLYULIB	THE WORKSPACE MUST BE NAMED BEFORE IT CAN BE SAVED.
APLY868I	APLYULIB	A CONTINUE WORKSPACE CAN ONLY BE SAVED VIA THE)CONTINUE COMMAND.

Figure 62 (Part 11 of 13). Message-to-Module Cross-Reference

Message ID	Issued by	Text
APLY869I	APLYULIB	YOU ARE CHANGING THE NAME OF YOUR WORKSPACE TO AN EXISTING NAME.
APLY870I	APLYULIB	IF THIS IS REALLY WHAT YOU WANT TO DO, THEN USE)WSID TO CHANGE THE WORKSPACE NAME BEFORE ATTEMPTING)SAVE.
APLY871E	APLYULIB	WORK DATA SET ALLOCATION FAILURE.
APLY872E	APLYULIB	WORK DATA SET OPEN FAILURE.
APLY873E	APLYULIB	WORK DATA SET TOO SMALL.
APLY874I	APLYULIB	MAXIMUM LIBRARY NUMBER EXCEEDED.
APLY875W	APLYULIB	MULTIPLE OWNERSHIPS OF THIS LIBRARY EXIST.
APLY876W	APLYULIB	ACCESS IS RESTRICTED TO THE FIRST, AND THAT IS CATALOGED UNDER xxxxxxxx.
APLY877E	APLYULIB	A SECONDARY ERROR HAS OCCURRED DURING LIBRARY CREATION DELETION PROCESSING.
APLY878E	APLYULIB	CREATION DELETION OF CATALOGED LIBRARY IDENTIFIER FAILURE - CATLG RC xx, SECONDARY RC yy.
APLY879I	APLYULIB	DATA SET NOT FOUND - xxxxxxxx.xxxxxxxx
APLY880I	APLYULIB	DATA SET NAMING CONFLICT.
APLY881I	APLYULIB	INVALID WORKSPACE NAME. WSID MUST CONTAIN NO MORE 8 CHARACTERS.
APLY882I	APLYULIB	WORKSPACE DATA SET NOT YET EXPIRED.
APLY883I	APLYULIB	THE WORKSPACE DATA SET IS IN USE BY SOMEONE ELSE. TRY AGAIN LATER.
APLY884I	APLYULIB	THE WORKSPACE DATA SET RESIDES ON A CURRENTLY UNAVAILABLE VOLUME.
APLY885E	APLYULIB	THERE IS INSUFFICIENT DIRECT ACCESS STORAGE SPACE TO SAVE THIS WORKSPACE.
APLY886E	APLYULIB	THE WORKSPACE DATA SET IS CATALOGED BUT NON-EXISTENT.
APLY887E	APLYULIB	DYNAMIC ALLOCATION FAILURE - FUNCTION CODE fc, DAIR RC dr, CATLG RC cr, DYNAM RC dynm.
APLY888W	APLYULIB	POSSIBLE DAMAGE TO YOUR WORKSPACE DATA SET. TRY TO SAVE WORKSPACE AGAIN.
APLY889W	APLYULIB	ERROR OCCURRED DURING ATTEMPT TO DROP THE CONTINUE WORKSPACE.
APLY890E	APLYULIB	SCRATCH FAILURE - CODE XX
APLY891E	APLYULIB	SCRATCH FAILURE - VOLUME volser, INCORRECT PASSWORD (CODE 08 - xx).
APLY892E	APLYULIB	UNCATALOG FAILURE - CODE xx.

Figure 62 (Part 12 of 13). Message-to-Module Cross-Reference

Message ID	Issued by	Text
APLY893E	APLYULIB	LIBRARY NOT EMPTY.
APLY894E	APLYULIB	YOUR PREFIX IS NOT DEFINED. DSNAMES REQUIRING IT CANNOT BE CONSTRUCTED.
APLY895E	APLYULIB	ISSUE THE PROFILE PREFIX COMMAND PRIOR TO THE VSAPL COMMAND.
APLY896E	APLYULIB	YOUR TSO USER IDENTIFICATION IS NOT DEFINED. ACCESS AUTHORITY CANNOT BE VERIFIED.
APLY897I	APLYUERR	PASSWORD PROTECTION NOT AVAILABLE IN THE SYSTEM
APLY910S	APLYUERR	ABEND - SYSTEM CODE - sys, USER CODE - usr.
APLY911S	APLYUERR	THE ATTEMPT TO SAVE A CONTINUE WORKSPACE HAS FAILED.
APLY920S	APLYUSVI	AUXILIARY PROCESSOR APxxxx ABENDED
APLY921E	APLYUSVI	UNRESOLVABLE SHARED VARIABLE INTERLOCK
APLY922E	APLYUSVI	COMMAND FAILED BY INSTALLATION EXIT
APLY923W	APLYUSVI	ABEND RECOVERY SET-UP FAILURE (CODE xx)
APLY924W	APLYUSVI	DYNAMIC ALLOCATION FAILURE -- NO DDNAMES FREE
APLY925W	APLYUSVI	TO EXIT VS APL, TYPE)OFF HOLD
APLY926W	APLYUSVI	UNRECOGNIZED CHARACTER OR TOO MANY CHARACTERS ENTERED

Figure 62 (Part 13 of 13). Message-to-Module Cross-Reference

UGH CODES

The translator and interpreter issue codes, called UGH codes, if a severe internal error condition occurs. Some unforeseen event may have arisen, and the workspace may have been damaged. Error recovery routines will clear the workspace, issue the message:

APLS637S SYSTEM ERROR IN APL PROCESSOR

and offer the user, in the case of CMS, the option of taking a snapshot dump of the workspace before resuming VS APL processing with a clear workspace. In the case of TSO, a dump is taken if you have allocated a dump data set. In the case of CICS/VS, a dump is automatically taken of the data set. In the case of VSPC, an automatic dump of the work space and associated control blocks is produced on the VSPC snap dump data set.

Regardless of the user's option, a mini-dump is produced at the terminal. See "How to Interpret the Terminal Mini-Dump."

The system error code for an interpreter or translator error contains the hexadecimal UGH code. The UGH codes appear in Figure 63.

