AID V3.2A

Core Manual

Edition November 2006

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Fax forms for sending us your comments are included at the back of the manual.

There you will also find the addresses of the relevant User Documentation Department.

Certified documentation according to DIN EN ISO 9001:2000

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2000.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright and Trademarks

Contents

1 Preface

AID (Advanced Interactive Debugger) is a powerful interactive debugging aid which runs under the operating system BS2000. AID V2.0A can be used as of BS2000 V9.5. Error diagnosis, debugging and preliminary recovery for all programs created under BS2000 are much shorter and easier with AID compared to other techniques, such as the insertion of debugging statements in the program. AID is permanently available and can be easily adapted to the relevant programming language. A program that has been tested by means of AID does not always have to be recompiled but can be used immediately in a production run. The functionality of AID and its test language, the AID commands, are primarily geared to interactive applications. It is quite possible, however, to employ AID in batch mode as well. AID offers comprehensive options for monitoring, runtime control, output and modification of memory contents. Users are also able to access information on program execution and on AID operation.

AID permits debugging both on the symbolic level of the appropriate programming language and on machine code level. During "symbolic debugging" of a program it is possible to reference data, statement labels and program segments with the names declared in the source code, and the statements without names can be referenced with the source reference generated by the compiler.

AID V3.2A can be used as of BS2000/OSD V5.0 or OSD/XC V1.0.

1.1 Target group

AID is intended for all software developers working under BS2000 with one of the programming languages COBOL, FORTRAN, C, C++, PL/I and ASSEMBH or wishing to test and/or correct the programs on machine code level.

1.2 Structure of the AID documentation

The AID documentation consists of a core manual and the language-specific manuals for symbolic debugging plus the manual for debugging on machine code level. For experience AID users there is an additional reference work, a Ready Reference [7], containing the syntax of all commands and also the operands, with brief explanations. The Reference Guide also contains the %SET tables. The manual for the language selected, together with the core manual, should provide all the information needed for testing. The manual for debugging on machine code level may be used in place of or in addition to any of the language-specific manuals.

AID Core Manual

The core manual provides an overview of AID and deals with facts and operands which are the same in all programming languages. The AID overview describes the BS2000 environment, explains basic concepts and presents the AID command set. The other chapters discuss preparations for testing; command input; the subcommand; addressing in AID; the operand *medium-a-quantity*; AID literals; and keywords. The manual also contains messages, BS2000 commands invalid in command sequences and operands supported for the last time in this version.

AID manuals

The manuals contain lists of the commands in alphabetical order. All simple memory references are described in the manuals.
AID - Debugging of COBOL Programs [2]
AID - Debugging of FORTRAN Programs [3]
AID - Debugging of PL/I Programs [4]
AID - Debugging of ASSEMBH Programs [5]
AID - Debugging of C/C++ Programs [6]
In the language-specific manuals, the description of the operands is tailored to the programming language involved. The user is expected to be familiar with the relevant language elements and operation of the corresponding compiler.

The additional possibilities of machine-oriented debugging are described in **Debugging on Machine Code Level [1]**.

The manual for debugging on machine code level can be used for programs for which no LSD records exist or for which the information from symbolic debugging does not suffice for error diagnosis. Testing on machine code level means that the user can employ the AID commands regardless of the programming language in which the program was written.

1.3 Readme file

Any functional changes or additions to the current product version as described in this user guide can be found in the product-specific readme file.
The readme file `SYSRME.produkt.version.E`. is saved on BS2000 system. Please ask your system administrator for the user login.
You can view the readme file using the `/SHOW-FILE` command or open it in an editor. You can also output it at a standard printer using the following command:

```
/PRINT-FILE FILE-NAME=dateiname,
/DOCUMENT-FORMAT=*TEXT(LINE-SPACING=*BY-EBCDIC-CONTR)
```

1.4 Changes made since AID V2.1A

The Readme file for AID V3.1 has been incorporated in the manual:

– Index specification in the event of arrays

– Extensions/changes in the %AID, %CONTROLn, %STOP and %TRACE commands

AID V3.2 supports Unicode. This has led to the following additions:

– Data type %UTF16 for representing strings whose characters have 2-byte UTF16 encoding

– Conversion functions %UTF16() and %C()

– UTFE literal U'..'

1.5 Notational conventions

italics Within the text, operands are shown in *italic lowercase*.

This symbol marks points in the text to which particular attention should be paid.

2 Metasyntax

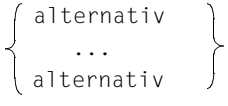
The following metasyntax is used for representing commands and their operands. The symbols used are listed below, together with a description of their meaning.

UPPERCASE

Character sequence which must be used in precisely this form when a function is selected.

lowercase

Character string representing a variable. It must be replaced by one of the permitted operand values.



`{alternative | ... | alternative}`

Alternatives between which a choice must be made. Both formats are equivalent.

`[optional]`

Specifications enclosed between square brackets may be omitted.

In the case of AID command names the part shown between square brackets must be omitted in full if it is omitted; any other abbreviations result in a syntax error.

`[...]`

Repeatability of an optional syntactical unit. If a separator, for example a comma, has to be placed before each repetition, it is shown before the dots representing repetition.

`{...}`

Repetition of a syntactical unit which must be specified once. If a separator, for example a comma, has to be placed before each repetition, it is shown before the dots representing repetition.

Underscore

The underscore identifies the default value that is used by AID if the user does not specify a value for an operand.

- The heavy-type period has a number of roles: it separates qualifications, or it stands for a *prequalification* (see %QUALIFY), or it is the operator for a byte offset, or it is part of the execution counter or subcommand name. The period is entered in the normal way with the period on the keyboard. It is shown here in heavier type merely to improve readability.

3 BS2000 environment, basic concepts and command set

3.1 AID in BS2000

The AID test system consists of two components:

– the user interface AID and
– the system interface AIDSYS.

This splitting makes the AID user interface independent of the BS2000 versions. All necessary system functions are implemented via AIDSYS. This independence from BS2000 versions is important if, for example, one of the available system tables is to be output in edited form from a dump generated on another system with a different BS2000 version. Version-dependent editing is performed via AIDSYS, which recognizes which BS2000 version was used to generate the dump, edits the required system table accordingly and then passes it to AID for output.

Input of AID commands and output of AID messages are effected via the system files SYSCMD and SYSOUT in the same way as for BS2000 commands (see Commands, Volumes 1 - 5 [8]).

3.1.1 Loading AID

AID is not loaded by the non-privileged user but by the system administrator with the /START-SUBSYSTEM AID command (see System Administrator Commands (SDF Format) [10]) under the TSOS ID. AID is then available to all users without any additional intervention.

3.1.2 Using AID

An AID debugging session may be started in one of two ways:

1. Load and start the program. If the program run is interrupted by an error and symbolic testing is desired, load the LSD records with the %SYMLIB command for the compilation unit in which the error has occurred. AID commands can then be entered.

```
/START-EXECUTABLE-PROGRAM FROM-FILE=...
...
%  IDAON51 PROGRAM INTERRUPT AT LOCATION '000B62 (MOBS), (CDUMP), EC=58'
%  IDAON45 DUMP DESIRED? REPLY (Y = USER/AREA DUMP; Y,SYSTEM = SYSTEM DUMP;
N = NO)
...
/%SYMLIB ...
/%SDUMP %NEST
...
```

2. Load the program. If symbolic testing is desired, specify the parameter which loads the LSD records together with the program. Enter AID monitoring commands and then start the program with an AID command.

```
/LOAD-EXECUTABLE-PROGRAM FROM-FILE=..., TEST-OPTIONS=AID
/%INSERT ...
/%R
...
```

3.1.3 AID and the BS2000 command interpreter

The AID functions are called via AID commands. An AID command starts with a

% character immediately followed by the command name:

```
%DISPLAY ARRAY1
```

AID commands can be entered whenever the task is in command mode. The AID commands are accepted by the BS2000 command interpreter like normal BS2000 commands. The command interpreter identifies the AID commands on the basis of the % character and passes them to AID for execution.

AID commands may be entered in interactive mode or in procedure files. The CMD macro permits AID commands to be called from a program (see Executive Macros [11]).
AID commands can coexist with BS2000 commands in command sequences.

3.1.4 AID and SDF

SDF (System Dialog Facility) is the interactive interface to BS2000. SDF has its own command language, which replaces the previous command language in ISP (Interactive String Processor) format. In the AID core manual and the language-specific manuals, BS2000 commands are always described in the EXPERT form of the SDF format (see Introductory Guide to the SDF Dialog Interface [16]). In some cases a comparison with the corresponding ISP commands is contained in the appendix; references to this are included where appropriate.

SDF notation is not available for AID commands.

3.1.5 AID link names

Dump files can be referenced with AID via link names D0 through D7.

Data output, trace listings and REPs may be written to output files. AID output files have the format `FCBTYPE=SAM, RECFORM=V, OPEN=EXTEND`. AID output files are assigned via link names F0 through F7.

3.1.6 Programs on XS computers

On all of BS2000/OSD versions programs can use the extended address space from over 16 Mbytes to 2 Gbytes. As a consequence, AID can also be used to test programs in the extended address space.

AID automatically adjusts to the addressing mode of the test object and works with both 24-bit (lower address space) and 31-bit (extended address space) addresses.

If for instance the program linkage of modules with different addressing modes is to be tested, AID offers the following functions (see Debugging on Machine Code Level [1]):

– keyword for the AMODE system information (%AMODE)

– displaying the current addressing mode (%DISPLAY)

– switching the addressing mode for the test object (%MOVE)

– switching the address interpretation for indirect addressing (%AINT)

3.1.7 Programs on ESA computers

As of BS2000/OSD V1.0, application programs on ESA (Enterprise System Architecture) computers can use not only the program space, which corresponds to the previous address space, but also other address spaces for data, the data spaces. Data spaces can only contain data; program code cannot be executed in a data space. They can be uniquely addressed via the SPID (space identification) or via one or more ALETs (access list entry tokens). To allow addressing with ALETs the access registers were introduced as an additional register record in parallel with the general registers. The ALETs are contained in the access registers. When AR (access register) mode is activated, the access registers are also analyzed during address translation in a machine instruction, and in that way data is addressed in a data space.

Only programs which run on ESA system with a version of BS2000/OSD ≥ 1.0 and which use ESA instructions are able to store data in a data space of this type (see Executive Macros [11).

AID provides the following functions for ESA support (see Debugging on Machine Code Level [1]):

– keyword for the ASC mode system information (%ASC) for interrogating AR mode.
– keywords for the access registers (%nAR, %AR)
– ALET and SPID qualification for the unique referencing of virtual addresses in data spaces
– keywords for the system information about the active data spaces (%DS[(ALET/SPID-qua)])
– identification of virtual addresses from data spaces with an asterisk "*" in the event of output with %DISPLAY and in the %TRACE log.

Data in data spaces can only be referenced via its virtual address. If it is intended to edit the data symbolically, this can be done with the aid of subsequent type modification.

3.1.8 Test privileges

AID users must be prevented from accessing and/or modifying arbitrary data sets and memory areas within the system. Each user entry in the JOIN file therefore contains a "test privilege" entry to control read/write access rights for testing. This entry is made by the system administrator (see the System Administrator Commands (SDF Format) [9]).

When a task is started, the lowest privileges (1,1) are assigned. They allow usage of the complete AID function range as described in the documentation.

If files or libraries have been protected by means of a read or execute password, access under AID is not possible unless the correct password is entered.
If memory areas are to be accessed which require higher privileges, the /MODIFY-TEST-OPTIONS command can be used to change privileges, provided this is permitted in the JOIN file entry. This entry may be viewed via the /SHOW-USER-ATTRIBUTES command (see the Commands, Volumes 1 - 5 [8]).

AID also offers keywords for access to protected areas such as registers.

3.2 Basic concepts

3.2.1 Test object

The program to be processed by means of AID is known as the test object. It may be loaded under the relevant user ID or may be present in the form of a memory dump in a dump file. Within a test session, switchover between these two options is possible, for example to compare data in the loaded program with data in a dump file or to compare dumps from different versions of the same object.

The program can always be tested on machine code level. Testing on the symbolic level is possible if LSD records have been created during compilation. The program to be tested need not be recompiled or relinked. As the program can be loaded without the symbolic information, further compiler or linkage editor runs after an error-free test run can be dispensed with. The program can be immediately employed for productive use.

3.2.2 Object structure list and LSD

AID works with two lists that contain information on the program.

When linking is performed with the linkage editor, the corresponding list is the ESV (External Symbols Vector)(default case). When linking is performed with TSOSLNK, the object structure list iscreated from the ESD (External Symbol Dictionary). Among other things, the object structure list contains information on the CSECTs, DSECTs and COMMONs of a program. If this list is available, it is possible to use the name of a CSECT or a COMMON to access the associated address, content and length.

The LSD (List for Symbolic Debugging) is the directory of the data names, statement names and program segment names defined in the module. It also contains the source references created by the compiler. LSD records are generated by the compiler, provided the appropriate compiler option is specified. If LSD records have been created, the names defined in the source program can be used to access the address, content, length and type of the relevant memory objects. The compiler-generated statement names permit access to the executable part of the program.

The class-5 memory requirements of a program with LSD records may reach a multiple of the actual program size, depending on the amount of symbolic definitions involved. If the object modules are stored in a PLAM library, the program may be loaded and started without LSD records, and the PLAM library that contains the LSD records can be opened with the %SYMLIB command. AID then loads the LSD records from the assigned library whenever they are required.

3.2.3 Symbolic versus machine code)

AID knows two debugging levels.

On the symbolic level, the compiler-generated symbolic addresses from the LSD records are used. Memory locations are referenced via the names assigned in the source program. AID output comprises program, data and statement names as well as source references. The keyword %HLLOC (High-Level LOCation), the operand of the %DISPLAY AID command, displays the symbolic localization information. This information comprises the context name, the name of the compilation unit, the name of the current main program or subprogram, and the name of the source reference to which the address is assigned. The AID commands %JUMP, %SDUMP and %SYMLIB can only be used at the symbolic level, i.e. if LSD records exist for the referenced program segment. For the %CONTROLn and %TRACE commands, the keyword for *criterion* decides whether tracing/monitoring takes place on the symbolic level. If AID is to calculate an address (complex memory reference), switchover from the symbolic to the machine code level is possible at any point. Use of address selection and the pointer operator (%@(name)->) enables the referenced memory object to be used with all its machine-oriented attributes.

On the machine code level, only the CSECT and COMMON information from the object structure list is used. AID output comprises virtual addresses and CSECT and COMMON names. The keyword %LOC (low-level LOCation), the operand of the AID command %DISPLAY, displays the localization information at the machine code level. This information comprises the context name, the name of the load unit, the name of the object module, the name of the CSECT and COMMON, and the CSECT-relative and COMMON-relative address. For the %CONTROLn and %TRACE commands, the keyword for *criterion* decides whether tracing/monitoring takes place on the machine code level. In a complex memory reference it is possible to link symbolic addresses and elements of addressing on the machine code level and therefore to continue to use all attributes of symbolic addressing.

3.2.4 AID work area

The AID work area is the address space in which memory objects can be referenced without a base qualification.
It comprises the non-privileged part of the virtual memory of the relevant task occupied by the program, including the connected subsystems, or the corresponding area in a memory dump.
Whether debugging is performed in a loaded program or in a memory dump can be determined by the %BASE command. If %BASE is not specified, the AID work area is in the loaded program. This is referred to as the AID default work area.
It is also possible to deviate from the currently set work area within a command by specifying a corresponding base qualification {E=VM | Dn} in an address operand.

If the AID work area is in a dump file, the commands for monitoring and runtime control cannot be used. Nor is it possible to modify a dump file with AID commands. Data can be output from a dump file, however, the call hierarchy can be traced back to the time of the program interrupt, machine code translated back to symbolic assembler notation, and character strings can be located in a dump file. In addition, data from a dump file can be used to overwrite the memory contents of a loaded program.

3.2.5 Memory objects and memory references

A set of contiguous bytes extending from a specific address in the memory area of the program is known as a memory object. This includes the data of a program as well as the instruction code. The registers outside the program memory and the program counter are likewise memory objects; they are referenced by AID via keywords.

Constants are not regarded as memory objects. This category includes all the constants defined in the program as well as the statement names, the source references, the results of address/length selection and of the length function, and the AID literals. All of these represent a value that cannot be changed and are lacking an address attribute.

A memory reference addresses a memory object. There are two kinds of memory reference: simple and complex. Simple memory references are virtual addresses, names whose address AID can fetch from the LSD records, and keywords.
In a complex memory reference, AID calculates an address on the basis of user specifications which also include information on the type and length of the memory object identified by this address. The following operations may occur in a complex memory reference: byte offset, indirect addressing, type/length modification, address selection.

If a memory reference is not in the currently valid AID work area or is outside the current main program or subprogram, or if it is not unique in that area, qualifications can be used to define the path to the desired memory reference.

3.2.6 Naming conventions in AID

All names used in AID commands to address programs or program segments, data or statements or to define subcommands can make use of the following character set, regardless of the programming language used:
a-z, A-Z, 0-9, $, #, @, underscore "_" or hyphen "-".

The hyphen is not permitted as the first character and also is only allowed as part of the name if SYMCHARS=STD has been set (%AID command).
If a hyphen is the last character of a name, the name can only be specified with N'...'.

It is generally necessary for all names which contain special characters or which can be ambigoues for AID to be set in N'...'. Labels with special characters and labels that are the starting address for a complex memory reference must be written in L'...'.

To ensure that AID differentiates between uppercase and lowercase notation, it is first necessary to enter the %AID LOW[=ON] command. In the case of BLS names, however, in other words names that are known to the binderloader-starter such as names of CSECTs, COMMONs or entries, and in the case of names of compilation units (or in Fortran: program units), account will only be taken of uppercase and lowercase notation if you enter the %AID LOW=ALL command.

The permissible length for BLS names and names of compilation units is 32 characters; names of data and program segments such as functions, procedures or subprograms may be up to 255 characters long. Names of subcommands may be up to 32 characters long including the prefixed characters "%•".