Dec	Hex	Module	Reason
2	0002	APLIESCA	Result of dyadic operation is niladic or monadic function.
3	0003	APLI[illegible]CA	Result of monadic operation is dyadic function.
13	000D	APLIEFNM	On operation stack, the word following a branch statement entered in immediate the word following a branch statement entered in immediate execution is not a stop word.
50	0032	APLIEPSI	Illegal data type bits in argument block.
104	0068	APLITIDS	1. ITBLDID called with register CT (text length) is 0. 2. ITSTSRCH called with register PT (text address) not addressing an alphabetic. 3. ITSTSRCH computed space-available as sufficient, but IESFIND reported workspace full.
105	0069	APLITFCH	Invalid index parameter input to ITFETCH.
109	006D	APLITSUB	Invalid input to ITFNLNO; register 5 not a valid offset to body of function, or register 4 not the name of a function.
123	007B	APLITINP	O-U-T indicated after TYI or TYOI, but input line is blank.
124	007C	APLITINI	In newly loaded workspace, WSMFREEA and WSMFREEZ overlap.
125	007D	APLITCME	Input length is greater than 255, or buffer size is not 1024.
126	007E	APLITCME	Object to be erased has invalid syntax class.
127	007F	APLITCME	Error return from ITSTSRCH.
128	0080	APLITINI	On winding back the R13 stack following a program check, the level which issued the APLON macro is not found.
132	0084	APLITCMI	Name of unknown system variable found in function call block.
134	0086	APLITCMI	Unknown object found on operation stack.

Figure 63 (Part 1 of 3). Hexadecimal UGH Codes

Dec	Hex	Module	Reason
135	0087	APLITCMS	Number of system variable address table entries does not equal number of in-use entries.
136	0088	APLITINI	Unexpected reason code on error return from IATABREF.
137	0089	APLITCMT	Unknown return code from IASVOFF.
138	008A	APLITCML	Unknown reason code on error return from IASCOPY.
139	008B	APLITCMT	Control returned following call to ITCMOFF
140	008C	APLITINI	Error return from ITSHV.
152	0098	APLITERR	Invalid data found on stack when cleaning up after user error.
153	0099	APLITEX	Operation stack should contain "null, level" and it does not.
155	009B	APLITPRL	Undefined token found while preparing a statement for display.
159	009F	APLITCPI	1. Unknown syntax class found. 2. Unknown error condition; expect SI damage, stack full, workspace full, expect SI damage, stack full, workspace full, or symbol table full. 3. ITCOPIN computed space-available as sufficient, but IESFIND reported workspace full. 4. ITCOPIN knows that function line 0 is valid, but ITLINE0 reported invalid syntax. 5. ITCOPIN and ITOKENIZ disagree on number of labels in a function.
160	00A0	APLITCMC	Nonzero return code from YYCORZ.
161	00A1	APLIAQFN	1. Error return from IRPRLINE. 2. IAQCR computed space-available as sufficient, but IESFIND reported workspace full.
162	00A2	APLIAQFN	1. Error return from ITSTSRCH. 2. IAQFX and ITOKENIZ disagree on number of labels in a function.
164	00A4	APLITCMG	Internal name of group not found.
170	00AA	APLITFDC	Function statement should have a label but does not, or it should not have a label but does.

Figure 63 (Part 2 of 3). Hexadecimal UGH Codes

Dec	Hex	Module	Reason
171	00AB	APLITFDC	Address table entry for function is already in use.
175	00AF	APLITCPO	Invalid syntax class found during copy.
205	00CD	APLIASHV	Unexpected reason code on error return from YYSREF.
206	00CE	APLIASHV	Unexpected reason code on error return from YYSPEC.
208	00D0	APLIASHV	Unexpected reason code on error return from YYSRET.
210	00D2	APLIATRN	Error return from ITINPUT on quote-quad input.
211	00D3	APLIATRN	Invalid input to IAPLFUN; address of embedded VS APL function is zero.
212	00D4	APLIATRN	Incorrect internal name found in DN word of block by IATIDY.
213	00D5	APLIASHV	Reason code on error return from YYSOFF indicates that user is not signed on.
214	00D6	APLIASHV	Unexpected reason code on error return from YYSOFF.
216	00D8	APLIASHV	Unexpected reason code on error return from YYSON.
225	00E1	APLIASHF	Unexpected reason code on error return from YYSACC.
226	00E2	APLIASHF	Unexpected reason code on error return from YYSQUERY (while executing quad-SVC).
230	00E6	APLIASHF	Unexpected reason code on error return from YYSOFFER.
231	0 0E7	APLIASHF	Unexpected shared variable quota of 0 on normal return from YYSOFFER.
232	00E8	APLIASHF	Unexpected reason code on error return from YYSQUERY (while executing quad-SVO).
240	00F0	APLIASHF	Unexpected reason code on error return from YYSQUERY (while executing quad-SVQ).
251	00FB	APLIAGOU	Cursor unexpectedly exceeds line width.
270	010E	APLIAROT	AP vector routine entered during matrix rotation.

Figure 63 (Part 3 of 3). Hexadecimal UGH Codes

ABNORMAL TERMINATION AND DUMPS UNDER COMMON SERVICES OR APS

Under certain internal error conditions, VS APL modules will intentionally generate abends. These abends are trapped by VS APL in abend exits, and, in many cases, a dump is then taken by the abend exit. These intentional abends are described in Figure 64.

Abend Code	Issuing Module	Source of Error
1002	APLACPRO	APLACSF
1004	APLACPRO	APLACQUE
1010	APLACDSL	GDDM
1020	APLACXCM	GDDM
1022	APLACXCM	APLACSF Also generated if APLACOPY gets a bad RC from APLACSF (also APLALINE)
1030	APLACRDA	GDDM
1032	APLACRDA	APLACSF
1040	APLACRSA	GDDM
1050	APLACNDP	GDDM
1060	APLADMSG	GDDM
1070	APLACPRM	GDDM
1072	APLACPRM	APLACSF
1074	APLACPRM	APLACQUE
1081	APLACQRY	Environment-dependent code for profile input and output
1301	APLXSTAK	Stack requirement exceeds maximum stack available
2001	APLXAC	Sign off requested before successful sign on
2010	APL126	Abend in GDDX

Figure 64. Abends Intentionally Generated by VS APL

As a result of unexpected return codes or entry into abend exits, some components issue a dump of selected areas of storage. Figure 65 lists the dump codes given with these dumps, the module that caused the dump to be issued, and the areas dumped.