Overview

Names	Length (max.)	%AID LOW=ON active	%AID LOW=ALL active
BLS names and names of compilation units	32	no	yes
Date and program names	255	yes	yes
Names of labels	255	yes	yes
Subcommand names	32 incl. %•	no	no

3.2.7 Character representation using UTF16 / UTFE

The support of Unicode means that the new data type %UTF16 is provided in AID to represent strings. With this data type each character has 2-byte encoding. The data type for representing strings which was supported by AID to date has 1-byte EBCDIC encoding.

AID supports UTFE character encoding for input and output media. This encoding is the EBCDIC variant of UTF8, which supports multibyte encoding with a variable byte length.

Setting an EBCDIC encoding table via %AID

The %AID command has been extended by the EBCDIC operand. This enables EBCDIC encoding of a C string to be specified which AID uses when conversion is to be carried out between a UTFE/%UTF16 string and a 1-byte C string.

AID supports all 1-byte EBCDIC encodings which the XHCS-SYS subsystem offers. You can use the %SHOW %CCSN command to diaplay the current names of the encoding tables.

The new functions %C(...) and %UTF16(...) allow you, for example, to convert the literal encoding into a different encoding.

3.3 AID commands

AID features a wide variety of functions, which are invoked via AID commands. This section provides an overview of the AID functionality. The complete command descriptions can be found in the language-specific manuals or in the manual for debugging on machine code level.
The AID command set can be divided into four function groups, whose commands and operands are shown in the summary below. Commands which can only be used for debugging on the symbolic level are identified by 'SY' in the second column.

Monitoring

Command name		Operands
%C[ONTROL]n		[criterion][,...] [IN control-area] <subcmd>
%IN[SERT]		test-point [<subcmd>] [control]
%ON		{write-event | event} [<subcmd>]
%REM[OVE]		target

Runtime control and logging

Command name		Operands
%CONT[INUE]		
%JUMP	SY	continuation
%R[ESUME]		
%STOP		
%T[RACE]		[number] [criterion][,...] [IN trace-area]

Output and modification of memory contents

Command name		Operands
%D[IS]A[SSEMBLE]		[number] [FROM start]
%D[ISPLAY]		{data} {,...} [medium-a-quantity][,...]
%F[IND]		[[ALL] search-criterion] [IN find-area] [alignment]
%M[OVE]		sender INTO receiver [REP]
%SD[UMP]	SY	[dump-area][,...] [medium-a-quantity][,...]
%SET		sender INTO receiver

Administration

Command name		Operands
%AID		[CHECK] [REP] [SYMCHARS] [OV] [LOW] [DELIM] [LANG] [EBCDIC]
%AINT		[aid-mode] [,...]
%BASE		[base]
%D[UMP]F[ILE]		[link [= file]]
%H[ELP]		[info-target] [medium-a-quantity][,...]
%OUT		[target-command [medium-a-quantity][,...]]
%OUTFILE		[link [=file]]
%Q[UALIFY]		[prequalification]
%SYMLIB	SY	[qua-a-lib]
%SHOW		[show-target]
%TITLE		[page-header]

3.3.1 Monitoring

The commands %CONTROLn, %INSERT and %ON support dynamic monitoring of program execution. They can be used to define monitoring conditions and subcommands (see chapter “Subcommand” on page 49). If the monitoring condition is satisfied, the related subcommand is processed. Each of the three commands specifies a different type of monitoring condition, which can be canceled with %REMOVE.

%CONTROLn *criterion*

criterion designates the type of source statements or machine instructions to be monitored.

%INSERT *test-point*

test-point designates an address in the executable part of the program.

%ON {*write-event* | *event*}

write-event activates write monitoring. *event* designates an event during program execution, such as an addressing error, a supervisor call (SVC), or dynamic loading of a module.

%REMOVE *target*

This command enables monitoring conditions to be canceled. *target* designates the condition to be canceled.

The user chooses the appropriate command for the desired monitoring job and can selectively control program execution via the associated subcommand. A subcommand specifies a command or sequence of commands and possibly a condition. Moreover, the subcommand may be given a name which serves to address its execution counter or to delete the subcommand. The appropriate AID commands can be used within the subcommand to define whether the program is to be interrupted or continued. It is thus possible to prepare an automatic test run. To do this it is essential, however, that all the necessary information for determining the further course of action upon occurrence of the monitoring condition is already available before input of the monitoring command. If the relevant commands are stored in the subcommand, the input of commands at the terminal during testing (for instance to change current memory or register states) is therefore no longer required.

3.3.2 Runtime control

The commands %CONTINUE, %RESUME, %STOP and %TRACE change the status of a loaded program. %JUMP can be used for FOR1 and COBOL85 programs to specify a continuation address that deviates from the coded program sequence. A loaded program can be in any of three defined program states:

1. The program has stopped.

   %STOP, actuation of the K2 key or termination of a %TRACE have interrupted the running program. The task is in command mode. Commands can be entered.

2. The program is running without tracing.

   %RESUME has started or continued the program. %CONTINUE has the same effect; if a %TRACE is still being processed, however, %CONTINUE resumes the program *with* tracing.

3. The program is running with tracing.

   %TRACE has started or continued the program. Program execution is logged as specified in %TRACE. %CONTINUE has the same effect if a %TRACE is still active.

If no other continuation address has been declared, program execution is continued at the interrupt point. By issuing %JUMP (FOR1 and COBOL85 only) or by altering the program counter (%PC) with %MOVE or %SET a different continuation address can be defined. In either intervention in the program sequence it is the user's responsibility to ensure that memory contents, register states and file statuses/contents are compatible with the specified continuation address.

%CONTINUE and %RESUME start or resume a loaded program. The difference between the two commands lies in the fact that %RESUME deletes any active %TRACE, whereas %CONTINUE does not.

%STOP suspends the program and issues a STOP message which contains information on the current interrupt point.

%TRACE activates the trace function. The program is running and the selected commands are logged. The %TRACE terminates when the specified number of commands have been logged or the program is continued with %RESUME after an interrupt. If the %TRACE is only interrupted because a subcommand containing a %STOP has been executed or one of the *control* operands KEEP or STOP has been executed or the K2 key has been depressed, the %TRACE can be resumed with %CONTINUE.
The %TRACE command has been extended by a *continue* operand with which you can control whether the program should stop (default value) or continue to run without logging, after %TRACE terminates.

3.3.3 Output and modification of memory contents

The commands %DISPLAY, %SDUMP and %DISASSEMBLE can be used to output memory contents and information on the program.
The commands %MOVE and %SET serve to modify memory contents in the loaded program.

%FIND searches for character strings.

%DISPLAY outputs the current content of memory objects, their addresses/lengths or the values of constants, statement names and source references. %DISPLAY can also be used to query system information, control the SYSLST feed or output AID literals, e.g. to annotate the test run. Output is effected via SYSOUT, SYSLST or to a cataloged file.
If a memory object is referenced with its name, AID outputs it in the data type and length specified in the source program. A different editing format can be defined via type/length modification.

%SDUMP outputs a symbolic dump. Output may include either data of the current call hierarchy, or the call hierarchy itself. The current call hierarchy ranges from the subprogram level where the program was interrupted to the subprograms invoked by CALL statements to the main module.
%SDUMP %NEST outputs the names of all program segments of the current call hierarchy as far as the linkage conventions are known to AID. Data or data areas can then be output from the program segments of this hierarchy.

%DISASSEMBLE "retranslates" memory contents from the executable part of a program, i.e. this command causes AID to display these memory contents edited in symbolic Assembler notation. Any memory contents which cannot be interpreted as a command are displayed in the form of an output line which contains the memory contents in hexadecimal notation as well as the note INVALID OPCODE.

%MOVE changes memory contents in the loaded program. %MOVE transfers a sender to a receiver, left-justified and in the length of the sender, without checking whether the storage types of sender and receiver are compatible and without matching the respective types. AID merely checks that the right-hand limit (= end address) of the receiver is not exceeded. Activation of the update dialog and creation of REP records are supported for %MOVE.

%SET changes memory contents in the loaded program. %SET transfers a sender to a receiver and checks, prior to the transfer, whether the storage types of sender and receiver are compatible. The transfer is effected in the length of the receiver and in accordance with the appropriate type; truncation, padding or type matching takes place as required. The rules for transfer with %SET are closely related to the relevant programming language. The %SET description in the language-specific manuals contains a table listing the permissible storage type combinations. Activation of the update dialog is supported for %SET.

%FIND locates a character string in specific data sets or in the entire user address space of the loaded program or in a dump file and displays the hits on the terminal (SYSOUT). For a hit, AID outputs the address at which the string was found and, if possible, the name of the associated CSECT or of the COMMON and the distance to the start address of CSECT or COMMON. To do that, the memory contents are output from the hit address up to the end of the search range, but to a length of no more than 12 bytes. In addition, AID stores the hit address in AID register %0G and the continuation address (hit address + search string length) in AID register %1G.

3.3.4 Administration functions

The commands %DUMPFILE, %SYMLIB and %OUTFILE support the administration of AID input and output files. The files can be assigned link names and opened or closed.
%OUT controls AID output, %TITLE specifies a header line for output to SYSLST.
%AID, %AINT, %BASE and %QUALIFY define global presettings.
%HELP displays help texts.
%SHOW calls up information about the currently valid default settings and about the AID commands which have been entered in the debugging run so far and which are still active.

%DUMPFILE helps administrate dump files, which are assigned via the AID link names D0 through D7. AID may be caused to open or close a dump file. The dump files contain memory dumps to be used for debugging.

%SYMLIB opens PLAM libraries in which the OMs or LLMs of the program have been stored with the LSD records.

AID accesses open PLAM libraries when a command references symbolic names that are contained in a compilation unit (in Fortran: program unit) for which no LSD records have been loaded. A base qualification can be used to assign a PLAM library to a specific AID work area.
Upon %SYMLIB declaration, AID merely checks whether the specified library can be opened; AID does not check whether the library contents match the program being processed. This makes it possible to declare libraries in advance which may be needed in the course of the debugging run. If several libraries are declared for a base qualification, AID searches them in the sequence in which they were specified in the %SYMLIB command.
AID can manage up to 14 PLAM libraries in parallel.

%OUTFILE administrates AID output files, to which the outputs of the commands %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP and %TRACE or the REPs of the %MOVE command are written. These files are assigned via the AID link names F0 through F7. If a file does not yet exist, AID catalogs and opens it. Conversely, open output files can be closed with this command.

%OUT defines the output media for the commands %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP and %TRACE and specifies whether the output of additional information is desired. Possible output media are the terminal (SYSOUT), SYSLST or an AID output file that can be assigned previously via %OUTFILE.
The commands %DISPLAY, %HELP and %SDUMP support a separate, local *medium-a-quantity* operand which temporarily overrides the definitions made for these commands in the %OUT command. The commands %DISASSEMBLE and %TRACE do not offer this operand.

%TITLE permits a specific page header text to be defined for SYSLST output and controls the page counter. Output to SYSLST is specified via %OUT or with the *medium-a-quantity* operand of an output command.

%AID is used to activate the update dialog, to create REPs for memory modifications with %MOVE, to control interpretation of the hyphen in names, to take account of the overlay structure of a program, to activate uppercase/lowercase interpretation for user inputs, to define other delimiters for character-type outputs, to define an EBCDIC character set for conversions from or to UTF16/UTFE or interpret and display characters, and to switch from German to English help texts or vice versa.

%AINT switches the interpretation of indirect address specifications in AID commands. This determines whether an address preceding a pointer operator (->) is to be interpreted by AID as a 24-bit or 31-bit address. This does not affect the addressing mode of the test object. A base qualification can be used to specify the area to which the address interpretation definition is to apply.

%BASE defines the base qualification, which applies to all subsequent commands in which no explicit base qualification is specified. %BASE specifies whether the AID work area is to be in the loaded program or in a dump file. A dump file used as a base qualification must have been assigned via %DUMPFILE.

%QUALIFY identifies qualifications or an address to be referenced in the address operand of another command by prefixing a period. This abbreviation is expedient when making multiple references to addresses which require the same qualifications or which are calculated on the basis of the same start address.

%HELP provides information on AID commands. The following can be output to the selected medium: either all AID commands or the specified command with its operands. The HELP information pool comprises information for both symbolic debugging and debugging on machine code level.

%SHOW provides information on the AID commands of the previous debugging run, about the currently valid global settings, or about the currently valid operand values of the commands.
%SHOW without an operand displays the AID command that was entered last.

With regard to the monitoring commands (%CONTROLn, %INSERT and %ON) %SHOW outputs a list of the original input strings of all active %CONTROLn or %ON commands or the list of set test points with the context, virtual address, CSECT or COMMON and distance from the start of the CSECT or COMMON. The original input string for a certain test point is obtained with %SHOW %INSERT *test-point*. If more than one %INSERTs are active at the test point, the commands are output in reverse order, i.e. the %INSERT that was entered last is listed first.
%SHOW %TRACE causes AID to output the %TRACE command that was entered last, the %TRACE steps that have been processed, and the currently valid operand values.
%SHOW %DISASSEMBLE outputs the currently valid operand values, while %SHOW %FIND outputs the command that was entered last and the last hit.
In relation to the commands that manage AID input or output files, %SHOW outputs all explicitly or implicitly opened files and various additional information. %SHOW %OUT displays the current output declarations of the commands whose output is controlled via %OUT. %SHOW %AID, %SHOW %BASE and %SHOW %QUALIFY each output the declarations that have been made with these commands.

3.3.5 Overview of the scope of validity of the commands

When working with AID it is important to know which commands, operand values or declarations are valid until the next command of the same type is entered or until the end of the program or task. This is the purpose of the following overview:

Command	Operand	Scope of validity
%AID	all	Valid until a new %AID is entered with a corresponding operand or until /EXIT-JOB.
%AINT	aid-mode	Valid until a new %AINT is entered for the same base qualification or until /EXIT-JOB or until the associated dump file is closed.
%BASE	base	Valid until a new %BASE is entered or until /EXIT-JOB or until the dump file declared as the *base* is closed.
%CONTROLn	criterion/ control-area	Can be taken over by the next %CONTROLn, otherwise valid until the %CONTROLn is deleted or until the end of the program.
	subcmd	Must always be specified.
%DISASSEMBLE	number	Can be taken over by a new %DISASSEMBLE; this option is available until the end of the program.
	start	The address following the instruction that was translated back last can be taken over as the *start* value; this option is available until the end of the program.
%DISPLAY	all	All operands must always be specified.
%DUMPFILE	link=file	The file assigned via *link* remains open until /EXIT-JOB unless it is explicitly closed.
%FIND	search-criterion	Can be taken over from a previous %FIND until the search through *find-area* has been completed. This option is available until /EXIT-JOB.
	find-area/ alignment	If no *search-criterion* is specified, *find-area/alignment* is taken over from the previous %FIND until the search through *find-area* has completed.
%INSERT	test-point	The test-point remains entered until it has been deleted with %REMOVE, until all %INSERTs have been deleted, or until the end of the program. For programs that are linked as overlays the test point remains entered in the module in which it was set, even if a different module has been loaded at the same point in the meantime.
	subcmd	The subcommand remains entered until it is deleted with %REMOVE, until the associated test is deleted or until the end of the program.
%MOVE	all	All operands must always be specified.

continued...

continued...

Command	Operand	Scope of validity
%ON	event/ write-event	The *event* remains entered until it has been deleted with %REMOVE, until all %ONs are deleted or until the end of the program. One exception is %ON %WRITE: a new *write-event* overwrites one that is already entered.
	subcmd	The subcommand remains entered until it has been deleted with %REMOVE, until the associated *event* is deleted, until all %ONs are deleted or until the end of the program.
%OUT	all	Valid until a new %OUT is entered with a corresponding operand or until /EXIT-JOB.
%OUTFILE	link=file	If *file* is not explicitly closed, it remains open until /EXIT-JOB.
%QUALIFY	prequalification	*prequalification* applies until it is overwritten by a new %QUALIFY, until it is canceled by a %QUALIFY without an operand, or until /EXIT-JOB.
%REMOVE	target	*target* must always be specified.
%SDUMP	all	Nothing can be taken over from a previous %SDUMP.
%SET	all	All operands must always be specified.
%SHOW	info-target	Nothing can be taken over from a previous %SHOW.
%SYMLIB	qual-a-lib	A library remains open until the next %SYMLIB for the same base qualification, until the next %SYMLIB without an operand, until /EXIT-JOB, or until the associated dump file is closed.
%TITLE	page-header	Valid until the next %TITLE or until the end of the program.
%TRACE	all	All operands continue to apply until they are overwritten by corresponding specifications in a new %TRACE or until the end of the program. *trace-area* will not be taken over if a %TRACE is entered without a *trace-area* and the interrupt point is not in *trace-area*.

4 Prerequisites for debugging with AID

Testing with AID is subdivided into symbolic debugging (where the names assigned in the source program are used for addressing) and debugging on machine code level (where virtual addresses are used). For symbolic debugging to be performed, the compiler must be caused to generate LSD records during compilation. Inclusion of the LSD records in the linkage and loading process can be controlled via corresponding operands. If no LSD records have been included, they can still be dynamically loaded by AID from a PLAM library.
Debugging on machine code level does not require any preparatory action.

If CSECTs are renamed with the LMS statement MODIFY-ELEMENT (substatement RENAME-SYMBOLS) or with the TSOSLNK statement RENAME, symbolic debugging is no longer possible. If an LLM has been generated directly by the compiler and the LLM contains the LSD records, CSECTs can be renamed with BINDER (MODIFY-MODULE-ATTRIBUTES statement). The compiler versions for which this possibility is available are shown in the respective user guide for each compiler.

Debugging on machine code level is not affected by the renaming of CSECTs. The CSECTs in the program can be referenced with the new names.

4.1 Debugging on machine code level

Debugging on machine code level requires no extra operands during compilation, linkage or loading. All the functions described in the manual Debugging on Machine Code Level [1] can be used without preliminary measures.

Within the process of linking by default an object structure list is generated from the external references (see Dynamic Binder Loader / Starter in BS2000/OSD [15]). When linking with BINDER it is the ESV (External Symbols Vector) and when linking with TSOSLNK it is the ESD (External Symbolic Dictionary).

However, no object structure list will be created if the PROGRAM operand SYMBOL-DICTIONARY=NO is written in the SAVE-LLM statement during linkage with BINDER or if the operand SYMTEST=NO is specified during program linkage. The following functions cannot be performed in that case:

– Output a list of all CSECTs and COMMONs of the application program (%D %SORTEDMAP or %D %MAP)
– Output localization information for a memory reference (%D %LOC(memref))
– Specify a CSECT/COMMON qualification in a memory reference
– Trace by means of the %CONTROLn and %TRACE commands, if these are to be explicitly or implicitly restricted to one CSECT
– Create REPs for corrections

In addition, in this case AID cannot output any CSECT-relative or COMMON-relative addresses with %TRACE, %DISASSEMBLE, %FIND or in the STOP message.

Caution is required in the case of LLMs or contexts which contain CSECTs of the same name: in this case it cannot be foreseen which CSECT will be referenced with AID.

4.2 Symbolic debugging

Symbolic debugging with AID enables data to be addressed with user-defined names from the source program and permits statements to be referenced through the input of statement names or source references. For this purpose AID needs information on the names used in the source programs of which the program to be tested is made up. This information is in two parts:

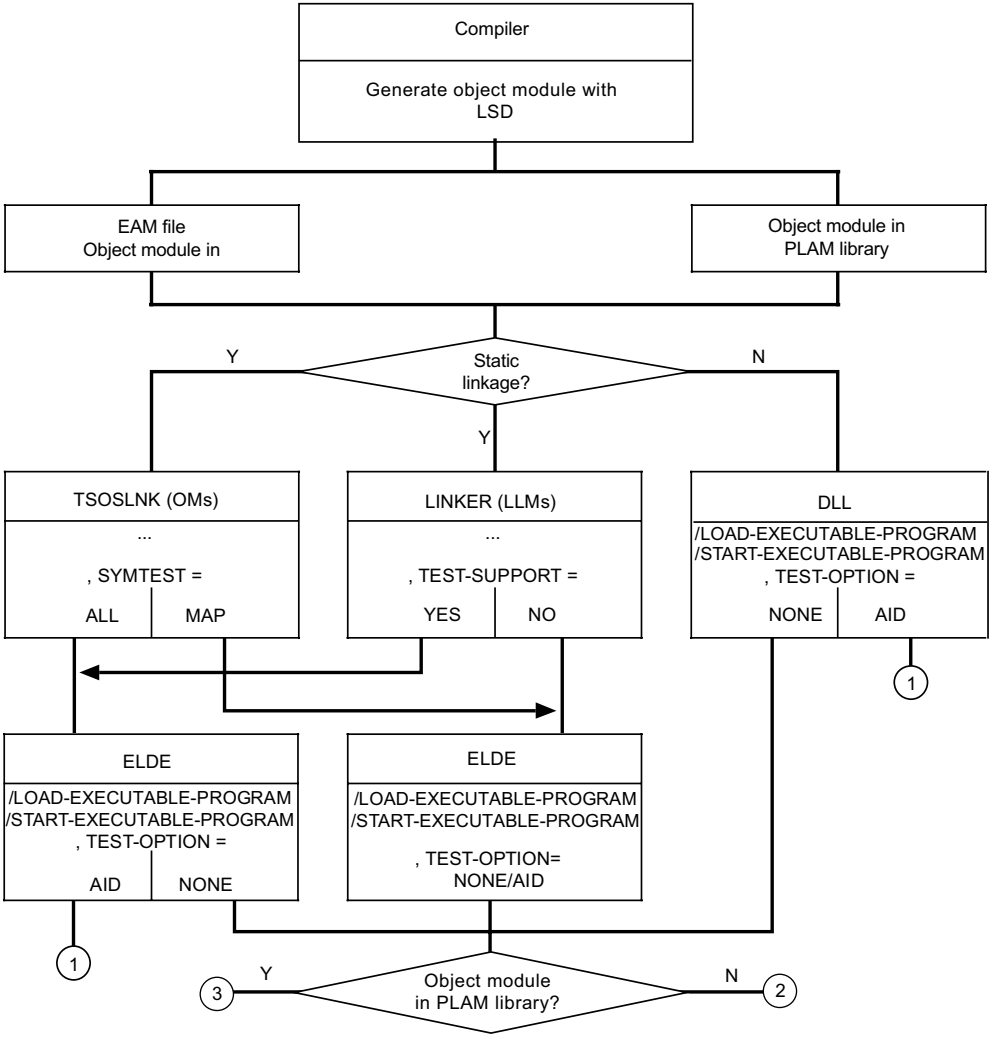
1. LSD (List for Symbolic Debugging): directory of the names and source references defined in the module.
2. ESD (External Symbol Dictionary): directory of the external references of a module during linkage with TSOSLNK or
   ESV (External Symbols Vector) when linking with BINDER.

The following sections deal with the various processing options for ESD/ESV and LSD records for each of the following steps in the software engineering cycle:

– source program compilation
– linkage and loading with DBL (DLL up to BS2000 V9.5) or
– linkage with TSOSLNK/BINDER and loading with ELDE

In addition AID offers the %SYMLIB command, which can be used to open PLAM libraries (see the "LMS (BS2000) [12]) from which AID dynamically loads the missing LSD records as required.

The following diagram outlines the options regarding the inclusion or non-inclusion of the compiler-generated LSD records during linkage and loading.



(1) The program may be tested symbolically without restrictions.

(2) The program may be tested symbolically with restrictions, i.e. names of program segments can be referenced and call hierarchies traced.

(3) The program may not be tested symbolically until the PLAM library containing the OMs or LLMs has been assigned using the %SYMLIB command.

There are thus different ways of supplying AID with LSD information. The prerequisite is always that the LSD records are passed to the generated object module (OM) or link and load module (LLM) during compilation. If the OM or LLM is stored in a PLAM library, the LSD records can either be included in linkage and loading or dynamically loaded by AID when required.

Dynamic loading of LSD records is especially useful for programs which are the result of multiple compilation runs and for which only certain modules are to be symbolically tested. The LSD that is to be dynamically loaded must have been created in the same compilation run as the module.

AID cannot perform dynamic loading from the temporary object module file (*OMF file).

4.2.1 Compilation

A compiler option is used to control generation of LSD records by the compiler. A detailed description of the relevant operands can be found in the language-specific manuals for AID (see [2] - [6]). Essentially there are two possibilities:

– Only the ESD/ESV is created, but no LSD records (default value). The program can only be tested on the machine code level.

– The compiler generates both LSD records and the ESD/ESV. The program can be tested symbolically under AID.

4.2.2 Linkage using BINDER

When linking using BINDER, LSD records can be incorporated in all linkage processes. In the BINDER statements which control the creation, modification or storage of an LLM, the TEST-SUPPORT operand determines whether the LSD records from link and load modules (LLMs) will be incorporated or not (see Binder in BS2000/OSD [14]). Statements which require the same operand values are summarized in the table below, shown with the syntax of the TEST-SUPPORT operand. A description of the operand values follows below the table of statements and the associated TEST-SUPPORT syntax options.

Statement	Meaning	TEST-SUPPORT operand values
START-LLM-CREATION	Creation of an LLM	TEST-SUPPORT={*YES / *NO}
START-LLM-UPDATE MODIFY-LLM-ATTRIBUTES	Update an LLM Modify the physical structure of an LLM	TEST-SUPPORT={*UNCHANGED / *YES / *NO}
MODIFY-MODULE-ATTRIBUTES	Modify the logical structure of an LLM	TEST-SUPPORT={*UNCHANGED / *INCLUSION-DEFAULT / *YES / *NO}
SAVE-LLM	Save an LLM	TEST-SUPPORT={*LAST-SAVE / *YES / *NO}
INCLUDE-MODULES REPLACE-MODULES RESOLVE-BY-AUTOLINK	Insertion of modules Replacement of modules Resolution of external references by Autolink	TEST-SUPPORT={*INCLUSION-DEFAULT / *YES / *NO}

*YES The LSD records are taken over. The linkage editor does check, however, whether the object module (OM) or link and load module (LLM) being processed actually contains LSD records.

*NO The LSD records are not taken over. If SYMBOL-DICTIONARY=YES has been set in the SAVE-LLM statement, however, i.e. the ESV has been included, it is possible to track back through call hierarchies. If LSD records have been created additionally during compilation and written to a PLAM library with the OM or LLM, the LSD can be dynamically loaded for symbolic debugging when required.

*UNCHANGED The LSD records, if there are any in the processed modules, are transferred to the current LLM.

*INCLUSION-DEFAULT

The values of the INCLUSION-DEFAULT operand from the START-LLM-CREATION, START-LLM-UPDATE, or MODIFY-LLM-ATTRIBUTES statements from the same edit run are transferred.

*LAST-SAVE The linkage editor takes over the values from the last SAVE-LLM statement in the same edit run. If no SAVE-LLM has previously been specified, the linkage editor inserts YES.

The BINDER allows CSECTs of the same name to be incorporated more than once in an LLM. During debugging with AID, however, this leads to unforeseeable results.

4.2.3 Linkage and loading via DBL or loading via ELDE

A program to be tested is called by means of the BS2000 command LOAD-EXECUTABLE-PROGRAM, whereupon AID commands can be entered.
A program to be processed by AID only in the event of an error can be loaded and started by means of START-EXECUTABLE-PROGRAM. A load unit included with the BIND macro call can also be tested with AID.

A program in the form of object modules (OMs) or link and load modules (LLMs) is loaded by the Dynamic Linking Loader DBL, whereas a program that is linked via TSOSLINK (load unit) is loaded by the loader ELDE (see Dynamic Binder Loader / Starter in BS2000/OSD [15]).

– Loading or loading and starting with the DBL called by SDF commands:

```
/LOAD-EXECUTABLE-PROGRAM  }
                          } ,TEST-OPTIONS = { NONE }
/START-EXECUTABLE-PROGRAM }               { AID  }
```

NONE The program is loaded without LSD records. Symbolic testing is possible only if the PLAM library containing the relevant object modules is made available to AID for dynamic loading of the LSD records.

AID The program is loaded with the LSD records. If no LSD records exist, the program is loaded nonetheless. If DBL-PARAMETERS:LOADING was specified at the same time, it must be ensured that the associated LOAD-INFORMATION operand, which controls loading of the ESV, is set to DEFINITIONS (default value) or to REFERENCES.

– Integration of an additional load unit with the DBL via the BIND macro call:

```
BIND      ,TSTOPT = { N[ONE] }
                    { A[ID] }
```

NONE Same meaning as above

AID The program is loaded with the LSD records. It will also be loaded even if it does not contain any LSD records. At the same time the LDINFO operand must be set to DEF or REF to ensure that the ESV is loaded.

– Loading or loading and starting via ELDE called by SDF commands:

```
/LOAD-EXECUTABLE-PROGRAM  }
                          } ,...,TEST-OPTIONS = { NONE }
/START-EXECUTABLE-PROGRAM }                     { AID  }
```

NONE The program is loaded without LSD records. Symbolic debugging is possible only if the PLAM library containing the relevant OMs is made available to AID for dynamic loading of the LSD records.

AID The program is loaded with the LSD records. It will also be loaded even if does not contain any LSD records.

Examples

1. `/LOAD-EXECUTABLE-PROGRAM FROM-FILE=*OMF,TEST-OPTIONS=AID`

   DLL loads an object module with LSD records from the *OMF file.

2. `/LOAD-EXECUTABLE-PROGRAM FROM-FILE=IDEAL,TEST-OPTIONS=AID`

   ELDE loads the linked program IDEAL with the LSD records.

3. `/LOAD-EXECUTABLE-PROGRAM FROM-FILE=*LIBRARY-ELEMENT(LIBRARY=PROGRAMLIB,ELEMENT-OR-SYMBOL=ROOTMOD)`

   The linked program ROOTMOD is loaded without LSD records from the PLAM library PROGRAMLIB.

The examples apply analogously for the /START-EXECUTABLE-PROGRAM command.

4.2.4 Dynamic loading of LSD records by AID

AID can dynamically load LSD records for a program from a PLAM library if the library contains the associated OMs or LLMs with the LSD records. The %SYMLIB command instructs AID to open the specified library. If AID processes a command with symbolic operands and recognizes that the related LSD records are not available in memory, AID accesses any libraries assigned and opened via %SYMLIB. AID checks whether the dynamically loaded LSD records are derived from the same compilation run as the module for which they are loaded.

If no library has been assigned or the assigned libraries do not contain the desired OM or LLM, or if they do not contain any LSD records, AID reports the LSD records to be missing. The required library can be assigned with a new %SYMLIB command. If the AID command for which the LSD records were missing is then repeated, AID will process it.

In the case of LLMs it is possible to mask CSECTs contained in them with the BINDER

statement MODIFY-SYMBOL-VISIBILITY. AID cannot dynamically load any LSD information for such CSECTs. The same applies to CSECTs for which the

RUN-TIME-VISIBILITY operand was set to YES, because this includes masking of the CSECT. This operand can be specified with the following BINDER statements:

– INCLUDE-MODULES
– MODIFY-MODULE-ATTRIBUTES
– REPLACE-MODULES
– RESOLVE-BY-AUTOLINK

Programs which contain masked CSECTs can only be put through symbolic debugging with AID if the LSD is loaded together with the program. Dynamic loading is only possible if the masking has been reset in a separate BINDER run.

It should be pointed out that AID terminates the LSD search in an LLM when it finds the first CSECT of the required name, even if the LSD for that CSECT is inconsistent, for example because the LSD has not come from the same compilation as the CSECT. It is therefore of no benefit if there is another CSECT of the same name with a consistent LSD contained in the same LLM.

Examples

1. ```
   /LOAD-EXECUTABLE-PROGRAM FROM-FILE=*LIBRARY-
   ELEMENT(LIBRARY=PROGRAMLIB,ELEMENT-OR-SYMBOL=ROOTMOD)
   ```

   The linked FORTRAN program ROOTMOD is loaded without LSD records from the PLAM library PROGRAMLIB.

   After input of the following AID command with which the statement with label 10 is referenced:

   ```
   %INSERT L'10'
   ```

   the following error message is issued:

   ```
   AID0378       Symbolic information missing
   ```

   If the PROGRAMLIB library contains the object module for this program with the associated LSD information, the command

   ```
   %SYMLIB      PROGRAMLIB
   ```

   can be used to assign the appropriate PLAM library. The %INSERT command can then be repeated and will be processed by AID. The object module and the load unit may be contained in different libraries, but the load unit must have been linked from the object module version from which the LSD records are dynamically loaded. If this is not the case, AID displays the following error message:

   ```
   AID0377       Symbolic information inconsistent for (&00)
                 &00 = programname
   ```

2. ```
   %SYMLIB E=D1.OBJMOD.LIB1,E=D1.OBJMOD.LIB2
   ```

   The PLAM libraries OBJMOD.LIB1 and OBJMOD.LIB2 are available for dynamic loading of the LSD records for the dump file with link name D1.

3. ```
   %QUALIFYE=D2
   %SYMLIB E=D3.LIB1,.LIB2,LIB3
   ```

   The PLAM library LIB1 is defined and opened for the dump file with link name D3.
   Library LIB2 is defined and opened for the dump file with link name D2.
   The PLAM library LIB3 is defined and opened for the current AID work area.

   If no %BASE command has been issued, the current AID work area is the virtual memory area of the loaded program. Following a %BASE command, the AID work area is the one specified in %BASE.
   ```

5 Command input

5.1 Command format

Every AID command starts with the percent character (%), immediately followed by the command name.
Operands may follow after at least one blank.
If operands and/or keywords are entered in succession without a predefined delimiter, they must be separated by at least one blank.
Operands must be entered in the sequence in which they appear in the format descriptions.

Command names

An AID command can be assigned a name like a BS2000 command. Such a name may comprise up to 255 characters:

– 1st character: A-Z, $, # or @

– all subsequent characters: A-Z, 0-9, $, #, @ or -,
where the hyphen (-) must not be the last character of the name.

Names which are branched to with SKIP-COMMANDS begin with a period; names from S-procedures to which the process branches with GOTO are concluded with a colon.
The name follows the slash which is output by the system in interactive mode and entered by the user in procedure files. This name and the % character of the AID command must be separated by at least one blank.
Example: /.START %AID CHECK=NO

The BS2000 command names serve as branch destinations in procedures; they are not relevant for testing with AID.

Continuation of input lines

If an AID command overflows into the next line, the same continuation mechanism applies as for BS2000 commands. In interactive mode, an input may extend over several lines. Alternatively each line may be concluded with a hyphen and sent off separately. The continuation line then starts after the slash displayed by the system.

In procedure files, a continuation line must be announced by a hyphen, which may be followed only by blanks up to the end of the line.
The continuation line must begin with a slash.

The length of an AID command must not exceed 1000 characters.

As there is only a limited area in memory which can be used for the interpretation of a command, the number of operands in a command is restricted. The individual command descriptions contain information on how many operands can be specified in each case.

Use of blanks and comments

Blanks and comments may be used to make AID commands clearer and easier to read. Just like in the BS2000 command language, comments must be enclosed in double quotes ("). Blanks and comments can be inserted whenever one of the following characters occurs:

```
–         Blank
.         Period
,         Comma
=         Equal sign
'...'     Apostrophe
(...)     Opening and closing parentheses
<...>     Opening and closing angle brackets
[...]     Opening and closing square brackets
;         Semicolon
+ – * /   Arithmetic operators
->        Pointer operator
```

When using the minus sign or hyphen "-" the presetting of the *SYMCHARS* operand in the %AID command must be taken into account.

Example

```
%CONTROL1    %CALL   "SORT CALL"   <%DISPLAY 'CALL'; %STOP>
```

5.2 Individual commands

AID commands may be entered in BS2000 command mode or called via the CMD macro interface. AID commands are accepted by BS2000 like BS2000 commands and passed to AID after they have been identified as AID commands on the basis of the % character. If the BS2000 command interpreter determines during input that an AID command is too long, it rejects it with an error message and the user can reenter the corrected command.