Dump Code	Issuing Module	Areas Dumped
CPRO	APLACPRO	Beginning of BND, DSM, area addressed by DSMYGM@
CRCP	APLACRCP	Beginning of BND, PTH/PTX, first stack block, first SMR
FYFL	APLXFYFL	FAB, registers, local stack
KAPS	APLXAK	Common control blocks followed by BND, 128-byte work area, APC, ECBs, SCVs, APLXAK work area
SSM and CAPS	APLXAC	BND, ANC, PCV, PCV ECB, MAI, error block DMP, ECB address list, SCV ECBs, APLXAC storage area, invocation parameters (if any), APCs, AP work areas
SSMK	APLXAK	Common control blocks followed by BND, 128-byte work area, APC, ECBs, SCVs, APLXAK work area
XGDA	APLXGCOM	BND, caller's GDM
XGDD	APLXGCOM	GST, GSTX, ANY active GSTPATs
XGDY	APLXGYC	GTS, caller's GDM
120X	APL120	BND, AP workarea
121X	APL121	BND, AP WORKAREA
A126	APL126	AP work area, BND

Figure 65. Common Dump Services Dumps and Issuing Modules

PROGRAM CHECKS AND DUMPS UNDER CICS/VS

DUMPS

General information on types of CICS/VS dumps, their content and format, and how to invoke them is contained in the CICS/VS Problem Determination Guide.

Two types of dumps are produced when VS APL is running under CICS/VS:

1. CICS/VS Formatted Dumps

   Formatted dumps may be requested by the CICS/VS master terminal operator, or may be produced automatically as a result of program checks or operating system abends. Frequently, the most useful information is in the CICS/VS internal trace table. (See CICS/VS Trace Information.)

   In addition, the APL, APLL, and APLT transactions each maintain an executor stack in the user extension to the CICS/VS TCA. Offsets to the stacks vary, but since the stacks contain EBCDIC routine IDs, visual identification of the stacks is normally straightforward.

   Program checks are a normal occurrence in the VS APL interpreter, so the FCT for the APLU transaction should never specify FDUMP= ASRA.

2. CICS/VS Storage Dumps

These may be produced as a result of any abend issued by CICS/VS (including ASRA and ASRB which are the secondary effects of program checks and system abends), or due to abends or dump requests issued by APL.

The APLU transaction attempts to recover from error conditions. However, if a VS APL system error is suspected, the transaction, before attempting recovery, issues a request to produce a storage dump.

In some cases, the storage dump request produced will have the same dump code as the CICS/VS abnormal termination code that alerted VS APL to the problem. However, the storage areas that are dumped will have been modified. In other cases, the storage dump request produced will have a unique VS APL dump code.

All CICS/VS storage dumps produced by APL are taken in module APLXDKMP, which dumps a series of common areas in addition to the individual storage segments explicitly requested for a given dump code.

The dumps produced by APLXDKMP for a given dump code contain the following:

a. A "DFHDC TYPE=PARTIAL" dump including the CSA, the TCA, the Trace table, and the particular storage segments specified in the request.

b. A "DFHDC TYPE=PARTIAL" dump of the GBL, if available.

c. A "DFHDC TYPE=PARTIAL" dump of the PRM, if available.

d. A "DFHDC TYPE=PARTIAL" dump of the user's PTH, PTX, PTK, and PRO, if available.

e. A "DFHDC TYPE=PARTIAL" dump of the user's SGN, if available.

f. If DEBUG(1) is on, a "DFHDC TYPE=PARTIAL" dump of shared storage.

g. A "DFHDC TYPE=PARTIAL" dump of the VCT, if available.

h. A DEBUG(1) is on, a "DFHDC TYPE=PARTIAL" dump of complete user's workspace, if available.

   or

   If DEBUG(1) is not on, a "DFHDC TYPE=PARTIAL" dump of the fixed-length beginning of user's workspace (as mapped by WSM) if available.

i. A "DFHDC TYPE=PARTIAL" dump of the transaction storage for the transaction in which the dump is being taken.

j. A "DFHDC TYPE=CICS" dump, which includes many of the tables used by CICS such as the PCT, the PPT, and the TCT.

k. And, if DEBUG(1) is on, a "DFHDC TYPE=PARTIAL" dump of program storage.

Because the DFHDC macro is invoked multiple times, multiple dumps are produced for each dump request. All of the dumps produced for a given dump request will have the requesting dump code. This, together with the sequence of areas dumped, allows the dumps for a given dump request to be identified.

Following are descriptions of the types of storage dumps and the information contained in them.

APLU Dumps with CICS/VS Abnormal Termination Codes

This type of dump will be produced if a dependent auxiliary processor abnormally terminates without having a defined abnormal termination exit routine or if that exit routine is unable to recover from the failure.

Transaction storage is dumped, but program storage is not. The global table, parm table, and first 4K bytes of the workspace are dumped as segment storage. On program checks, offset 288 (118) in the TCA contains the PSW and all register contents at the time the failure occurred.

APLU Dumps with a NXIT Dump Code

This type of dump will be produced for recursive errors. Transaction storage is dumped, but program storage is not. The global table, parm table, and first 4K bytes of the workspace are dumped as segment storage. On program checks, offset 288 (118) in the TCA contains the PSW and all register contents at the time the failure occurred.

After the dump is produced, the user will be forced to sign off VS APL.

APLU Dumps with an EXEC Dump Code

This type of dump will be produced if a problem occurs in handling YY service requests that is suspected to be a VS APL executor system error. Transaction storage is dumped, but program storage is not.

The global table, parm table, and first 4K of the workspace are dumped as segment storage. For program checks, offset 288 (118) in the TCA contains the PSW and all register contents at the time the program check occurred. For conditions other than program checks, the first word of the PSW contains a VS APL or CICS/VS abnormal termination code.

Possible VS APL abnormal termination codes and their meanings are:

FIXS The primary user task stack overflowed.

DSPS The dispatcher stack overflowed.

APLT The terminal transaction is not properly defined.

APLU Dumps with an NTRP Dump Code

This type of dump will be produced if a VS APL interpreter system error is suspected.

This dump consists of segment storage only. It contains the user's workspace and perterm (PTH, PTX, PTK, and PRO control blocks).

APLU Dumps with a REGS Dump Code

This is a 1-page dump containing register information that is taken when corresponding dump information is displayed at the user's terminal. It generally indicates that either an interpreter or executor system error occurred, and it is normally accompanied by an EXEC or NTRP dump.

Note that since this register dump contains all the information as it was formatted for display, only the right hand portion of the dump should be consulted.