AID checks the command syntax and semantics and determines whether the operand values can be processed in the current test situation. An error message is issued, for example, if a symbolic address is referenced which is not stored in the available LSD records (see section “Basic concepts” on page 18).

If a syntactically invalid command is entered, AID issues an appropriate error message and marks the location where it detected the error. The corrected command can then be entered once more.

Once AID has accepted and executed a command, the type of command involved determines whether the program is started or further commands can be entered.

5.3 Command sequences and subcommands

Command sequences can be formed to combine a number of AID and/or BS2000 commands. Successive commands must be separated by semicolons.
A command sequence must not be longer than 1000 characters (same limit as for a single AID command).
Command sequences are executed immediately. They are processed from left to right.

All commands in a command sequence which start with % are identified by AID as AID commands and immediately checked for errors. If AID senses a syntax error, the entire command sequence is rejected during input. AID interprets commands without a leading % character as BS2000 commands and accepts them without any further check. Errored or illegal BS2000 commands are thus not recognized until command execution and lead to abortion of the command sequence. Processing of the command sequence is also aborted in the case of serious errors in AID commands, such as address overflow. The system is then in command mode, i.e. the user may enter further commands.
If an AID command cannot be executed because a specified name is not stored in the LSD records or no LSD records are loaded, AID issues an appropriate error message for this command and continues processing of any subsequent commands.
Since the entire command sequence must be reentered after certain errors, lengthy command sequences should be used in completely tested procedure files only.

Command sequences may include all those BS2000 commands which are permitted in the CMD macro (see Executive Macros [11]) and nearly all AID commands.

The following commands are illegal in command sequences:

AID commands: %AID, %ALIAS, %BASE, %DUMPFILE, %HELP, %OUT, %QUALIFY, %?

BS2000 commands: See list in appendix.

SDF-P control flow commands are likewise illegal in command sequences.

Moreover, some BS2000 commands that are permitted in command sequences terminate a loaded program, i.e. the program can then no longer be processed with AID commands. This applies to the following BS2000 commands (see the Commands, Volumes 1 - 5 [8]):

Command	Function
CALL-PROCEDURE	Call a procedure
EXIT-JOB	Terminate a job
HELP-SDF	Information on how to call SDF commands
LOAD-PROGRAM	Load a program
LOAD-EXECUTABLE-PROGRAM	Load a program
LOGOFF	Terminate a job
START-PROGRAM	Load and start a program
START-EXECUTABLE-PROGRAM	Load and start a program

A list of the corresponding commands in ISP format is contained in the appendix.

The loaded program is likewise terminated when any of the user-own commands defined with SDF-A and implemented by command procedures is called (see Executive Macros [11]).

The commands %TRACE, %RESUME, %CONTINUE and %STOP terminate a command sequence. After %STOP the system is in command mode, whereas the commands %TRACE, %RESUME and %CONTINUE start or continue the program. This is why all of these commands should only occur as the last item in a command sequence.

A subcommand is not a command in its own right but an operand of the monitoring commands %CONTROLn, %INSERT and %ON.
The subcommand is not processed until the monitoring condition has been satisfied.
The command section of subcommands is subject to the same rules as command sequences, with the following exceptions:

– The length of monitoring command plus subcommand must not exceed 1000 characters.

– Like %CONTINUE, %RESUME, %TRACE and %STOP, a %REMOVE for the subcommand just executed makes sense as the final command only, since any ensuing commands of the subcommand will not be executed.

– In the subcommand of a %CONTROLn it is not permitted to specify another %CONTROLn command or an %INSERT, %JUMP (COBOL85, FOR1) or %ON.

Examples

1. `%INSERT S'20' <%DISPLAY A,B;%SET A INTO B;SET-FILE-LINK...;%REM %•>`

   When the running program arrives at statement 20, AID outputs the contents of variables A and B, assigns the value of A to variable B, and calls the SDF command SET-FILE-LINK. Since the subcommand also contains a %REMOVE %• it is deleted following execution.

2. `%ON %LPOV <%DISPLAY %LINK>`

   Whenever a module is dynamically loaded during a program run, AID outputs its name. The program run is continued.

5.4 Command files

AID commands may also be contained in BS2000 procedure files. If an input record is to begin with an AID command, the first character must be a slash followed by the % character of the AID command. Any label for /SKIP-COMMANDS or /GOTO must precede the % character however.

If a BS2000 procedure contains an AID command requiring an acknowledgment (see /%AID CHECK=ALL.../%SET...), AID inserts Y as an answer in batch mode.

Example

```
/LOAD-EXECUTABLE-PROGRAM FROM-FILE=*LIB-ELEM(LIB=TESTLIB,LIB-OR-SYM=TESTLLM),
                                                                TEST-OPT=AID
/BEG: %SET 17 INTO SLF
/%INSERT S'71'  <%DISPLAY I,J,K>
.
.
.
/GOTO BEG
/END: EXIT-PROC
```

6 Subcommand

6.1 Description

A subcommand is an operand of one of the monitoring commands %CONTROLn, %INSERT and %ON. These commands define a monitoring condition which must be satisfied for the related subcommand to be processed. This offers the option of effectively controlling the debugging run and of setting up automated test sequences where, for instance, current data statuses are written to logging files or contents of data fields or registers are modified at predefined points in the program.

```
subcmd-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                                    ⎧AID-command    ⎫
<[subcmdname:] [(condition):] [     ⎨               ⎬ {;...}]>
                                    ⎩BS2000-command ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

The subcommand name can be used in the course of the debugging run to reference the subcommand, for example to interrogate the subcommand execution counter or to delete the subcommand. Execution of the subcommand may depend on a condition, which must be situated between the subcommand name and the command section. The command section may consist of a single command or a command sequence and may contain both AID and BS2000 commands (see also section “Command sequences and subcommands” on page 45).

In %INSERT, %CONTROLn and %ON, AID inserts <%STOP> as a subcommand if no subcommand is specified by the user. If, however, a *subcmdname* or *condition* is specified and just the command section is omitted, AID does not add a %STOP but behaves (at the test point or on occurrence of the defined event) as if a %CONTINUE were inserted:

– the execution counter is incremented (can be queried via %•*subcmdname*)

– the program continues

– any %TRACE is resumed.

Address operands in the command section of a subcommand which do not contain a complete explicit qualification are complemented during input in accordance with the currently valid definitions for the base qualification (see %BASE) and for *prequalification*

(see %QUALIFY). Only the syntax of a subcommand is checked during input. Whether the specified symbolic addresses are contained in the LSD records or whether LSD records for a program segment that is referenced via a qualification are loaded at all is not checked by AID until subcommand execution. This means the corresponding LSD records need not yet be loaded when the subcommand is entered. Likewise, when qualifications are used there is no check at subcommand input as to whether the program segments identified in that way exist or have been loaded.

The subcommands of %INSERT or %ON can be chained according to the LIFO principle, i.e. subcommands may be modified/updated later as a function of the test results. Detailed information can be found in section “Chaining” on page 60.
In the subcommands for %INSERT and %ON, further %INSERT and %ON commands may be defined. This is described in section “Nesting” on page 62.
Some commands are not permitted in subcommands and/or abort the subcommand, the program, or even the task. chapter “Command input” on page 43 contains complete information on this topic as well as a description of error handling in subcommands.

Examples

1. `%CONTROL1 %STMT IN (S'20':S'27') <%DISPLAY A_ARR>`

   The contents of all elements of field A_ARR are output before statements 20 through 27 are processed.

2. `%INSERT INPUT <%DISPLAY INDAT;%SET KEY INTO I-KEY>`

   Every time the running program reaches the paragraph with the name INPUT the input record INDAT is output and the contents of key field KEY are transferred to I-KEY.

3. `%ON %LPOV <%SDUMP %NEST>`

   The current call hierarchy is displayed every time a new segment has been loaded.

4. `%INSERT V'2C' <%SET #2C'INTO%5;%DISPLAY'INS_2C!!!'>'`

   Prior to execution of the instruction with address V’2C’ AID loads register 5 with the value #2C’anddisplaysthetext"INS_2C!!!".'

5. `%ON %SVC <%C1 %INSTR <%R>; %C2 %INSTR <%REM %C>; %T 2 %INSTR>; %R`

   All SVCs from input of the above %ON command onwards are logged.

   Firstly, %ON %SVC specifies that the subsequent subcommand is executed before execution of an SVC. The subcommand contains two %CONTROL commands and one %TRACE, each with the %INSTR criterion. The %TRACE executes the next instruction, which is the SVC, and logs it. Execution of the next instruction triggers processing of the two subcommands for %CONTROL1 and %CONTROL2: program execution is continued with %RESUME, until the next SVC is detected; the second subcommand (%REMOVE %CONTROL) immediately resets the %CONTROL because otherwise a %RESUME would be executed for each subsequent instruction, which would greatly reduce the speed of program execution.

6.2 Name and execution counter

A subcommand name comprises up to 30 characters; the first character may be A-Z, $, @ or underscore "_", the subsequent characters may also include the digits 0-9 and the hyphen (-). A hyphen at the end of a command line is always interpreted as a continuation character in interactive mode. The subcommand name is concluded with a colon, which is not part of the name however.
The name must follow the opening angle bracket and must be unique: identical subcommand names are rejected by AID with an error message. In the case of nested subcommands AID checks the name not during input but only at subcommand execution. In particular, internal subcommands from nesting which are executed more than once can be given a name only if they are explicitly deleted after every time of their execution.
Up to 256 different subcommand names may be assigned.
If the subcommand contains a %STOP, the subcommand name is included in the STOP message.
The subcommand name must be preceded by the string %• if it is to be used as a general AID operand. This results in the AID keyword %•*subcmdname* which can be used in the course of the test run to reference the subcommand and its execution counter.
All subcommands, even those which do not have a name, can be referenced within the subcommand by means of the %• string. Outside the subcommand it is not possible to reference the execution counter and subcommand with the %• string.

The execution counter is directly connected with the subcommand name, because it can be referenced via that name. AID also has an execution counter for subcommands which do not have a name, however; this counter can only be interrogated within the associated subcommand with the %• string.
The execution counter is a numerical value which is incremented by one each time the subcommand is processed. The counter is also incremented if the subcommand contains

a condition whose result is FALSE (preventing execution of the associated command section). The user may modify the execution counter via %MOVE or %SET. It is possible for the execution counter to assume a negative value.
The current status of the execution counter can be queried via %DISPLAY %•*subcmdname* (or %DISPLAY %.z for the subcommand about to be executed).

Examples

1. `%CONTROL1 %IO <IO: %CONTINUE>`

   The %•IO execution counter is incremented by 1 on each input/output operation of the program.

2. `%IN L'200' <L200: %DISPLAY %•IO; %STOP>`

   The program stops at label 200 and AID outputs the status of the execution counter belonging to the subcommand with the name %•IO from example 1.

3. `%CONTROL2 %CALL <PAR: %D %•,PAR1,PAR2,PAR3 P=MAX>`

   This command monitors the subprogram calls. The execution counter reflects the number of CALL statements that have already been issued. Parameters PAR1, PAR2 and PAR3 are additionally logged on SYSLST.
   As the subcommand has the name PAR, %DISPLAY %•PAR can be entered at any point during further testing to determine how many CALL statements have been executed by the program.

6.3 Conditional execution

AID offers the possibility of making the execution of a subcommand dependent on a condition. The condition must be enclosed in parentheses and concluded by a colon; it is situated immediately before the command section of the subcommand.

AID checks the condition and assigns the value TRUE or FALSE to it. Only in the case of TRUE is the command section executed; in the event of FALSE it is skipped.

```
condition OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                                 ⎧AND⎫
([NOT] comparison1 [⎨OR ⎬ [NOT] comparison2] [...]):
                                 ⎩XOR⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Operands can be formed and used for *comparison*$_n$ in accordance with the following syntax:

```
comparison OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

⎧      ⎧dataname          ⎫⎫             ⎧      ⎧dataname          ⎫⎫
⎪      ⎪statementname     ⎪⎪             ⎪      ⎪statementname     ⎪⎪
⎪      ⎪S'...'            ⎪⎪             ⎪      ⎪S'...'            ⎪⎪
⎪      ⎪compl-memref      ⎪⎪             ⎪      ⎪compl-memref      ⎪⎪
⎪[qua]⎨                   ⎬⎪ ⎧EQ | NE⎫  ⎪[qua]⎨                   ⎬⎪
⎪      ⎪V'f...f'          ⎪⎪ ⎪GE | LE⎪  ⎪      ⎪V'f...f'          ⎪⎪
⎨      ⎪C=csect           ⎪⎨ ⎨       ⎬  ⎨      ⎪C=csect           ⎪⎬
⎪      ⎪COM=common        ⎪⎪ ⎪GT | LT⎪  ⎪      ⎪COM=common        ⎪⎪
⎪      ⎪keyword           ⎪⎪ ⎩NG | NL⎭  ⎪      ⎪keyword           ⎪⎪
⎪%@(...)                  ⎪⎪            ⎪%@(...)                  ⎪⎪
⎪%L(...)                  ⎪⎪            ⎪%L(...)                  ⎪⎪
⎪%L=(expression)           ⎪⎪            ⎪%L=(expression)           ⎪⎪
⎩AID-literal               ⎭⎭            ⎩AID-literal               ⎭⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Summary of relational and Boolean operators:

Relational operators		Boolean operators	
EQ	equal	NOT	logical negation
NE	not equal	AND	logical AND
LE	less or equal	OR	logical OR (inclusive)
LT	less than	XOR	logical OR (exclusive)
NL	not less than		
GE	greater or equal		
GT	greater than		
NG	not greater		

The relational operators are all of the same precedence and are processed before the Boolean operators.

The Boolean operators are subject to the following order of precedence:

NOT highest precedence

AND second-highest precedence

OR/XOR lowest precedence

Operators of the same precedence are processed from left to right. Appropriate parentheses must be used if the operators are to be processed in an order other than their predefined precedence.

Examples

```
(I EQ J AND VAR EQ 'A' OR VAR EQ 'B'):
corresponds to:
(((I EQ J) AND (VAR EQ 'A')) OR (VAR EQ 'B')):

(NOT VAR EQ 'A' OR ADR NE %OG):
corresponds to:
((NOT (VAR EQ 'A')) OR (ADR NE %OG)):
```

Parentheses must also be used if the relational operators or Boolean operators can be confused with variable names and consequently might be rejected as faulty syntax.

The only unary operator is NOT, i.e. it refers to one operand only. All the other Boolean operators and all relational operators are binary, i.e. they link two operands with each other.

The relational operators support the comparison of precisely two operands in each case; chaining is not possible. If, for instance, the condition `(A EQ B EQ C):` is specified, AID rejects it during input with the message `AID0271 Syntax error`. Instead the condition should be
represented by `(A EQ B AND B EQ C):`.

Any memory reference permitted for AID (see section “Memory references” on page 71) can be used as an operand for relational operators. If data items from the user program are employed, they are assigned one of the following storage types:

– binary string (≙ %X)

– character (≙ %C)

– numeric (≙ %A, %F, %P, %D).

Character-type memory contents of up to 1000 bytes can be compared in a condition. Logical variables may be compared if a type modification is used to define a different storage type (e.g. `(ALOG%X EQ X'FF'):`). Among the keywords the subcommand execution counters, the AID registers, all program registers and the program counter (%PC) can be used for comparisons.

AID literals are likewise permitted as comparison operands. For character literals (C'x...x') AID always uses the code of the input mediums, that means the coded character set ot the terminal or the prozedure file with AID commands. If character-type memory contents are to be compared in ASCII, the comparison text must be converted into a hexadecimal literal, e.g. C'Hugo' has the hexadecimal value X'4875676F' in ASCII.

When a condition is formulated, the operand types must be compatible. The table on the next page shows which comparisons are permitted and how the comparison takes place. The permissibility of a comparison is not checked until the monitoring event has occurred. In the case of an error, AID issues an appropriate message and sets the comparison result to FALSE, i.e. the command section of the subcommand is not executed.

AID distinguishes between binary, character and numerical comparisons. AID derives the type of comparison from the type of the operands involved. AID converts and compares the operands of a condition according to a specific, language-independent algorithm because it is quite possible for a user to compare data items from modules that are written in different programming languages. Therefore the result of an AID comparison will not necessarily match the result of a similar comparison in a particular programming language:

– In the case of a character comparison the shorter operand is logically blank-filled, and AID compares two operands of the same length, whereas Fortran for example always evaluates the result of the comparison as FALSE if the operands involved are of different lengths.

– In the case of a binary comparison, the operand is padded with X'00' to the right and the subsequent procedure is the same as for character comparison.

– In the case of numeric comparisons, different results may arise from the fact that AID does not work to the same degree of precision as the respective programming language during the conversion of the operands.

– COBOL assigns the numerically edited variables to the numerical operands; for AID, these variables belong to the character memory type.

Particular attention must be paid to this situation if it is intended to compare operands from various different programming languages.

The comparisons that are made using the relational operators serve as operands for the Boolean operators. Logical variable such as those used in Fortran cannot be used in this case.
Boolean operators also enable more than two comparisons to be linked with each other: the upper limit is determined by the complexity of the comparison operands and the size of the internal AID input buffer.

The following table shows how the various operand types are compared with each other and which comparisons are not permitted.

storage type 1st operand	storage type 2nd operand					
	%X X' f...f' B' b...b'	numeric	%C C' x...x' U' x...x'	%UTF16/ NATIONAL	numeric Literal	Pointer
%X X' f...f' B' b...b'	bin	bin	bin	bin	–	bin
numeric	bin	num	num(1)	numchar UTF16-chr	num	–
%C C' x...x' U' x...x'	bin	num(1)	char	UTF16-conv UTF16-chr	num(1)	–
%UTF16/ NATIONAL	bin	numchar UTF16-chr	UTF16-conv UTF16-chr	UTF16-chr	numchar UTF16-chr	–
numeric Literal	–	num	num(1)	numchar UTF16-chr	num	–
Pointer	bin	–	–	–	–	bin

bin: Binary comparison
Comparison takes place bitwise from left to right. The shorter operand is padded with zeros (B'0') to the right.

char: Character comparison
Comparison takes place bytewise from left to right. The shorter operand is padded with blanks (X'40') to the right.

num: Numerical comparison
The arithmetical values of the two operands are compared.

numchar
The printable numeric string in UTF16 encoding from the integer numeric field is used for the comparison.

UTF16-chr
Character comparison in UTF16 encoding
Comparison takes place bytewise from left to right. However, UTF16 characters are used for padding and truncation (blanks in 2-byte encoding).

Ordering relations which are used in EBCDIC encoding are no longer supported by AID for the bytewise UTF16 comparison.

UTF16-conv
: If an operand is of the type %UTF16 and the operand to be compared is of the type %C or a C/U literal, it is converted to %UTF16 by UTF16-conv. The comparison can then take place as with UTF16-chr.

num[1]
: If a character-type operand contains only digits and is no more than 19 characters in length, it is compared numerically, provided the second operand is of the numeric type.
All other character-type operands cannot be compared with numeric storage types or numeric literals.

–
: Comparison not possible

An attempted comparison is rejected with an error message and the result is set to FALSE.

Numeric storage types:

%A, %Y (corresponds to %AL2)	unsigned integer
%F, %H (corresponds to %FL2)	signed integer
%P	packed number
%D	floating-point number
%PC	program counter
all registers	
%•[*subkdoname*]	execution counter

and all symbolically addressed numeric-type data items.

Not all data items treated numerically in the various programming languages have a numeric storage type in AID; for details see the language-specific AID manuals (%SET table).

Numeric literals:

```
[{±}]n                          integer
#f...f'hexadecimalnumber'
[{±}]n.m                        decimal number
[{±}]mantissaE[{±}]exponent     floating-point number
```

Storage types %S and %SX are not very useful for comparisons and are therefore not listed in the above table. %S is treated like %XL2 and %SX like %XL4.

Further information on storage types and literals can be found in chapter “AID literals” on page 101 and chapter “Keywords” on page 109.

Examples

1. `%IN S'18' <(%• LT 10): %D I,J; %MOVE X'58' INTO V'348'>`

   When the program reaches statement 18, AID interrupts the program run and checks the subcommand condition. On the first nine times the command section is executed, i.e. AID outputs the values for variables I and J on the screen and sets the content of virtual address V'348' to X'58'. As the subcommand does not contain a %STOP the program run is resumed at statement 18.
   On each further pass of test point S'18' the command section of the subcommand is skipped.

2. `%IN S'25' <INS25: (ACHAR EQ 'END'): %D ISUM,JSUM; %STOP>`

   The subcommand of test point S'25' is not executed until the field with the symbolic address ACHAR has the content 'END'. AID then displays the contents of the sum fields ISUM and JSUM on the screen and the program switches to command mode.

3. `%CONTROL1 %IO <OUTPUT: (SLF NE 200): %D %•, OUTDAT, SLF P=MAX>`

   On testing a program which is supposed to output records with a length of 200 the record length is found to be incorrect at times. The command %CONTROL1 can be used to monitor the record length field SLF. Every time a record is output and SLF does not contain the value 200, AID writes the contents of the execution counter and of the fields OUTDAT and SLF to SYSLST.

4. `%INSERT S'18' <IN1:(I EQ 15): %SET 1 INTO J; %D ARRAY(K),K;%STOP>`

   A test point is set for statement number 18. A subcommand with the name IN1 is entered for this test point: whenever index I has the value 15, index J is set to 1 and the vector element ARRAY(K) is output with the associated index K. The program is then suspended.

5. `%SET 0 INTO %0G`

```
%INSERT S'25'      <CN1: (CODGT NE '3'): %SET %L=(1 + %0G) INTO
%0G;-
                     %D %•, %0G, CNO, OUTDAT P=MAX>
```

   The %SET sets AID register %0G to 0. The %INSERT sets a test point for statement 25 and defines a subcommand with the name CN1. Whenever the test point is reached, AID increments the counter %•CN1 by 1. Only if the code digit CNO at the test point is '3' will register %0G be incremented by 1, with AID writing the counter statuses, the contents of the code digit CNO and the output record OUTDAT to SYSLST (P=MAX).

In other words, the content of %•CN1 shows how often the program executes statement 25, and %0G counts how often the code digit CNO at the test point is not equal to 3.

6. `%IN PROC <(%3 GT 4096 AND %5->%L1 EQ 'A'): %SET %L=(%5 + 2) INTO %5>`

   Prior to execution of the PROC statement, AID checks whether register %3 contains a value > 4096 and whether the memory location referenced by register %5 contains an 'A' at the same time. If so, the content of register %5 is incremented by 2 and program execution continues.

6.4 Chaining

On input of several %INSERTs for the same *test-point* or several %ONs for the same *event* AID prefixes the last subcommand to the preceding one (LIFO principle). One exception is the *write-event* with %ON. Chaining is not possible in this case; the command that is entered last overwrites the previous one. AID draws attention to this with warning AID0496. Commands are also chained if the newer command does not have an explicit subcommand; in this case the implicitly generated <%STOP> command is prefixed to the subcommand already entered, i.e. the older subcommand is not executed any more. It is expedient, however, to first delete the subcommand no longer required, for otherwise AID has to administrate a "deadwood" entry throughout the remaining debugging sequence.

If a chained subcommand contains a condition, this condition applies for the associated command section only. The subcommands ensuing in the chain are handled in one of two ways:

– The conditional command section is concluded with %CONTINUE, %RESUME, %TRACE or %STOP. Ensuing subcommands are processed only if the condition result is FALSE.

– The conditional command section does not contain any %CONTINUE, %RESUME, %TRACE or %STOP. Ensuing subcommands are always processed regardless of whether the condition result is TRUE or FALSE.

Subcommand chaining for %CONTROLn is not possible. A new %CONTROLn overwrites all operand values of an earlier %CONTROLn for the same number n with entries from the new command.

Examples

1. The following commands are entered in the test run:

```
%ON %LPOV(SUBTOT)
          .
          .
          .
%ON %LPOV(SUBTOT)      <%DISPLAY S=B1@.PROC=B1.CHAR_DAT>
```

   On input of the first %ON, AID adds <%STOP> as a subcommand, because no subcommand has been explicitly specified. Chaining after input of the second %ON results in the following subcommand for the %LPOV(SUBTOT) event, i.e. after the SUBTOT module has been loaded:

```
<%DISPLAY S=B1@.PROC=B1.CHAR_DAT; %STOP>
```

2. The following example shows the effect of LIFO chaining with a <%STOP> command inserted by default (implicit subcommand).

```
%INSERT ST4  <%D TEXTDAT>
                    .
                    .
                    .
%INSERT ST4
```

The second %INSERT contains no subcommand, therefore AID adds a <%STOP> command. Since the second %INSERT designates the same test point as the previous one, it is prefixed and leads to the following chained subcommand sequence:
`<%STOP;%DISPLAY TEXTDAT>`

As the execution of a subcommand is aborted by %STOP, the `%DISPLAY TEXTDAT` command will never be executed; but it remains registered as a subcommand for test point ST4 and cannot be deleted from the chain either, because it has no name. It is best to assign a name to each subcommand so that there is always the possibility of deleting a subcommand from the chain via its name in the event of chaining being incorrect by mistake.
In the above example it would have been better to delete the first %INSERT via

```
%REMOVE ST4
```

and then enter

```
%INSERT ST4
```

3. The following %INSERTs can be used in a procedure in order to search for a character literal. In the event of a hit, the located address is stored in AID register %0G and the length of the desired string is stored in %2G (see %FIND).

```
%INSERT V'1648' <(%0G NE -1): %SET %L=(%1G - %0G) INTO %2G>
%INSERT V'1648' <%FIND C'x...x'>
```

Chaining is necessary because a condition can only be stated at the beginning of a subcommand. It is only through LIFO chaining that the required subcommand is generated for test point V’1648’:

```
<%FIND C'x...x'; (%0G NE -1): %SET %L=(%1G - %0G) INTO %2G>
```

```
4. %INSERT S'50' <%D NO,INDAT; %STOP>

   %INSERT S'50' <(ISW EQ X'FF'): %SET X'00' INTO ISW; %CONT>
```

The two %INSERTs for the same source reference, i.e. statement 50, result in the following conditional subcommand with a THEN and an ELSE branch at test point S'50' (the various parts of the construct are marked with IF, THEN and ELSE to make it easier to read):

```
(ISWITCH EQ X'FF'): %S X'00' INTO ISWITCH; %CONT; %D NUMBER, INDAT; %STOP#1
↑                   ↑                      ↑
IF                  THEN                          ELSE
```

Whenever statement S'50' is about to be executed, AID interrupts the program sequence and checks the content of switch ISW. If the switch contains the value X'FF', it is reset to X'00' and the program is resumed. Otherwise AID outputs the contents of NO and INDAT and halts the program.

6.5 Nesting

The subcommand of an %INSERT or %ON may contain another %INSERT or %ON. This phenomenon is known as subcommand nesting and is supported by AID over several subcommand levels. The depth to which subcommands can be nested is dependent on their complexity and on the size of the internal input buffer for AID.
Nested subcommands take effect step by step. While the monitoring condition of the first generation (outer level) is immediately entered by AID and can thus cause an interrupt already in the ensuing program sequence, AID does not enter the *test-point* (%INSERT) or *write-event* or *event* (%ON) in more recent subcommands until the monitoring condition of the immediately preceding generation has triggered an interrupt. If another, different command occurs within %INSERT or %ON nesting for a monitoring condition already entered, the new subcommand is additionally prefixed to the older one (LIFO principle). In contrast, a subcommand for a test point or an event of an inner nesting structure will not be chained if the test point of the next higher level of nesting is passed through several times because of a program loop or if the event in the outer nesting structure occurs more than once.

Subcommands for a %CONTROLn cannot be nested, which is why the commands %CONTROLn, %INSERT and %ON are illegal in the subcommand of a %CONTROLn (apart from the commands that are never allowed in any subcommand). In addition, it is not permitted to specify a %JUMP (COBOL85, FOR1) in subcommands of a %CONTROLn.

Examples

1. `%IN ST3 <%DISPLAY 'INSERT1', TEXTDAT;%IN OUTPUT <%D 'INSERT2', I,J,K,-`
   `NUM-TAB; %ON %SVC(186) <%D 'OPEN DAT1',I,J>>>`

   The example relates to a COBOL program. %INSERT ST3 defines paragraph ST3 as a test point; this %INSERT contains another %INSERT nested within it, which in turn contains a %ON command. The test point OUTPUT and the event %SVC(186) (≙OPEN) do not yet affect program execution. They are not activated until the test point of the %INSERT is reached in whose subcommand they are defined. When symbolic address ST3 is encountered in the program, the related *subcmd* is executed, i.e. the literal 'INSERT1' and the content of output record TEXTDAT are output and the test point OUTPUT is set. The subcommand for test point OUTPUT is not yet effective. The test points ST3 and OUTPUT have thus been set so far in the program to be tested.

   As the subcommand for test point ST3 does not contain a %STOP command, the program is resumed. When the address OUTPUT is reached in the program, `%DISPLAY 'INSERT2',I,J,K, NUM-TAB` is executed. In addition to this command, the subcommand contains a %ON for the event %SVC(186). If AID subsequently recognizes a SVC for opening a file, it executes the subcommand defined in the %ON: the literal 'OPEN DAT1' and the contents of indexes I and J are output.

2. `%IN ST4 <%D TEXTDAT>`
   `%ON %LPOV (SUBTOT) <%REMOVE ST4; %IN ST4 <%D 'SUBTOT LOADED'; %STOP >>`
   `%RESUME`

   Whenever test point ST4 is reached, AID outputs the memory contents of data field TEXTDAT. If the declared event %LPOV (SUBTOT) occurs, i.e. when the SUBTOT module is loaded, AID executes the subcommand in the %ON command. Test point ST4 is deleted, but a new subcommand is immediately entered for this test point:

   `<%DISPLAY 'SUBTOT LOADED'; %STOP>`

   When ST4 is encountered the next time, AID displays the text 'SUBTOT LOADED' and interrupts the program sequence; new commands can then be entered.

6.6 Deletion

The %REMOVE command is available for the deletion of subcommands. A subcommand is implicitly deleted when the associated monitoring command is deleted or, in the case of %INSERT, the associated test point or, in the case of %ON, the associated event.
A subcommand can be explicitly deleted via its name. This option only applies to subcommands of a %CONTROLn or %INSERT. As it is not possible to chain subcommands for a %CONTROLn, the effect of a %REMOVE %•*subcmdname* is the same as %REMOVE %CONTROL*n*. With %INSERT, however, it is possible to chain a whole series of subcommands consecutively for a certain test point. In this case %REMOVE %•*subcmdname* removes a single subcommand from the chain via its name. If the subcommand does not have a name, it can only be deleted together with the entire test point. It is therefore always advisable to assign a name to subcommands.

In the case of nested subcommands it is not possible to remove inner subcommands (not even via their names) from the nesting structure if they have not yet been entered at the associated test point. Such subcommands can only be deleted together with the entire %INSERT (%REMOVE %INSERT) or with the test point (%REMOVE *test-point*).

The current subcommand can be deleted immediately after it has been executed if %REMOVE %• is written as the last command in the command section. The %REMOVE %• is executed immediately; this has the effect that any commands which follow will also be deleted and therefore can no longer be executed.

7 Addressing in AID

The commands for execution monitoring, the %JUMP command (specifying a continuation address) and the commands for the output and modification of memory contents require operands which identify an address or a specific area in the memory. An address must be specified in the executable part of the program for %DISASSEMBLE, %INSERT, %JUMP and %REMOVE. %CONTROLn and %TRACE each require an operand which is a memory area in the executable part of the program, whereas in the case of the %DISPLAY, %FIND, %MOVE and %SET commands the specified memory area may also be in the data section of the program.

In AID, an address is designated by an address constant or by a complex memory reference. A memory area can be specified by a qualification or a memory reference, dependent on the command, or an area can be defined by two addresses; the area then lies between the first and second address. A detailed description of the operands that have to be specified is given in the descriptions of commands in the language-specific manuals and in the manual for debugging on machine code level.
The %SDUMP command has a special status; its associated operand *dump-area* designates either a name range, which can be specified with a qualification, or a single data item.
The following sections contain descriptions of all terms that can be used to designate an address in AID, the various qualifications, and the simple and complex memory references.

7.1 Qualifications

Qualifications define the path to a memory object which is outside the currently valid AID work area or which is not within the current main program or subprogram, or which is not unique there. In some cases addressing may end with a qualification, i.e. the qualification can reference the memory object itself. There is a distinction between the base qualification and area qualifications. Qualifications are always specified in the order from the higher-ranking to the lower-ranking qualification, and only to the extent that is necessary for unique path identification. Redundant qualifications are ignored by AID.

Successive qualifications are separated by periods. A period must also be placed between the last qualification and the ensuing address section.

%QUALIFY is used for predefining qualifications. A prefixed period in an address operand will fetch these predefined qualifications.

7.1.1 Base qualification

The base qualification identifies the environment, i.e. it determines whether an ensuing address is to be located in virtual memory or in a dump file. The base qualification is equally applicable for symbolic and machine-oriented debugging.

E=VM Default value; designates the virtual memory area of the loaded program.

E=Dn Designates a memory dump in a dump file with a link name from the range D0 - D7; the dump file must have been assigned via %DUMPFILE.

The base qualification can be globally defined with %BASE or specified in the address operand for an individual memory reference. The base qualification is permitted as the only operand in the %BASE, %QUALIFY and %SDUMP commands. In all other address or area operands a base qualification must be followed by one of the following terms:

– area qualification
– data name
– statement name
– source reference
– virtual address
– keyword

7.1.2 Area qualifications

An area qualification designates a certain subarea of a program. The various program subareas are defined/named during programming, compilation or linkage. Area qualifications are specified when an address does not reside in the program segment which is being executed. There are different area qualifications for debugging on machine code level and for the various programming languages. The area qualifications for symbolic debugging are determined by the structure of the relevant language; chapter “Prerequisites for debugging with AID” on page 33 of the language-specific manuals describes which program segments are referenced by which qualifications.

SPID=X’f...f’	ESA systems, machine code
ALET={X’f...f’|%nAR|%nG}	ESA systems, machine code
CTX=context	symbolic and machine code
L=loadunit	machine code
O=objectmodule	machine code
C=csect/segmentname/sharename	machine code / COBOL
COM=common	symbolic and machine code
S=srcname	all programming languages
PROC=name	all programming languages
PROG=name	Assembler, COBOL, FORTRAN
ONUNIT=’onunitname’	PL/I
BLK=’blkname’	C++/C, PL/I

Area qualifications are specified in the address operand for a memory reference, where they are used for path description. Only those qualifications which are required for unique referencing need be specified. However, if the interrupt point is in the routines of the runtime system, data and statements can only be referenced in their own program via the full qualification.
In commands which require an area operand it is permissible to use any area qualifications apart from SPID and ALET in order to designate an area. C=*csect* and COM=*common* can also be used as the start address. All area qualifications can be declared as prequalifications with %QUALIFY.

In a complex memory reference it is essential that the subsequent operations do not exceed the area limits. Although the length attribute of an area qualification cannot be accessed, a check is made as to whether the result of a byte offset or a length modification is still within the area specified in the qualification.

The **ALET and SPID qualifications** can only be used on ESA systems. They identify a data space, and can only be used before a virtual address or a complex memory reference which is formed without symbolic components.

The **context qualification** identifies the context in which the memory areas or addresses referenced by subsequent qualifications or address data are supposed to lie. It is only necessary if a CSECT, COMMON or compilation unit is contained in a number of contexts and the current interrupt point is not located in the CSECT, COMMON or compilation unit in which the memory object selected by the address operand is contained.

The context qualification is specified by CTX=context. Here, *context* is the name assigned explicitly in the BIND macro with the LNKCTX[@] operand, or the implicit name LOCAL#DEFAULT if LNKCTX[@] has not been specified. Programs loaded dynamically with the DBL are given the same context name, assigned as the default: LOCAL#DEFAULT. Programs linked statically with TSOSLNK are assigned the context CTXPHASE. Other contexts of program may result from connection to a shared code program (for example to a DSSM subsystem or to a program in a COMMON MEMORY POOL).
The *prequalification* operand of the %QUALIFY command and the *dump-area* operand in the %SDUMP may end with CTX=*context*. In all other address or area operands a CTX qualification must always be followed by one of the following:

– another area qualification

– data name

– statement name

– source reference

The **L and O qualifications** are specified when it is necessary to describe the path to one of several CSECTs or COMMONs of the same name. All that need be specified is the L and/or O qualification that is sufficient to provide unique reference. An L and/or O qualification must always be followed by a C or COM qualification.

The **C and COM qualifications** can be used as area specifications in the %CONTROLn and %TRACE commands. If the C qualification is used to designate a CSECT or the COM qualification to designate a COMMON, only a machine code criterion may be specified. The specification C=*sharename/segmentname*, which can be used when debugging COBOL programs, may only be combined with a symbolic criterion. However, a C qualification can never be followed by a symbolic memory reference, not even in COBOL.

In the %DISASSEMBLE, %INSERT and %REMOVE commands the start address of the CSECT or COMMON is specified with the C or COM qualification respectively. Similarly, in the %DISPLAY, %FIND, %MOVE, %ON %WRITE(...) and %SET commands the address operand can end with C=*csect*/COM=*common*. The effect of this is to reference the entire CSECT or COMMON. In these commands the CSECT or COMMON is used as a machine-code memory reference. The C/COM qualification can also be used as a memory reference within a complex memory reference (see section “Machine code memory references” on page 72).

The **S, PROC, BLK, ONUNIT and PROG qualifications** can be used to identify a memory area in %CONTROLn and %TRACE or the name range in %SDUMP. These qualifications stand for the entire program segment specified; they cannot be used as memory references however.

An S qualification can be followed by a:

– PROC, BLK or ONUNIT qualification
– data name
– statement name
– source reference

A PROC, BLK or ONUNIT qualification can be followed by a:

– data name
– statement name
– source reference

The **PROG qualification** is a combination of S=srcname•PROC=name if *srcname* and *name* are identical. It can be used in Assembler, COBOL and Fortran and can be employed for the Assembler, COBOL and Fortran languages.

Examples