APLU Dumps with a Knnn Dump Code

This dump is produced if an auxiliary processor has terminated abnormally. It indicates either that a program check has occurred within the auxiliary processor or that the host system requested the dump while performing a service for the auxiliary processor. In the dump code, 'nnn' is the numeric identifier of the auxiliary processor. see dumps) see dumps)

APLU Dumps with an nnnS Dump Code

This dump is produced if an auxiliary processor overflows its stack. In the dump code, 'nnn' is the numeric identifier of the auxiliary processor.

Other Dump Codes

Figure 66 is a list of the codes that may be received as a result of the execution of the DFHDC or DFHPC macros, with the names of the modules responsible for the abend and an indication of the possible cause. These are in addition to those in Figure 65.

Code	Module	Possible Cause
AICA	APLKADSP	Runaway task timer
AMTX	APLKADSP	(See CICS/VS Messages and Codes.)
AP*	APLKADSP	An auxiliary processor has a nonzero return code
APLS	APLKASTB	Bootstrap stack overflow
APLT	APLKTSRV	Nonzero return code from DFHIC
ASRA	APLKADSP	Program check
DSPS	APLKADSP	Despatcher stack overflow
ECBL	APLKADSP	Logic failure in processing ECBs
EXEC	APLKIFIX	Stack overflow
FIXS	APLKIFIX	Primary user task stack overflow
LIBS	APLKLIBG	Library task stack overflow
ICER	APLKEHCP	Unsuccessful I/O GET operation
ICIO	APLKEHCP	Unrecoverable I/O error
LENE	APLKEHCP	Inadequate TWA (record from GET was too long)
LIBE	APLKLIBV	Library Error
LIBT	APLKAGBL	Library termination failure
NOTR	APLKTSRV	Terminal out of service
NTRP	APLKMSCA	APL interpreter error (YY-dump)
NTWA	APLKTCTL	TWA inadequate for minimum stack
RDIR	APLKASON	Read directory error
RESM	APLKASTB	Work area full
RGRE	APLKAGBL	Global request element invalid
SSNA	APLKSSUB	Shared storage damage
SSTK	APLKSSUB	Stack overflow
STAK	APLKASON APLKEHCP APLKTCTL	Stack overflow
UNSP	APLKEHCP	Unsupported terminal type
XSGN	APLKASON	Signon table invalid
YOFF	APLKFSCL	Abend exit for KFOFF processing

Figure 66. Codes from DFHDC or DFHPC

CICS/VS TRACE INFORMATION

The CICS/VS executor issues "user 193" trace calls to CICS/VS using CICS/VS macro DFHTR.

Figure 67 shows the format of the rightmost two words in x'C1' (User 193) CICS/VS trace table entries. These two words are titled "Field A" and "Field B" in CICS/VS trace table listings. Following is an expanded description of some of the fields, indexed by the hexadecimal value in byte '+C'.

+ 8	+ 9	+ A	+ B	+ C	+ D	+ E	Creator	Function
Key type				x'80'	Key number		APLXGKT	Asynch. Input
Key type				x'81'	Key number		APLXGKT	Synch. Input
GDDM Request Code				x'82'	Not Used		APLXGKU APLXGKT	Calling GDDM
GDDM Error Code				x'83'	GDDM Severity Code		APLXGKT APLXGKU	GDDM returned
Abend Code				x'84'	Abend count		APLXGKT	In abend exit
Abend Code				x'85'	Abend count		APLXGKU	In abend exit
GDDX request code				x'86'	Return point		APLXGKR	APLXGKR called
GDDX error code				x'87'	GDDX severity Code		APLXGKR	APLXGKR returns
First 4 characters in name				x'88'	Last 3 characters in name		APLXGKU	GDDM PGM being released
Address of storage block				x'89'	Address of next storage block		APLXGKU	GDDM storage being released
Ecb Offset	Process number	AP Offset in Parm		x'A0'	Entry Point		APLKADSP	Dispatch
Wait type	Ecb/List pointer			x'A1'	Return Point		DPLKADSP	APLKWAIT
Parm value				x'A2'	Return Point		APLKADSP	APLKEXIT
Intrrpt code	PSW address			x'A3'	Entry Point		APLKADSP	Call Exit
YY code	Routine EP			x'A5'	WSMNSI		APLKIFIX	YYroute
ASYNC	YYcode	SRCOD		x'A6'	PTKPCOP,PMSK,MFLG		APLKIFIX	To Intrp
Return	MAI address			x'A7	Return point		APLXMKSG	APLXMKSG exit
TRQD TYP1	TRQD TYP2	TRQD OPT	x'00'	x'AA'	Return Point		APLKTREQ	APLKTERM

Figure 67 (Part 1 of 2). Format of CICS/VS Trace Table

+ 8	+ 9	+ A	+ B	+ C	+ D	+ E	Creator	Function
TSSRQ	TSS address			x'AB'	Return Point		APLKTSRV APLKTREQ APLKTRQO	TSRSCHED
WREQCD	TSS address			x'AC'	Write length		APLKTCWR	CTL Wrte
TSS address				x'AD'	AID	Read length	APLKTCTL	CTL Read
Not Used				x'AE'	TCT aid	Not used	APLKTCTL	Asynch. Input
Reg 15	ACBOFLG	ACBSTRN	ACBER-FLG	x'B0'	Return point		APLKVOPS APLKDOPS	Lib Open
Reg 15	RPLRTNC	RPLFDB2	RPLER-RCD	x'B1'	3-byte RBA		APLKVOPS APLKDOPS	Lib Get
Reg 15	RPLRTNC	RPLFDB2	RPLER-RCD	x'B2'	3-byte RBA		APLKVOPS APLKDOPS	Lib Put
Reg 15	ACBOFLG	ACBSTRN	ACBER-FLG	x'B3'	Return point		APLKVOPS APLKDOPS	Lib Close
GRELR-code	Type code	Return code		x'B4'	Perterm (PTH) address		APLKLIBG	APLKLIBG exit
SSM Request	Shared Var No.	Pershare Index		x'C3'	Return point		APLKSSVP	SSM Call
x'03'	x'00'	Return Code	Reason Code	x'C4'	Perproc location		APLKSSVP	SSM Exit
Rcode	Read length	Type	flags	x'D0'	Return point		APLKEMGR	Dest Mgr
Request byte	FAB address			x'F0'	Return Point		APLKLIBF	LIBF Call
FABLRCOD		FABSTAT		x'F1'	FABCRREC		APLKLIBF	LIBF ret
Check word				x'FF'	Intrrpt Code	ILC/CC/Pgm mask	APLKIFIX	Micro Code chk

Figure 67 (Part 2 of 2). Format of CICS/VS Trace Table

The contents of these fields, for values of '+C', are listed below.

x'80' in '+C'

Creator: APLXGKT.

Function: Traces asynchronous input when GDDM is being used to manage the user's terminal. Asynchronous input is input generated by one of the interrupt key at the terminal when APL is not waiting for input.

Field values:

- '+8': Key type:
  - x'00': ENTER
  - x'01': PF
  - x'02': Light pen

x'03': Badge reader

x'04': PA

x'05': CLEAR

x'06': Any other type of interrupt

'+D': Value, if any, associated with key type:

If key type=x'01', the key number in hex

If key type=x'03', x'00' (success), or x'01' (failure)

If key type=x'04', the key number in hex

x'81' in '+C'

Creator: APLXGKT.

Function: Traces synchronous input when GDDM is being used to manage the user's terminal. Synchronous input is input generated by one of the interrupt keys at the terminal when APL is waiting for input.

Field values:

'+8': Key type:

x'00': ENTER

x'01': PF

x'02': Light pen

x'03': Badge reader

x'04': PA

x'05': CLEAR

x'06': Any other type of interrupt

'+D': Value, if any, associated with key type:

If key type=x'01', the key number in hex

If key type=x'03', x'00' (success), or x'01' (failure)

If key type=x'04', the key number in hex

x'82' in '+C'

Creator: APLXGKT or APLXGKU

Function: One of these modules is about to call GDDM

Field values:

'+8': GDDM request code. See the GDDM User's Guide.

x'83' in '+C'

Creator: APLXGKT or APLXGKU

Function: Control has just returned from GDDM to one of these modules.

Field values:

'+8': GDDM error code. If nonzero, this code indentifies an error message listed in the GDDM User's Guide.

'+D': GDDM severity code. If the error code is non-zero, the severity of the error as returned by GDDM.

x'84' in '+C'

Creator: APLXGKT

Function: Traces abend codes trapped in the APLXGKT abend exit.

Field values:

'+8': The abend code trapped.

'+D': The number of abends trapped since APLXGKT last started processing a request from APLXGKU.

x'85' in '+C'

Creator: APLXGKU

Function: Traces abend codes trapped in the APLXGKU abend exit.

Field values:

'+8': The abend code trapped.

'+D': The number of abends trapped since APLXGKU last started processing a request from APLXGKRR.

x'86' in '+C'

Creator: APLXGKR

Function: Traces requests passed to APLXGKR from APLXGKRR.

Field values:

'+8': GDDX request code. The same as the GDDM request codes identified in the GDDM User's Guide, except that two additional codes are possible:

x'00000001': Initialize a GDDX path.

x'00000002': Terminate a GDDX path.

'+D': Return point in module calling APLXGKR.

x'87' in '+C'

Creator: APLXGKR

Function: Traces error and severity code about to be returned by APLXGKR to it's caller.

Field values:

'+8': GDDX error code. Meaning depends on value in the GDDX severity code field.

'+D': GDDX severity code.

If x'000000', request successfully processed.

If x'000001', an error has been detected by APL. The GDDX error code indicates what error has been detected, as defined in the mapping macro for the GDM request block.

If x'000002', an abend occurred and was trapped. The GDDX error code is the ABEND which was trapped.

If x'000004', x'000008', or x'00000C', then an error has occurred during a part of the processing which would be handled by GDDM if GDDM were controlling the session but has actually been handled by APL because GDDM(OFF) was specified when APL was invoked or because GDDM is not available. The error code identifies an error message from the GDDM User's Guide which explains the error that occurred.

x'88' in '+C'

Creator: APLXGKU

Function: Traces the names of the GDDM programs as they are released, when an abend occurs during GDDM termination processing and GDDM has not released all of its loaded programs.

Field values:

'+8': The first 4 characters of the 8-character program name.

'+D': The last 3 characters of the 8-character program name.

x'89' in '+C'

Creator: APLXGKU

Function: Traces the addresses of GDDM shared storage blocks as they are being freed, when an abend occurs during GDDM termination processing and GDDM has not freed all of its shared storage.

Field values:

x'A0' in '+C'.

'+8': ECB Offset

'+9': process number, which is equal to the DPD number

'+A': 2-byte AP offset into PARM

'+D': 3-byte EP for the module issuing APLKTRCE macro

x'A1' in '+C'.

'+8': APLKWAIT Codes:

x'80': On unless stop AP

x'40': Single ECB

x'20': APL ECB(s) only

x'10': System ECBs

'+9': 3-byte ECB/ECB list pointer

'+B': 3-byte Return address for module issuing APLKTRCE macro

x'A2' in '+C'.

'+8': APLKEXIT 3-byte Parm value

'+D': 3-byte return address for module issuing APLKTRCE macro x'A3' in '+C'.

'+8': CALLEXIT Interrupt Code

'+9': 3-byte Address from PSW

'+D': 3-byte EP for the module issuing APLKTRCE macro

x'A5' in '+C'.

'+8':

'+9': YYROUTE Code

'+A': 3-byte EP address for module issuing YYcode

'+D': 3-byte WSMNSI

x'A6' in '+C'.

'+8': ASYNC:

x'80' DATTN

x'40' QEND

x'20' CPULM

x'04' NOOUT

x'02' FOFF

x'01' ATTN

'+9': YY Code

'+A': 2-byte SRCOD

'+D': PTKPCOP

'+E': PMSK

'+F': PFLG

x'A7' in '+C'

Creator: APLXMKSG

Function: Traces calls to common main storage services under CICS/VS.

Field values:

'+8': Return code.

'+9': Address of MAI for request.

'+D': Return point in calling routine.

x'AA' in '+C'.

'+8': APLKTERM TYP1 Codes:

x'01' Format

x'02' Write

x'03' Read

x'05' Getdata

x'06' Fldattr

'+9': TYP2 Codes

'+A': OPT

'+B': x'00'

'+D': 3-byte return address for the module issuing APLKTRCE macro

x'AB' in '+C'.