```
1. %BASE E=VM
   %DUMPFILE D1=M.DUMP
   %DISPLAY V'10A', E=D1.V'10A'
```

%BASE defines the virtual memory area of the loaded program as the base qualification. %DUMPFILE assigns the link name D1 to the file M.DUMP.
As the base qualification E=VM applies, AID outputs four bytes as of address V'10A' of the loaded program for the first entry in the %DISPLAY command and four bytes as of address V'10A' from a dump for the second entry. This dump resides in a file that was assigned to link name D1 via %DUMPFILE.
Four bytes constitute the implicit length of a V address.

```
2. %DISPLAY S=COMPUTE@.PROC=COMPUTE.SUM
```

AID outputs the contents of the variable SUM from the program unit COMPUTE@ of a Fortran program. The S and PROC qualifications are necessary if the program has been interrupted in different program unit.

```
3. %QUALIFY E=D1.S=COMPUTE@.PROC=COMPUTE
   %DISPLAY .SUM
```

AID prefixes the defined prequalification to the period preceding SUM. This results in the command: `%DISPLAY E=D1.S=COMPUTE@.PROC=COMPUTE.SUM`
AID outputs the data field SUM from program unit COMPUTE@ residing in the dump file assigned to link name D1.

4. `%DISPLAY E=D2.S=TEST@.BLK='23'.var`

   The dump file with the link name D2, which contains the memory dump of a C program, and within that the compilation unit with the code module name TEST@, contains the local variable var, in the block starting in line 23. AID outputs the contents of the variable.

5. `%MOVE L=LAD1.C=CS1.(%L(L=LAD1.C=CS1) - 4) INTO %2G`

   AID transfers the last four bytes of CSECT CS1 from load unit LAD1 to AID register %2G.
   The L qualification is necessary because CS1 is not the current CSECT and the name CS1 is not unique within the program system.

6. `%INSERT S=COMPUTE@.PROC=COMPUTE.S'16' <%DISPLAY %•>`

   AID sets a test point for statement 16 in program unit COMPUTE@. When this test point is reached in the program sequence, the execution counter is output and the program continues.

7.2 Memory references

A memory reference can be used in an AID command to address a memory object. If the memory reference defines a string, e.g. of the type %C or %UTF16, the character conversion function %C() or %UTF16() can be applied to the memory reference.
AID distinguishes between simple and complex memory references.
Example of simple memory references include:

- virtual addresses: V'f...f'
- data names: VAR1, FIELD(I)
- keywords: %14, %2D, %PC, %CLASS6
- C qualifications: C=CS1
- COM qualifications: COM=CB
- statement names: L'20', COMPUTE1
- source references: S'133', S'44ADD'

Examples of complex memory references are:

- %@(VAR1)->.(%L=(I+5))%XL20
- C=CS1.#100'%SX->%CL8'

The simple memory references (i.e. the machine code and symbolic memory references and the keywords), together with their attributes and characteristics, are described in section "Machine code memory references" on page 72.

A complex memory reference is an instruction to be used by AID for calculating an address or by the user for modifying the attributes of a memory object. In a complex memory reference the user may incorporate symbolic and machine code memory references, keywords, constants and AID literals for byte offset, indirect addressing, type/length modification and address selection operations in order to determine the type and length of a memory reference or to cause AID to compute an address required in a particular test situation. Information on complex memory references and their associated operations is given in section "Symbolic memory references" on page 74.

Attributes

Attributes describe the characteristics of a memory object or of a constant.
Memory objects have up to six attributes:

- name (optional)
- address
- content
- length
- storage type
- output type

Selectors can be used to access the address, length and storage type attributes. Modification is used to alter the length or storage type. The length attribute also defines the relevant area: address to address + (length - 1). The limits of this area are checked in the case of length modification and byte offset operations. During a transfer with %MOVE, a check is made to see whether *sender* fits in the area limits of *receiver*. An exception is the virtual address, which is assigned the entire user address space as its area although the length attribute is only 4 bytes.

The manner in which AID takes account of the attributes in the individual commands and the checks performed on this occasion are described for the respective commands in the language-specific manuals and the manual for debugging on machine code level.
The AID mechanism for incorporating these attributes in the calculation of a complex memory reference is described under the various operations in the present chapter.

Constants do not have an address attribute and can thus be used in special cases only; in particular they cannot be subjected to address selection.

7.2.1 Machine code memory references

The CSECTs, COMMONs and virtual addresses are machine code memory references.

The **CSECTs and COMMONs** are specified in the form of a C or COM qualification with C=*csect* and COM=*common* respectively. As the CSECTs/COMMONs are specified in the same way as qualifications and can also be used in the same way as qualifications in certain commands, they are also described in section “Area qualifications” on page 67.

As a memory reference, the C/COM qualification has the following attributes:

```
Name
Address
Content
Length (length of CSECT/COMMON)
Storage type (%X)
Output type (dump)
```

The area limits are defined by the start address and the length of the CSECT/COMMON.

The following operations can be used on a C/COM qualification:

– address selector
– length selector
– byte offset
– type modification
– length modification

A **virtual address** is specified in the following format:
V'f...f', where *'f...f'* is a hexadecimal number of up to 8 digits between '0' and '7FFFFFFF'.
A virtual address directly references a memory location in the loaded program or in a dump file. On ESA systems it is therefore also possible to reference a memory location in a data space, for which it is necessary to specify an ALET/SPID qualification before the virtual address. Otherwise the only meaningful entry before a virtual address is a base qualification.

The result of a byte offset or of indirect addressing is also a virtual address, and therefore also has its attributes.

The attributes of a virtual address are as follows:

```
Address (f...f)
Content
Length (4 bytes)
Storage type (%X)
Output type (dump)
```

The area limits extend from V'0' to V'7FFFFFFF'.
Unlike all other memory objects, for which address and length at the same time define the area limits, virtual address operations have the entire user address space at their disposal, the only restriction being that the lowest address V'0' and the highest possible address V'7FFFFFFF' must not be exceeded.

A virtual address may be followed by:
– byte offset (•)
– indirect addressing (->)
– type modification
– length modification

Examples

```
1. %DISPLAY V'100'->->->%C
```

The four bytes as of address V'100' have the content X'00000A1A'. Address V'A1A' has the content X'0000000F' (first pointer operator). Address V'F' has the content X'0000B001' (second pointer operator). Address V'B001' has the content X'F1F2F3F4' (third pointer operator). AID interprets this as characters and outputs '1234'.

```
2. %MOVE E=D1.V'206'.(%1)->.(%2-5) INTO %2G
```

In dump file D1, address V’206’ is the starting point for a byte offset expressed in terms of the content of register %1 (X’00000004’). The memory contents (X’0000B111’) there (V’20A’) are used as the address for a pointer operation. From this new address (V’B111’) an offset expressed by the content of register %2 (X’00000008’) minus 5 is made, and from the address thus obtained (V’B114’) four bytes are transferred to AID register %2G.

7.2.2 Symbolic memory references

Symbolic memory references are the symbolic addresses which the compiler stores in the LSD records in the course of compilation. They include the names of data and statements assigned by the user in the program, in other words labels, entries or function names, and the source references generated by the compiler, via which every executable statement of a program can be referenced, regardless of whether the statement has a label or not. If LSD records have been created and are available, therefore, AID is able to access the associated addresses and the attributes linked to the addresses via data names or statement names or via source references.

Statement names and source references are address constants and only become a memory referenced when they are followed by a pointer operator. Without a pointer operator they can only be used in those commands which require an address as an operand. However, if it is intended to reference the instruction code that is at the corresponding address in the memory, the pointer operator must be added.

7.2.2.1 Data names

Data names are names of variables, data structures, fields, matrixes or vectors, depending on the language tools and terminology of the programming language involved. The items of tables or structures can be accessed in AID just like in a programming language statement, i.e. by placing the requisite identifiers, indexes or subscripts after the data name. For any exceptions see the command descriptions in the language-specific manuals.

Constants defined in the source program are likewise regarded as data names. They are specified, for instance, via EQU (Assembler), via *literal* and *symbolic character* in the SPECIAL NAMES paragraph (COBOL), or via PARAMETER (FORTRAN). As they do not occupy memory space, however, they cannot be used in the same way as all the other data. They have no address attribute; only the value of the constant is available to AID. The remaining attributes cannot be used.

The attributes of data names are defined in the source program, except for the output type, which AID determines on the basis of the storage/output type assignment (see section “General storage types” on page 109).

Data names have the following attributes:

```
Name
Address
Content
Length
Storage type
Output type
```

The area limits are defined by the address and the length.

Data names can be used in all commands addressing the data section. Selectors support access to the address, length and storage type attributes so that results can be output, transferred or modified or switchover to the machine code level can take place.

A data name may be subjected to or followed by:

– address selector
– length selector
– type selector
– character conversion function
– byte offset (•)
– length modification
– type modification
– indirect addressing (->), provided the data name is of type %A

A type modification serves to alter the storage type or the associated output type.
For the %DISPLAY and %SET commands, the memory contents must match the storage type defined in the type modification.

A length modification serves to alter the length associated with a data name. The data type is not retained, AID assumes storage type %X.
Length modification must not lead to a transgression of the area limits, i.e. the modified length must not exceed the implicit length from the length attribute.

If a deviation from the implicit attributes of a data name is desired, the address selector can be applied to the data name followed by a pointer operator. %@(dataname)-> then references the virtual address of a data name, which means the attributes of a virtual address take effect.

Indexes and subscripts

If a data name is the name of a tabular structure, it may be indexed in the same way as in a programming language statement. COBOL distinguishes between indexing and subscripting, although subscripting corresponds to indexing in other programming languages. Special features of how COBOL indexes are handled by AID are described in the User Guide "Debugging of COBOL Programs".

The index can be specified as follows:

```
{n                    }
{dataname             }
{arithmetic expression}
```

n

Integer with a value $-2^{31} \leq n \leq 2^{31}-1$.

dataname

Index defined for the vector, or numeric variable situated in the same program segment as the vector; i.e. the qualification of the vector is taken over for the index.

arithmetic expression

The value for *index* is calculated by AID. Permissible are the arithmetic operators (+, -, /, *) and the above-mentioned operands *n* and *dataname*. For COBOL it is the case that only the subscript, not the index, can be used in an arithmetic expression.

You can specify a range of indexes:

index1:*index2*

This designates the range between *index1* and *index2*. Both must lie within the index limits, and *index1* must be less than or equal to *index2*.

Examples

```
1. %DISPLAY V'10A'%T(SYMBOL)
```

The memory contents as of address V'10A' are interpreted with the storage type of SYMBOL and are output in the associated output type and length of SYMBOL. AID checks that the memory contents of V'10A' and the storage type of SYMBOL are compatible.

```
2. %DISPLAY %@(DATARECORD)->.4%T(INPUT)
```

DATARECORD has no data structure description; however, the structure of INPUT corresponds to that of DATARECORD with the restriction that a 4-byte number is situated at the beginning of DATARECORD. The address selector and subsequent pointer operator reference the virtual address of DATARECORD, i.e. the area limits

are no longer binding. A byte offset skips the first four bytes of DATARECORD. The type selector defines the storage type and the length of INPUT, and the memory contents as of the calculated address are output accordingly.

7.2.2.2 Statement names and source references

Statement names are names assigned in the source program to labels, sections, paragraphs or label/entry constants/variables, depending on the language tools and terminology of the programming language involved.

Statement names stand for the address of the instruction code generated for the first statement following the label. They are specified in the following format:

L'number'	Label (Fortran)
L'name'	Label (all programming languages); in %DISASSEMBLE, %INSERT, %REMOVE possible without L'...' if not followed by address computation.
name	Function (C++/C), program name (COBOL, Fortran, Assembler; can only be used in the %DISASSEMBLE, %INSERT and %REMOVE commands for identifying the program start), entry constant or variable (PL/I)

Source references are the numbers or names of statements, generated by the compiler, which are stored in the LSD records and via which the statements can be referenced which are neither at the start of a main program or subprogram nor have a label. The compiler-oriented statement designations and compiler listing entries can be found in the language-specific manuals.
Source references stand for the address of the instruction code generated for a statement. Source references are specified in the format S'number/name'.

Characteristics

Source references and statement names are address constants. They occupy no memory space, i.e. they have no address attribute and cannot be changed. Entry and label variables in PL/I are an exception; memory space is created for these as for all other data, and they can be overwritten via %SET.

If LSD records have been generated and are available, AID can use statement names and source references to access the instruction code.

Statement names and source references can be used as simple memory references in the commands %DISASSEMBLE, %JUMP, %INSERT and %REMOVE. Source references can also be used to specify a memory area in the %CONTROLn and %TRACE commands. In

%DISPLAY, %MOVE and %SET the value of the address constant is referenced, but can only be used as *sender*. An exception are the section and paragraph names in COBOL. Here, AID knows not only the address of the first command but also the end of the section or paragraph. Section and paragraph names can thus also be used as an area specification in the %CONTROLn and %TRACE commands.

Otherwise statement names and source references to be used for referencing a memory location must be followed by a pointer operator. Only the address constant is available to AID; the remaining attributes cannot be used.

A statement name may only be followed by indirect addressing (->).

Examples

```
1. %DISPLAY L'TOTAL'
   %DISPLAY L'TOTAL'->
```

The first %DISPLAY outputs the address of the instruction code generated for the first statement following the label TOTAL.
The second %DISPLAY outputs four bytes of memory contents as of this address.

```
2. %INSERT S'123'
   %INSERT S'123'->.(-6)
```

The first %INSERT sets a test point for the address of the instruction code generated for statement 123. The source reference S'123' is used as a simple memory reference here.
The second %INSERT sets a test point for the address of the instruction code which is located six bytes before the test point of the first %INSERT. This time, the source reference S'123' is used in a complex memory reference and therefore its function as an address constant must be observed and the pointer operator placed accordingly.

```
3. %MOVE L'123' INTO %2G
   %MOVE X'D2' INTO L'123'->
```

The address V'A1A' is stored in the LSD records for label 123 in a Fortran program. The first %MOVE transfers this address to AID register %2G. The second %MOVE transfers the hexadecimal literal X'D2' to the memory location with the address V'A1A'.

7.2.3 Keywords

Memory objects outside the program memory which are used by AID or by the program can be referenced by AID via keywords. This applies to general registers

%0 - %15, floating-point registers %nE, %nD and %nQ, the access register %nAR, the program counter %PC, AID registers %0G - %15G and %nGD, as well as the execution counter %•subcmdname. Class 5 and class 6 memories can also be addressed with the keywords %CLASS5, -ABOVE and -BELOW and %CLASS6, -ABOVE and -BELOW. All other keywords cannot be used as memory references. Only a base qualification can be specified before a keyword.
All keywords which can be used in AID are described in chapter “Keywords” on page 109.

Keywords have the following attributes:

```
Name (%name)
Address
Content
Length
Storage type
Output type
```

The area limits are defined by the start address and the length.

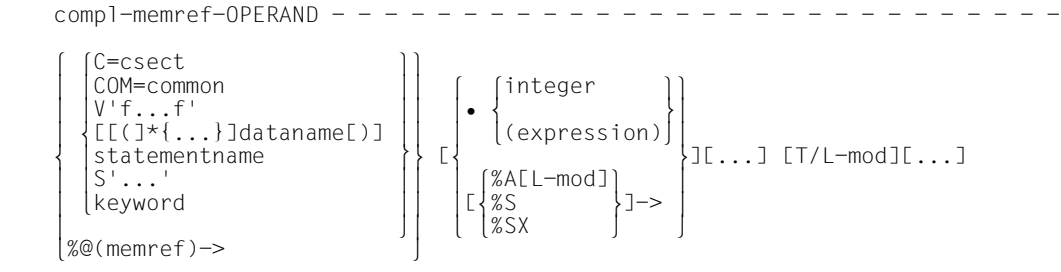
A keyword may be subjected to or followed by:

– address selector (the result is unusable if the address is located outside the user area)
– length selector
– byte offset (•)
– indirect addressing (->)
– length modification
– type modification

In the case of byte offset and length modification, AID checks the area limits. General registers may be used without type modification before a pointer operator, even though they are of type %F.

7.2.4 Complex memory references

A complex memory reference is where an address computation is carried out, on the basis of a symbolic or machine code memory reference or of a keyword. The result of a complex memory reference, without a final type and length modification, is a virtual address with storage type %XL4. The calculated address can, however, be assigned to the required storage type via a type modification or length modification or both.



C=csect	C qualification
COM=common	Common qualification
V'f...f'	virtual address
dataname	Data names
statementname	Statement names
S'...'	Source references
keyword	Keywords
%@(memref)	Address selector
T/L-mod	Type and/or length modification
%A, %S, %SX	Storage types for address interpretation
• {...}	Byte offset
->	Indirect addressing (pointer operator)
*	Indirect addressing (content operator, C++/C only)
(expression)	Arithmetic expression

Byte offset, indirect addressing, type and length modification, arithmetic expression and the address selector are described in the following sections. All other terms are explained at the start of this chapter.

7.2.4.1 Byte offset "•"

Byte offset enables byte-by-byte positioning forwards or backwards from a particular address. A byte offset always results in a virtual address. A byte offset must not exceed the area limits of the memory object involved.