'+8': TSSRQ:

x'80' Pending Format

x'40' Write

x'20' Read

x'10' New Fld Attr

x'08' Alarm Pending

x'04' Set Cursor

x'02' Restore

x'01' Hardcopy

'+9': 3-byte TSS address

'+D': 3-byte return address for the module issuing APLKTRCE macro

x'AC' in '+C'.

'+8': Control Write WREQCD Codes:

x'80' Restore in listen

x'40' Any form restore

x'20' APL task waiting

x'08' Normal schedule

'+9': 3-byte TSS address

'+D': 3-byte write length

x'AD' in '+C'.

'+8': Control Read 4-byte TSS address

'+D': AID Codes:

x'7D' Enter

x'6D' Clear

x'6C' PA1

x'6E' PA2

x'F1-F10' PF keys 1-10

'+D': 2-byte read length

X'AE' in '+C'

Creator: APLKTCTL

Function: Traces asynchronous input when GDDM is not being used to manage the user's terminal. Asynchronous input is input generated by one of the interrupt keys at the terminal when APL is not waiting for input.

Field values:

'+D': TCT AID byte. See the one of the CICS/VS application programmer's reference guides.

X'B0' in '+C'.

'+8': R15 from Library Open

'+9': ACBOFLG byte

'+A': ACBSTRN byte

'+B': ACBERFLG byte

'+D': 3-byte return address for the module issuing APLKTRCE macro

X'B1' in '+C'.

'+8': R15 from Library Get

'+9': RPLRTNC byte

'+A': RPLFDB2 byte

'+B': RPLERRCD byte

'+D': 3-byte RBA

X'B2' in '+C'.

'+8': R15 from Library Put

'+9': RPLRTNC byte

'+A': RPLFDB2 byte

'+B': RPLERRCD byte

'+D': 3-byte RBA

X'B3' in '+C'.

'+8': R15 from Library Close

'+9': ACBOFLG byte

'+A': ACBSTRN byte

'+B': ACBERFLG byte

'+D': 3-byte return address for the module issuing APLKTRCE macro

X'B4' in '+C'

Creator: APLKLIBG

Function: Traces library requests.

Field values:

'+8': GRELRCOD (2nd byte).

'+9': APLKG type codes:

x'00' Load

x'01' Save

x'02' Drop

x'03' WDIR

x'04' WLIB

x'05' UDIR

x'06' CFIL

x'07' WFIL

x'08' UFIL

x'09' RLIB

'+A': 2-byte return code.

'+D': Perterm (PTH) address.

x'C3' in '+C'.

'+8': SSM Call Request Byte:

x'00' Cleanup

x'01' ACC

x'02' CPY

x'03' OFR

x'04' QRY

x'05' REF

x'06' RET

x'07' SOF

x'08' SON

x'09' SPC

x'0A' ACHK

'+9': Shared variable number

'+A': 2-byte x'FFFF' minus shared variable number

'+D': 3-byte return address for the module issuing APLKTRCE macro

x'C4' in '+C'.

'+8': x'03'

'+9': x'00'

'+A': Return Code

'+B': Reason Code

'+D': 3-byte PERPROC location

x'D0' in '+C'.

'+8': Destination manager return code

'+9': Rlen byte

'+A': Type byte

'+B': Flags byte

'+D': 3-byte return address for the module issuing APLKTRCE
Set Timer PTKMFLG:

x'80' MXUSE

x'40' IWAIT

x'20' MINEX

x'10' INTRP

x'08' MNDMPF

x'04' MEXA

x'02' TIMEO

x'01' MTPOP

'+9': 3-byte exit address (replaced)

'+D': 3-byte return address

x'F0' in '+C'.

'+8': LIBF Call Req. byte:

x'01' OPSW

x'02' OPSR

x'03' OPDR

x'04' CSEQ

x'05' CDIR

x'06' DEL

x'07' SHRY

x'08' SHRN

x'09' CFSZ

x'0A' SEQW

x'0B' SEQR

x'0C' DIRU

x'0D' DIRD

x'0E' PWCH

x'0F' CLOS

'+9': 3-byte FAB address

'+D': 3-byte return address for the module issuing APLKTRCE macro

X'F1' in '+C'.

'+8': LIBF Return FABLRCOD (2-bytes)

'+A': 2-byte FABSTAT

'+D': 3-byte FABCRREC

X'FF' in '+C'.

'+8': Microcode check word (4 bytes)

'+D': INT code

'+E': 2-byte ILC, CC, PGM mask

PROGRAM CHECKS AND DUMPS UNDER CMS

If a severe error of unexpected nature occurs, the VS APL processor or CMS routines receive control, perform limited error handling, produce messages, and provide dumps either automatically or at user option. The following information is useful in interpreting these diagnostics.

DURING INITIALIZATION OF THE VS APL PROCESSOR

If a severe error occurs during initialization of VS APL, the APL008I error message is printed at the terminal, abnormal termination occurs, and a dump is automatically taken of all of virtual storage. The STAE exit routine produces the dump by simulating a DUMP 0-END CP command.

Contents of the Dump

In the dump, register 1 points to the STAE work area. This area contains the abend code, PSW, and general registers at abnormal termination. The abend codes issued for VS APL appear in Figure 68.

Code	Meaning
1	VS APL initialization has discovered an error. A previous message has explained the problem.
2	Same as user code 1, except that because of the nature of the error the VS APL processor was unable to type an error message.
1xx	An unexpected program check caused the abnormal termination. xx is the decimal program check code.
system	CMS has invoked the abnormal termination.

Figure 68. VS APL Abend Codes

Additional Information for Program Check

If a program check was responsible for the abnormal termination (user code 1xx), message APL018I is printed at the terminal. It contains the address and program check code from the program check PSW. Register 2, at the time of the abnormal termination, contains the address of the PIE. The PIE in turn contains the program check PSW and registers 14 through 2 as they were at the time of the program check. Registers 3 through 13 at the time of the program check are stored as registers 3 through 13 in the STAE work area.

AFTER INITIALIZATION

If an error occurs during operation of the VS APL processor, error messages are issued, and dumps may be produced at the system printer at the user's option.

System Error in the Interpreter or Translator

When a system error occurs in the interpreter, messages APL108I, APL109I, and APL110D are issued. They identify the error as a system error and prompt the user on what action to take to produce a dump. If the dump is taken, it contains a dump of the active workspace area and the PERTERM header block. Headings within the dump explain its contents. See also "How to Interpret the Terminal Mini-Dump" and "How to Interpret the Snapshot Workspace Dump Produced at the Printer."