```
byte-offset - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

          ⎧number       ⎫
memref • ⎨              ⎬
          ⎩(expression) ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

- Offset operator

memref

May be any memory location referenced in any manner:

virtual address, data name, keyword, C qualification or complex memory reference.

number

Positive integer (decimal or hexadecimal) between 0 and 2^{31}-1.

expression

Value between -2^{31} and 2^{31}-1 that is calculated by AID.

expression is described in section “Address, type and length selectors” on page 93. It may comprise numbers, numerical contents of memory references, the result of address/length selector and length function, and the arithmetic operators (+ - * /).

Byte offset can only be effected within the area limits of the relevant memory object and results in a virtual address with a length of 4. These four bytes must fit within the area limits of the memory object. If these limits are violated, AID issues an error message.

Except for virtual addresses, the area limits are determined by the start address and the length attribute. For a virtual address the entire virtual memory area (V’0’ through V’7FFFFFFF’) can be used. In the case of data names, the keywords %CLASS6, -ABOVE, -BELOW and the C qualification, the symbolic level may be left via address selection followed by a pointer operator: %@(...)-> thus switches to the area limits of a virtual address.

A byte offset may be followed by:

– byte offset (•)
– indirect addressing (->)
– length modification
– type modification

Examples

```
1. %DISPLAY SYMBOL.10
   %DISPLAY %@(SYMBOL)->.10
```

SYMBOL has a length of 10. An offset by 10 bytes cannot be executed by AID since this would reference the first four bytes after SYMBOL and thus violate the area limits of SYMBOL. AID issues an error message.

Address selection followed by a pointer operator switches to machine code level, where the area limits of a virtual address apply. A byte offset positions to the first byte after SYMBOL, as in the first %DISPLAY. AID can now execute this %DISPLAY and outputs the first four bytes after SYMBOL.

```
2. %D %@(VAR)->.(%L(ELEM(1))*5)%T(ELEM(1))
```

Let it be assumed that a COBOL program contains a vector ELEM with 10 elements ELEM(1) to ELEM(10). Variable VAR is to be redefined as a vector in the structure of ELEM, and its 6th element is to be output. The length of element ELEM from a table called TAB is to apply. From the start address of VAR, AID positions forwards via a byte offset using the value derived by multiplying the length of ELEM by 5. The consequence of subsequent type modification is that the contents are output at the calculated address in the type and length of an element of ELEM.
If ELEM were specified without an index, AID would assume the type and length of the entire vector ELEM.

```
3. %D %5->.(%L(INDEX)*%L(ADDRESS))%CL=(%L(ADDRESS)+50)
```

The content of register 5 (X'00000A00') is used as an address. As of address V'A00' an offset derived by multiplying the length of INDEX (2) by the length of ADDRESS (7) is effected. The memory contents at address V'A0E' (#A00'+(2*7)<String#A0E')areoutputincharacterformatwitha'length of 57 (7+50).

```
4. %D S'123COMP'->.8%S->%L10
```

Address '1B0' is stored in the LSD records for the COBOL source reference S'123COMP'. The pointer operator positions to the memory location with the address V'1B0'. An offset of 8 bytes is effected. The content X'600F0130' at the new location, i.e. at address V'1B8', is interpreted with %S. Base register 6 (content X'000B010') plus displacement #00F'resultintheaddressV'B01F',whichisreferenced'with the pointer operator. 10 bytes as of this address are output in dump format.

```
5. %D C=CS1.(%L(C=CS1))
   %D C=CS1.(%L(C=CS1)-4)
```

The first %DISPLAY is rejected since a byte offset with the length of CS1 would reference the first byte after CS1 and thus exceed the area limits of CS1.
The second %DISPLAY reduces the offset by four bytes. The area limits of CS1 are not violated, and AID outputs the last four bytes of CS1.

```
6. %D V'4'.(-5).4
   %D V'4'.(4-5)
   %D V'4'.4.(-5)
```

The byte offset in the first %DISPLAY is rejected since (-5) would violate the lower area limit of virtual addresses (V'0'), although the final result would be within the permissible range due to the second offset.
In the second and third %DISPLAYs, no offset exceeds the area limits and AID outputs four bytes as of address V'3'.

```
7. %D V'100' .(%1 + %2)
```

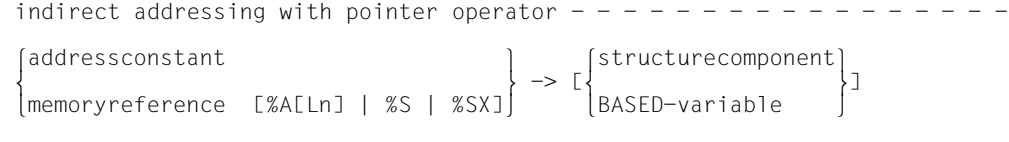
Address V'100' is incremented by the sum of the contents of registers 1 and 2.

7.2.4.2 Indirect addressing "->" / "*"

In indirect addressing AID uses an address constant or a memory content as an address for another memory location. If the pointer operator is used as a unary operator, the result is a virtual address. The pointer operation therefore causes transition to machine code level. If indirect addressing is carried out with the pointer operator as a binary operator or if the content operator is used, the user remains on the symbolic level even after indirect addressing, and the result is edited in accordance with the corresponding data definition from the source program.

In any 4-byte address used for indirect addressing, AID takes the current addressing mode of the test object into account. It can be interrogated with %DISPLAY %AMODE. A different address interpretation can be declared for the pointer operation with %AINT.

Pointer operator



-> Pointer operator

addressconstant

Address constant (statement name, source reference, or result of an address selection). Names of labels must be set before "->" in L'...'.

memoryreference

May be any memory location containing an address. Address-type data can be used without type modification.

[%A[Ln] | %S | %SX]

Type modification enabling a memory location to be interpreted as an address. %S and %SX simulate addressing as carried out by machine instructions. AID thus calculates addresses in the same way as the hardware, either from base register and displacement (%S) or from index register, base register and displacement (%SX). For details see section "Storage types for interpreting machine instructions" on page 110.

```
{structurecomponent}
{BASED-variable    }
```

In both of these cases the pointer operator is used in order to reconstruct indirect addressing which forms part of the language elements of the programming language, i.e. in order to reference a structure component in C++/C via a pointer or in order to reference a BASED variable via the associated pointer in PL/I The attributes of the structure components or BASED variables as defined in the source program apply to the result of the indirect addressing.

The content operator "*"

In C++/C it is also possible to use the content operator for dereferencing instead of the pointer operator.

The address referenced with the content operator is interpreted in accordance with its data type, declared in the program. There is no switch to machine code level as happens in the case of unary dereferencing by the pointer operator.
In contrast with C++/C, where the content operator can also be applied to vectors, the content operator in AID is only allowed for pointers.

```
indirect-addressing with content operator - - - - - - - - - - - - - - - - - -

[(]* {...} pointer-variable[)]

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

* Content operator

pointer-variable

type-specific pointer of a C++/C program

The content operator can be repeated several times. It may be necessary to define the order of processing by bracketing. The content operator is evaluated at a lower priority after the pointer operator, byte offset and indexing.

Indirect addressing may be followed by:

– byte offset (•)
– indirect addressing (->)
– length modification
– type modification

Examples

1. `%DISPLAY V'10A', V'10A'->`

AID output

```
/%DISPLAY V'10A', V'10A'->
 V'0000010A' = ABSOLUT  + #0000010A''|
 0000010A (0000010A) 00000478                                  ....
 V'00000478' = ABSOLUT  + #00000478''|
 00000478 (00000478) E3C5E7E3                                  TEXT
```

AID outputs four bytes as of address V’10A’ in dump format. For the second operand, AID uses this memory content (X’00000478’) as the address in a pointer operation and outputs four bytes as of address V’478’ in dump format.

2. `%FIND C'***'`

`%DISPLAY %1G->`

%FIND searches the memory for the string ’***’. If AID locates the string, AID register %1G holds the continuation address, i.e. the address of the first byte following the located string. %DISPLAY outputs the memory contents following the search criterion.

```
3. %SET %7 INTO V'14C0'%SX->
   Content of general register 4:X'00000100'
   Content of general register 6:X'00004000'
   Memory contents as of address V'14C0':X'50746B00' ≙ ST
                                                R7,X'B00'(R4,R6)
```

AID simulates transfer using the 'store' instruction (ST, instruction code X'50') and transfers the content of general register 7 as of address V'4C00'. AID calculates the address from memory content X'50746B00' as follows:

```
X'507' is ignored
X'4' use content of register 4:      '00000100'
X'6' add content of register 6:      '00004000'
X'B00' add displacement:                  'B00'
_____________________________________________
results in the address                  '4C00'
```

```
4. %SET X'C1C2C3C4' INTO V'14C2'%S->
```

The register and memory contents are the same as in example 3.

This %SET transfers the hexadecimal literal X'C1C2C3C4' to the memory location with the address V'4B00'. AID calculates the address from memory content X'6B00' as follows:

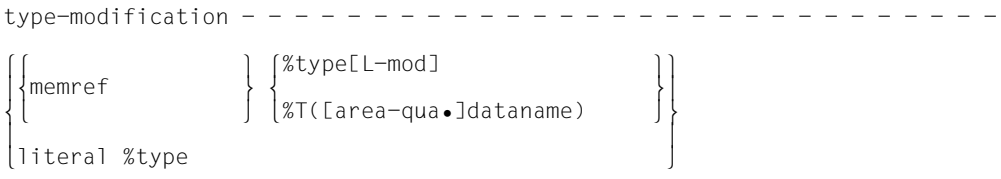
```
X'6' use content of register 6:   '00004000'
X'B00' add displacement:               'B00'
_____________________________________________
results in the address                 '4B00'
```

7.2.4.3 Type modification

Type modification is used to give a memory content an interpretation other than suggested by its storage type attribute. This may be necessary in the following cases:

- type matching for %SET
- differing output format for %DISPLAY
- conversion of a literal (only allowed for %DISPLAY)
- interpretation as an address before a pointer operator
- interpretation/editing in a different structure (redefinition of a memory location)
- interpretation as an integer in an expression

Type modification is only expedient before a pointer operator and at the end of a complex memory reference in order to interpret the storage content as of the calculated address) or the literal in the required storage type.



memref

May designate any memory location referenced in any manner:

virtual address, data name, keyword, C qualification

literal

The AID literals are described in chapter “AID literals” on page 101.

%type[L-mod]

Keyword for storage types with optional length specification:
%X, %C, %P, %D, %F, %A and %UTF16 (see chapter “Keywords” on page 109).
The length can be specified via all length modification options. If no length is specified, the length attribute of the modified memory object is retained.
The storage types %H, %Y, %S and %SX have a fixed length and cannot therefore be used with a length specification.
In the case of storage type %UTF16, the length must be a multiple of 2.
Length modification is not permitted for literals.

%T([area-qua•]dataname)

The type selector interprets a memory location with the storage type and length of other data definitions. This implies that the rules of length modification must be observed (e.g. no transgression of area limits).
dataname may be qualified, i.e. derived from a different program segment; no base qualification is permitted however.

During type modification AID checks whether the memory contents match the selected storage type. If this is not the case, AID issues an error message.

Each storage type is assigned to an output type (see chapter “Keywords” on page 109). This means the type modification can be used to change the output type.

The storage types %D, %P, %F and %A have only certain permissible lengths (see chapter “Keywords” on page 109). If they are used without a length specification, the length of the modified memory object must match one of the lengths permitted for the storage type. Otherwise AID rejects the type modification and reports a length error.
The storage types %S, %H and %Y have a fixed length of 2 bytes, %SX has a fixed length of 4 bytes. They effect an implicit length modification, which must be able to take place within the defined area limits.

The type modification with storage type %UTF16 is permitted if the (implicit) length of the memory location is a multiple of 2.
The type modification %UTF16 is permitted for X literals but forbidden for C and U literals.

As the %SET command takes type and length into account during transfer and converts the storage type of the send field into that of the receive field prior to a numerical transfer if necessary, the data types of the send and receive fields must be compatible (see the table in the %SET description of the language-specific manuals
[2] - [6]). If the data types are not compatible, a storage type matching the memory contents and compatible with the receive field can be specified.
The following restriction applies with regard to the %SET command for programming languages that allow the definition of structures: structures can only be modified using a %SET command if the send and receive fields have the same structure. If one of the addresses was not described as a structure during programming, it can be assigned the required structure by means of type selection. In that case, however, the current memory contents must match the definition of the structure.

Examples

```
1. %DISPLAY V'10A'%F
```

The memory content as of address V’10A’ is interpreted as a signed binary value and output as a signed integer. Without type modification, the virtual address would have a hexadecimal storage type (%X) with a length of four bytes, i.e. output type DUMP.

```
2. %INSERT V'4710'%SX->
```

The memory content of address V’4710’ is evaluated for test point calculation in accordance with the %SX format.

```
3. %SET RECORD.10%PL5 INTO AMOUNT
```

A COBOL program contains a data item, RECORD, with a length of 45 bytes, which contains a sequence of packed numbers, each 5 bytes long. AMOUNT is a numeric unpacked data element. The first two numbers are skipped with the byte offset. As a result of type and length modification, the third packed number is unpacked from RECORD and transferred right-justified to AMOUNT.

```
4. %D V'134'.(INDEX * 4)%T(LINE)
```

The number of bytes from the contents of INDEX multiplied by 4 are added to virtual address V’134’ by byte offset. The memory content at the calculated address is edited in accordance with the type and length of the data definition for LINE and then output.

5. `%DISPLAY %1%F`

   Without type modification, AID would output the content of register 1 as a hexadecimal number. Type modification %F causes AID to edit the register content before output as a signed integer.

6. `%DISPLAY X'20AC'%UTF16`
   As a result of the type modification the output takes place in dump format, i.e. not only the hexadecimal code 20AC is output but also the interpretation as UTF16 code, in this case the Euro symbol.

7.2.4.4 Length modification

A length modification permits a deviation from the predefined length of a memory reference. AID then uses the specified length instead of the length stored in the length attribute. The value of a length modification must be between 1 and 65535.

If *type* is not specified, length modification implies a type modification into storage type %X.

A length modification must not violate the area limits of the modified memory object, i.e. the new length cannot exceed the end address.

```
length-modification - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                    ⎧Ln              ⎫
memref %[type]      ⎨L(memref)       ⎬
                    ⎩L=(expression)  ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

memref

May designate any memory location referenced in any manner:

virtual address, data name, keyword, C qualification or complex memory reference.

type

If type and length modification are to be effected, a storage type keyword must be entered (%X, %C, %P, %D, %F, %A, %UTF16) followed by L without another % character.
Example:
VAR1%L5 or VAR1%CL5

%Ln

A length modification beginning with %L implies a type modification into the default storage type %X.

n is a positive integer or hexadecimal number, where: $0 \leq n \leq 65535$
in accordance with the permissible value for a length modification.

%L(memref)

The length selector uses the length attribute of a different memory reference for length modification. The length selector is applied to data names, C qualifications and COM qualifications.

%L=(expression)

The length function causes AID to calculate the length. *expression* is described in section “Address, type and length selectors” on page 93. It is formed from integers, contents of memory references with type ’integer’ (%F or %A) and a length ≤ 8, the result of address selector, length selector and length function, and the arithmetic operators (+ - * /).
The operands involved and the result must be in the value range of a %FL8 field. If the *expression* of a length function contains only a memory reference, AID assumes the content (instead of the length) as the value for length modification.

The value range $-2^{63} \leq n < +2^{64}$ is supported. This enables data types %FL8 with values $-2^{63} \leq n < +2^{63}$ and %AL8 with values $0 \leq n < +2^{64}$ to be represented correctly. If the result does not comply with the value range, error message AID0470 is issued.

If a length selector is used for a vector but no index is specified, the length of the entire vector is selected. It is only through specification of an index that AID can access the length of an element of the vector.

Examples

```
1. %DISPLAY V'10A'%L=(VAR1)
```

VAR1 is of type ’integer’ and contains the value 23. 23 bytes in dump format are output as of address V’10A’.

```
2. %SET CVAR1%CL(CVAR) INTO CVAR
```

If it is assumed that CVAR1 and CVAR are two character variables in a Fortran program and that CVAR1 is longer than CVAR, the length modification makes it possible to transfer CVAR1 left-justified with the same length as CVAR.

```
3. %SET %L(CVAR) INTO %2G
```

The length of variable CVAR is transferred to AID register %2G.

```
4. %DISPLAY V'10A'%AL3->
```

The contents of three bytes as of address V’10A’ are interpreted as an address, and AID outputs four bytes in dump format (%XL4) as of the memory location thus referenced.

```
5. %D V'10A'%L=(INDEX*12-%L(NAME))
```

Here, the length is derived from multiplying the content of INDEX by 12 and subtracting the length of NAME.

```
6. %D V'4700'%L=(%L(C=CS1)-%L(INDAT))
```

The length is calculated from the length of CSECT CS1 minus the length of INDAT.