Whether a dump is taken or not, a system error message is issued at the terminal. It contains the PSW and register information.

Program Check in the Executor

When a program check occurs in the executor, messages APL102I, APL104I, APL105I, and APL106I are issued at the terminal. These messages provide information about the PSW and registers at the time of the program check. The processor forces an abnormal termination with an ABEND code of 1xx, where xx is the program interrupt code.

After abnormal termination, a STAE exit routine receives control. This routine issues message APL115I and prompts the user with message APL103D on how to produce a full storage dump at the printer. The full storage dump contains the contents of the active registers at the time of the interrupt. The registers contain the following information:

Register 2 contains the address of the 104-byte STAE work area. The format of the STAE work area is described in OS/VS2 System Programming Library: Supervisor. The work area contains the registers and PSW at the time of abnormal termination.

Register 8 in the dump contains the ABEND code. The ABEND code is either the hexadecimal equivalent of decimal 1xx (program check) or 001 (for abnormal termination issued by the executor). For the latter case, the previous message has explained the problem.

Register 10 contains the address of the executor global table.

Register 11 contains the address of the active workspace area.

Program Interrupt in the Shared Storage Manager or Auxiliary Processor

If program interrrupt has occurred in the shared storage manager or one of the auxiliary processors, the processor issues message APL114I. The error is handled in the same manner as an interpreter or translator system error (described above), except that the user is prompted to request a full storage dump instead of a snapshot workspace dump. If a dump is taken, it contains the contents of the registers at the time of the interrupt. Register 10 in the dump points to the global table, which contains the addresses of the auxiliary processors, shared memory, and the auxiliary processor work areas.

Abnormal Termination in the Executor

If abnormal termination occurs in the executor, the messages issued and STAE exit routine processing are the same as for Program Check in the Executor (see above).

Program Check Loop in the VS APL Processor

During error recovery, a second program check may occur in the processor. In this case, the processor issues messages APL101I, APL104I, APL105I, and APL106I. Then it prints at the terminal the PSW and the contents of the registers at the time of the second program check. Then it issues message APL115I. To obtain a full storage dump in this case, follow the procedure described in "How to Produce a Dump."

This dump will contain information about the first and second program checks that have occurred. The registers and PSW that are printed are those at the time of the second program check.

To obtain information about the first program check, find the address of the global table at absolute address X'440' in NUCON. The global table contains the address of the WSM at a displacement of X'48'. WSMREGSV contains the contents of registers 0 through 15. The doubleword at WSMPCPSW contains the PSW at the time of the first program check.

HOW TO PRODUCE A DUMP

Three types of dumps are possible: an ordinary full system dump of all of virtual storage, a snapshot dump of the active workspace, or a mini-dump of the registers and other information.

In certain cases the processor prompts the user on which dump to request and how to request it. If, however, the processor loses control, the user may request a full system dump, using the facilities of CP, in the manner described below.

Full System Dump

For a full system dump, type:

```
DUMP 0-END
CLOSE PRINT
BEGIN
```

Snapshot Workspace Dump

For a snapshot workspace dump, type:

```
BEGIN xxxxxx
```

where xxxxxx is an address provided in the prompt line that appears at the terminal after a system error has occurred in the VS APL processor.

The mini-dump is produced automatically at the terminal for certain system errors.

Sample Prompting Sequence

An example of the prompting sequence is shown in Figure 69.

```
     APL108I SYSTEM ERROR IN APL PROCESSOR.

To take a workspace dump on the printer, type:

   BEGIN 0216AE

To skip a workspace dump, type:

  BEGIN
  ADSTOP AT 17728E
  CP

If you want a snapshot dump of the active workspace, type at the keyboard:

  BEGIN 216AE

The system will respond like this:

  DUMPING LOC 050000
  DUMPING LOC 060000
  DUMPING LOC 070000
  DUMPING LOC 080000
  DUMPING LOC 090000
  DUMPING LOC 0A0000
  DUMPING LOC 0B0000
  DUMPING LOC 0C0000
  DUMPING LOC 0D0000
  DUMPING LOC 0E0000
  DUMPING LOC 0F0000
  DUMPING LOC 100000
  DUMPING LOC 110000
  DUMPING LOC 120000
  DUMPING LOC 130000
  DUMPING LOC 140000
  DUMPING LOC 150000
  DUMPING LOC 160000
  DUMPING LOC 170000
  COMMAND COMPLETE
  COMMAND COMPLETE

The snapshot workspace dump will be produced at the system printer.
Following the above response is a mini-dump, as described below.

If you do not want the snapshot workspace dump, simply type:

  BEGIN

The system will respond with a mini-dump like this:

10:31:16 04/11/81 SYSTEM ERROR 00 0002 0000 0001 5C02D9DC

00000000 00000000 00000010 0012C15E 000008CC 00000028 2B04A9A0 0002ECA2

00000008 00000020 00000004 00049000 0002CFD8 00176C60 00176C80 5C02D9C4

42FC6000 00000000 00000000 00000000 00000000 00000000 000000000

CLEAR WS
```

Figure 69. Sample Prompting Sequence

HOW TO INTERPRET THE TERMINAL MINI-DUMP

The terminal mini-dump consists of five lines. The first line contains the time and date:

```
10:31:16 04/11/81
```

the indication:

```
SYSTEM ERROR
```

two characters of meaningless data:

```
00
```

the dump number:

```
0002
```

the system error code:

```
0000 0001
```

and the right half of the PSW (the address of the instruction where the error was detected).

The second and third lines contain the contents of registers 0 through 15 at the time the error occurred.

The fourth line contains the contents of the floating-point registers.

The fifth line contains the message:

```
CLEAR WS
```

indicating that the user's workspace has been cleared and is ready for new input.

How to Determine the Type of VS APL System Error

There are three types of system errors for the VS APL processor; they may be distinguished by examining the system error code.

The three types appear in Figure 70.

Error Code	Meaning
0000 xxxx	If the first word of the system error code is 0000, it is a system error in the interpreter or translator. xxxx is the hexadecimal UGH code. For a description of the UGH codes, see Figure 63.
nnnn 8000	If the second word is 8000 (first bit is 1), it is a system error in the executor. nnnn is the service request code value. The service request codes are listed under "Service Request Calls" in "Linkage Conventions" at the beginning of this section.
xxxx yyyy	Program check. The two words are the left half of the PSW, where yyyy is the program check code as documented in IBM System/370 Principles of Operation.

Figure 70. VS APL Processor System Errors

HOW TO INTERPRET THE SNAPSHOT WORKSPACE DUMP PRODUCED AT THE PRINTER

The snapshot workspace dump contains the contents of the active workspace area and the PERTERM header. The dump of the active workspace contains: the general and floating-point registers of the executor at the time the dump was requested (not relevant for debugging purposes), the keys (always provided by CP in a dump), the executor's PSW (not relevant), and the contents of the workspace.

Where to Find Information in the Snapshot Workspace Dump and the Mini-Dump

Register 10 in the dump contains the address of the executor global table. With the aid of the global table format in "Data Areas," the global table can be used to locate the WSM and other control blocks useful for debugging.

HOW TO LOCATE THE TOP TOKEN ON THE OPERATION STACK: The top token on the operation stack is at the address in WSMTSADR plus four bytes. WSMTSADR is at the address contained in register 11 plus X'0954' bytes.

HOW TO DETERMINE THE ROUTINE/MODULE WHERE THE ERROR OCCURRED: Obtain a link edit map. It shows the load address of each module. In the mini-dump produced at the terminal, register 12 contains the address of the routine/module (CSECT) that was functioning when the dump occurred.

PROGRAM CHECKS AND DUMPS UNDER TSO

When a severe error occurs under TSO, the following actions are taken:

1. If the data set APLDUMP is allocated, a dump is taken.
2. The command)CONTINUE is issued.
3. The command)OFF HOLD is issued.

ABNORMAL TERMINATION/SYSTEM ERROR/PROGRAM CHECK UNDER VSPC

When a severe error in the VS APL processor occurs, the processor or VSPC routines receive control, perform limited error handling, produce messages, and provide dumps.

When a severe error occurs in the executor, one of the above messages is logged, a dump is taken, and VS APL is terminated abnormally. The dump is a VSPC dump as described in <u>VS Personal Computing (VSPC) Program Logic</u>.

When a severe error occurs in the interpreter or translator, a mini-dump is printed at the terminal. See "How to Interpret the Terminal Mini-Dump," above. All dumps and the system log are sent to the operator. In all cases, a 'SYSTEM ERROR' message is received.

When an error occurs during operation of the VS APL processor, error messages are issued, and dumps are produced at the system printer.

SYSTEM ERROR IN THE INTERPRETER OR TRANSLATOR: Errors in the interpreter or translator that cannot be handled by these routines will produce the common CMS/VSPC message at the terminal: 'xx SYSTEM ERROR xyz - REGS xx' followed by clear workspace. A dump and a log message are produced.

VS APL MICROCODE ASSIST

Some systems use the VS APL microcode assist. If an error persists that may involve this assist (or its software interface), the following can be used to determine this involvement: Perform the same VS APL procedure that is causing the error, without using the assist. (The test for microcode can be cancelled by an option of the DEBUG operand of the APL command.) If the error does not occur, it is probably in the assist or in the VS APL software interface (although it may be a user error); if the error continues to occur, it is probably not in either the assist or the VS APL software interface.

DEBUG OPERAND OF THE APL COMMAND

DEBUG is an optional operand of the APL or VSAPL command; it alters the normal error recovery actions of VS APL so that abnormal operating situations may be recorded and isolated for debugging. For a description of its options and their effects, see VS APL for CMS: Terminal User's Guide or VS APL for TSO: Terminal User's Guide.

INFORMATION NEEDED FOR PROBLEM DETERMINATION AND DIAGNOSIS

If you submit an APAR or contact IBM central service about an apparent error in the VS APL processor, you will be asked to supply information that is needed to diagnose and correct problems. Please be ready to do the following:

1. Identify the operating system, with the versions and release levels that apply; for example:
   - VM/370, CMS Version 2, PLC15
   - OS/VS2 MVS, Release 3.8, TSO
   - DOS/VSE, Release 36, CICS/VS Release 1.5.0
2. Identify the VS APL release level; for example: VS APL, Release 4.0.
3. Identify the processor; for example: S/370, Model 145.
4. Tell whether or not your processor has the APL microcode assist feature; if it has, give its Engineering Change (EC) level. Tell also whether the error is in VS APL or in the microcode assist. (For suggestions, see the sections above on "VS APL Microcode Assist" and "DEBUG Operand of the APL Command.")
5. Describe any modifications made to VS APL by your installation. Tell the names of object modules and routines that have been modified locally.
6. Tell how reproducible the error is:
   - Can it be reproduced always?
   - Can it be reproduced only sometimes?
   - Have you not been able to reproduce it?

   If the error is reproducible, reproduce it in the most direct way possible. For example, reduce the number of statements within a user-defined function to the fewest needed to cause the error to occur.

7. Identify and describe any auxiliary processors that were active when the error occurred.

8. If possible, provide a printout of a terminal session showing the error and how to reproduce it.

9. Provide a current linkage editor map of all VS APL load modules. (This map is generated when VS APL is installed and when maintenance updates are made.)

10. If the error is a system error or an abnormal termination, provide a dump of the active workspace.

11. For VSPC, provide a copy of the user profile.

12. For CICS/VS, if an abend occurred, provide a listing of the CICS/VS dump data set. (If the error occurs in the VS APL APLT or APLX transaction, take a CICS/VS partition dump; in other cases, a CICS/VS Snap dump will suffice.

13. For CICS/VS, provide a CICS/VS auxiliary trace when the CICS/VS incore trace does not show the source of a problem and the error appears to be in a VS APL executor module.

(Names of individual entry points can be easily found in Section 3. Program Organization, organized in alphabetic order.

Entry points and modules sorted either by module name or by entry point can be found in Section 4. Directory.

Because of their ease of search, entry points and module names are excluded from this index.)

A

B

C

F

G

H

I

L

W

LY20-8032-3

VS APL Program Logic (File No. S370-22) Printed in U.S.A. LY20-8032-3

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

VS APL Program Logic
LY20-8032-3

Reader's Comment Form

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL ____________________

Previous TNL ____________________

Previous TNL ____________________

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Thank you for your cooperation.

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

LY20-8032-3

Reader's Comment Form

Fold and tape   Please do not staple   Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS   PERMIT NO. 40   ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape   Please do not staple   Fold and tape

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

VS APL Program Logic (File No. S370-22) Printed in U.S.A. LY20-8032-3