7.2.4.5 Arithmetic expression

In a byte offset, length function or index, an arithmetic expression can be specified for calculation of the requisite value by AID. An expression may thus be used wherever an integer value is required.
AID processes the arithmetic operators according to the mathematical rules for the resolution of an arithmetic expression. The order of processing can be changed by inserting parentheses. It is advisable to insert a blank before and after a minus sign "-" so that no misinterpretation can occur in the %AID SYMCHARS[=STD] setting.

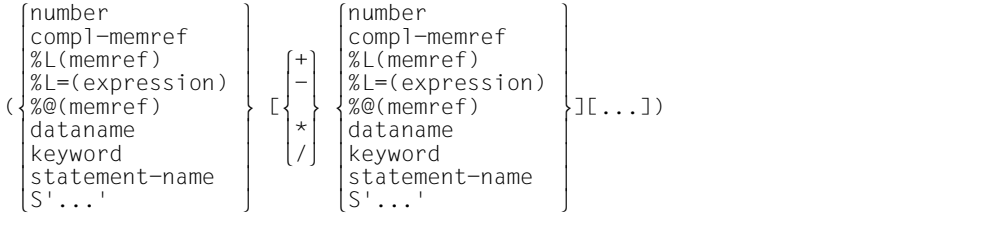
The following applies for each processing step of *expression*:

$-2^{63} \leq$ intermediate result $\leq 2^{63}-1$.

The following applies for a byte offset: $-2^{31} \leq$ end result $\leq 2^{31}-1$

The following applies for the length function: $0 \leq$ end result ≤ 65535

For the index, the limits defined in the source program apply. Moreover, only the operands *number* and *dataname* may be used (see section “Data names” on page 74).



number

Integer or hexadecimal number between -2^{63} and $2^{63}-1$.

compl-memref

May designate any memory location referenced in any manner. Its content must be an integer, i.e. of type %F or %A with a length ≤ 8.

The content of a memory reference can thus be used for a byte offset, for length modification, or as an index/subscript.

%L(memref)

The length selector is used to access the length attribute of a memory reference. The result is an integer. Length selectors are only applied to data names, C qualifications and COM qualifications, since the length of other memory references such as keywords is known anyhow.

%L=(expression)

AID calculates an integer value via the length function.
expression corresponds to the rules described here.

%@(memref)

The address selector is used to access the address attribute of a memory reference. The result is an address constant (%AL4).

dataname

Must be defined in the source program with type 'integer' or 'address' and a length ≤ 8.
The contents of *dataname* are used for calculating the arithmetic expression.

keyword

The contents of *keyword* are used for calculating the arithmetic expression. The following keywords may be specified (see chapter "Keywords" on page 109):

```
%n                  General register, 0 ≤ n ≤ 15
%nG                 AID general register, 0 ≤ n ≤ 15
%PC                 Program counter
%•[subcmdname]      Execution counter; the abbreviation %• designates
                    the execution counter of the currently active
                    subcommand
```

statement-name

As statement names are address constants, they can be used in expressions.

S'...'

Source references are likewise address constants and can thus be used in expressions.

Examples

```
1. %D %L=(%1+5)
```

The length derived from the content of register %1 plus 5 is output.

```
2. %D V'0'.(V'100'%AL2 + %L(C=CSECT))
```

As of address V'0', a byte offset is effected with the length of the expression specified in parentheses. First the contents of the two bytes with addresses V'100' and V'101' are interpreted as positive integers with a length of 2. The length of CSECT is then added to than. AID outputs 4 bytes in dump format as of the memory location thus calculated.

```
3. %S %L=((V'0'%AL4 + V'4'%AL1) * NUM1) INTO %2G
```

The value calculated by the length function is transferred to AID register %2G. V'0' contains X'00000005', V'4' contains X'FFF5003A' and NUM1 contains the value. After the type and length modifications, this results in (5 + 255) * 3 = 780. The value 780 is thus transferred to %2G.

7.2.4.6 Address, type and length selectors

The selectors support access to the attributes of a memory reference.

```
selectors - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

%@(memref)
%T([area-qua•]dataname)
%L(memref)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

%@(memref)

The address selector accesses the address attribute of a memory reference. The result is an address constant (%AL4). An address selector may be employed before a pointer operator to access a memory location, or as an unsigned binary number in an expression. %DISPLAY can be used to output the result of an address selection.
The address selector is applied to data names, C qualifications and COM qualifications. The addresses of keywords outside the user area can be output via %DISPLAY but cannot be used as a memory reference with a subsequent pointer operator.

%T([area-qua•]dataname)

The type selector accesses the type attribute and length attribute of a memory reference. The selected data type can only be used for type modification. *dataname* may be qualified.

%L(memref)

The length selector accesses the length attribute of a memory reference. The result is a positive integer. Length selectors may be employed for length modification or in expressions. %DISPLAY can be used to output the result of a length selection. The length selector is applied to data names, C qualifications and COM qualifications only, since the length of other memory references such as keywords and virtual addresses is known anyhow.

Examples

```
1. %D %@(VAR)
   %D %@(VAR)->.8
   %S V'2E'.(%@(VAR))%CL2 INTO X
   %D V'A1A'%XL=(2+@(VAR))
```

Use of the address selector:

The first %DISPLAY outputs an address.

The second %DISPLAY switches to machine code level (address selection and pointer operator) so that the byte offset operation is not impeded by the area limits of VAR.
%SET uses the address of VAR as a value for a byte offset.
The last %DISPLAY uses the value of the address of VAR in order to calculate the length.

```
2. %D V'100'%T(VAR)
   %S V'100'%T(INT) INTO NUM1
```

Use of the type selector:

%DISPLAY outputs the content of a virtual address with the type and length of VAR (redefinition). %SET interprets the content of a virtual address with the type and length of integer variable INT so that its value is retained during transfer to the numeric variable NUM1.

```
3. %D %L(VAR)
   %S %L(VAR) INTO NUM1
   %D V'A1A'.(2+%L(VAR))
   %D V'A1A'%L=(%L(VAR)*5)
```

Use of the length selector:

The first %DISPLAY outputs the length of VAR.

%SET transfers the value of the length of VAR to the numeric variable NUM1.
The second %DISPLAY uses the length of VAR as a value in a byte offset expression.
The last %DISPLAY uses the value of the length of VAR in a length modification.

7.2.4.7 Special features of the interaction of various components

If a complex memory reference begins with an address constant (for example

with a source reference or a label), the pointer operator must be written next. Names of labels must always be set in L'...'.

Without the pointer operator, address constants may be positioned anywhere within the *compl-memref* where hexadecimal numbers can be written.

After a byte offset of pointer operation (exceptions apply to C++/C and PL/I, see section "Byte offset "•"" on page 81 and section "Indirect addressing "->" / "*"" on page 83), the implicit storage type and implicit length of the start address are lost. If no other storage type and length are explicitly defined, storage type %X with a length of 4 applies at the calculated location, unless the complex memory reference is used as the receiver in the %MOVE command. In this case the area that may be overwritten with %MOVE extends from the start address of *compl-memref* to the end of the memory occupied by the program.

The memory area assigned for an operand in a complex memory reference must not be exceeded by a byte offset or a length modification, otherwise AID issues an error message. However, if it is intended to use the start address of
a memory object without having to pay attention to the area boundaries, address selection should be used in conjunction with the pointer operator (%@(...)->). This takes the user away from the symbolic level, which at the same time means that the type attribute and length attribute of the referenced object can then only be accessed via the corresponding selectors.

Some compilers, such as C++/C and PL/I, generate a prologue for each program or subprogram during compilation. *function* (C++/C) or *entry* (PL/I) without a subsequent pointer operator designate the first executable statement of the corresponding function or procedure. However, if *function* or *entry* is followed by the pointer operator in order to move to a further position starting from the start of the function or procedure, it must be ensured that address calculation begins at the start address of the prologue.

Character encoding of a string

The interpretation of the string encoding can be changed using the conversion functions %C() and %UTF16(). The memory reference must be of the type “string”, i.e. of the type %C or %UTF16.

The conversion functions have an effect only when %C() is applied to the type %UTF16 or %UTF16() to the type %C. The memory locations remain unchanged here. AID continues to work implicitly with the converted memory location.

The CCSN for type %C from the %AID EBCDIC setting is used. The settings that are currently applicable can be displayed with the %SHOW %AID command.

Example:

The byte X'BB' is contained at memory location V'00'.

1. `%AID EBCDIC=EDF03IRV`
   The byte X'BB' consequently defines the character '[' in CCSN EDF03IRV.
   %UTF16(V'00' %CL1) results in the hexadecimal value X'005B'.
2. `%AID EBCDIC=EDF03DRV`
   The byte X'BB' X'BB' defines the character C'Ä' in CCSN EDF03DRV.
   %UTF16(V'00' %CL1) results in the hexadecimal value X'00C4'.

8 Medium-a-quantity operand

The *medium-a-quantity* operand defines the output medium to be used by AID and whether additional information apart from the contents of the specified memory area (data) is to be output.
This operand may be output more than once, separated by commas; for example, T=MIN,P=MAX can be used to output minimum information at the terminal and maximum information at SYSLST.

The *medium-a-quantity* operand may be specified in the following commands:

%DISPLAY
%HELP
%SDUMP
%OUT

The *medium-a-quantity* operand of the %OUT command also affects:

%DISASSEMBLE
%TRACE

The %OUT command can be used to predefine *medium-a-quantity* for %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP and %TRACE. This presetting applies throughout the debugging session until a new definition is made or termination is initiated with /EXIT-JOB.

A local *medium-a-quantity* specification in %DISPLAY, %SDUMP and %HELP applies for the current command only; afterwards the *medium-a-quantity* value of the %OUT command or the default value T=MAX takes effect again. If no *medium-a-quantity* is specified in %DISPLAY, %SDUMP or %HELP the *medium-a-quantity* value of the %OUT command applies. If %OUT contains no *medium-a-quantity* declaration either, the default value T=MAX is assumed.

AID takes the coded character set name (CCSN) asigned into account for all output media whenever UTF16/UTFE characters are to be output to the output medium. However, AID only supports UTFE or 1-byte EBCDIC codes.

```
medium-a-quantity-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - -

{ T  }     {MAX}
{ H  } = {     }
{ Fn }     {MIN}
{ P  }

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

T Terminal output via SYSOUT.

H Hardcopy output (includes terminal output and cannot be specified together with *T*)

Fn File output. Fn designates the link name for the output file.

n is a number with a value $0 \leq n \leq 7$.

There are three ways of creating the associated file:

1. %OUTFILE command with link name and file name.
2. /FILE command for Fn.
3. For a link name with no file name assignment, AID issues a FILE macro with the file name AID.OUTFILE.Fn in accordance with the link name Fn. The file is created with `FCBTYPE=SAM, OPEN=EXTEND, RECFORM=V`.

When using the link name F6, remember that F6 is the default link name for REP files.

P Output to SYSLST.

MAX

The data is output with a maximum of additional information.

This includes information on the AID work area and on the interrupt point, but also information on the data to be output.

For %HELP the {MIN | MAX} specification has no effect, but one of the two entries is mandatory for syntactical reasons.
If output with the operand value MAX is effected for an output medium for the first time, or if the line contents have changed as compared with a previous output, up to three lines appear before the actual output:

– task line: provides information on the current AID work area and contains the task identifier (TID) and the task sequence number (TSN) or the link name of the dump file.

– header line: contains information on the interrupt point, i.e. the address at which the program stopped at the time of output.

– target line: contains information on the address to be output, i.e. the CSECT/COMMON containing the address and the displacement to the beginning of CSECT/COMMON.

The following is output for each data area:

– data name and, in the case of vectors, the index boundary list
– content (plus the associated index in the case of vectors);

  in the case of identical lines the text "repeated lines: n" is output.

– virtual address of the first byte of each data line in the case of output on machine code level
– displacement of the first byte of each data line to the beginning of a CSECT if the address can be assigned to a CSECT; otherwise the address of the first byte of each data line relative to the beginning of the data area

MIN

No additional information is output with the data.

Examples

```
%OUT %DA T=MIN
%C1 %IO <%DA FROM %PC-->;%STOP>
%R
00000BB6  L     R1,A8(R0,R11)
00000BBA  L     R15,98(R0,R11)
00000BBE  BALR  R14,R15
00000BC0  DC    X'0001'  INVALID OPCODE
00000BC2  CLI   396(R12),X'F0'
00000BC6  L     R13,B4(R0,R2)
00000BCA  BC    B'1100',E0(R0,R13)
00000BCE  L     R15,A4(R0,R11)
00000BD2  BAL   R14,4(R0,R15)
00000BD6  DC    X'0000'  INVALID OPCODE
STOPPED AT SRC_REF: 540PE, SOURCE: TEST, PROC: TEST
```

The data retrieved by the %DISASSEMBLE command is output without any additional information. The virtual addresses of the respective commands are prefixed in the form of 8-digit hexadecimal numbers.

```
%OUT %DA T=MAX
%C1 %IO <%DA FROM %PC->;%STOP>
%R
TEST+C22          L     R1,A8(R0,R11)          58 10 B0A8
TEST+C26          L     R15,9C(R0,R11)         58 F0 B09C
TEST+C2A          BALR  R14,R15                05 EF
TEST+C2C          CLI   396(R12),X'F2'         95 F2 C396
TEST+C30          L     R13,B4(R0,R2)          58 D0 20B4
TEST+C34          BC    B'1100',14A(R0,R13)    47 C0 D14A
TEST+C38          L     R15,A4(R0,R11)         58 F0 B0A4
TEST+C3C          BAL   R14,4(R0,R15)          45 E0 F004
TEST+C40          DC    X'0000'  INVALID OPCODE 00 00
TEST+C42          CLI   396(R12),X'F0'         95 F0 C396
STOPPED AT SRC_REF: 58REA, SOURCE: TEST, PROC: TEST
```

The data retrieved by the %DISASSEMBLE command is output with additional information. The command addresses are stated in the form of relative addresses, i.e. as program names plus displacement to the beginning of the program. The disassembled commands are followed by the memory contents in hexadecimal format.

```
%OUT %D T=MIN
%D ABC-TAB
01          ABC-TAB
 02         CHARS( 1: 26)
            |A| |B| |C| |D| |E| |F| |G| |H| |I| |J| |K| |L| |M|
            |N| |O| |P| |Q| |R| |S| |T| |U| |V| |W| |X| |Y| |Z|
```

The table ABC-TAB from a COBOL program is output via %DISPLAY without any additional information. The level numbers and the contents of the table items are output.

```
%D ABC-TAB T=MAX
*** TID: 000000D1 *** TSN: 8438 ************************************************
SRC_REF:     58ADD   SOURCE: MOBS      PROC: MOBS  ******************************
01          ABC-TAB
 02         CHARS( 1: 26)
            (  1) |A|  (  2) |B|  (  3) |C|  (  4) |D|  (  5) |E|  (  6) |F|
            (  7) |G|  (  8) |H|  (  9) |I|  ( 10) |J|  ( 11) |K|  ( 12) |L|
            ( 13) |M|  ( 14) |N|  ( 15) |O|  ( 16) |P|  ( 17) |Q|  ( 18) |R|
            ( 19) |S|  ( 20) |T|  ( 21) |U|  ( 22) |V|  ( 23) |W|  ( 24) |X|
            ( 25) |Y|  ( 26) |Z|
```

The table ABC-TAB from a COBOL program is output via %DISPLAY with additional information. The actual output is preceded by a task line and a header line. The level numbers, the contents of the table items and the associated indexes are output.

9 AID literals

An AID literal can be specified as an operand in the AID commands %DISPLAY, %FIND, %MOVE and %SET.

9.1 Alphanumeric literals

9.1.1 Character literal

9.1.1.1 Input formats

{C'x...x' | 'x...x'C | 'x...x'| U'x...x'}

Maximum length: 80 characters.
Character set for *x*: any character which can be entered at the terminal.

If the coded character set for the input medium is not UTFE, a UTFE character string can be specified with the aid of the U literal.

Lowercase letters can be entered as such only if %AID LOW[=ON] has been specified. Normally (default setting) lowercase letters are converted into uppercase (see %AID); lowercase letters can then only be specified in the form of hexadecimal literals.

Apostrophes which are to be included in the literal must be duplicated (").

%DISPLAY
The literal is output. It may be converted using a type modification.

%FIND
x may also be the wildcard symbol '%', which stands for an arbitrary character and is always reported as a hit by %FIND.
When %C() and %UTF16() are applied to the search literal, '%' is no longer supported as a wildcard symbol.

%MOVE
The literal is transferred to the receive field left-justified and in the length of the literal. If the literal is longer than the receive field, the transfer is rejected with an appropriate message.

%SET
If the literal consists of numerals only, has a length ≤ 18 and is to be transferred to a numeric field, it is converted like a numeric literal and transferred retaining its correct value.

If its content is not purely numeric or its length > 18, the literal can be transferred in alphanumeric form to a character field (%C), or in binary form to a field with type modification %X, where it is stored left-justified. If the receive field is longer than the literal, the literal is padded on the right, with blanks (C'␣' ≙ X'40') in the case of alphanumeric transfer or with X'00' in the case of binary transfer. If the literal is longer than the receive field, it is truncated on the right and a warning issued.

9.1.1.2 Character encoding

Following the introduction of Unicode, AID supports the **c**oded **c**haracter **s**et **n**ame (CCSN) assigned to input and output media.
A CCSN can be assigned to a file using the CODED-CHARACTER-SET operand in the MODIFY-FILE-ATTRIBUTES command.

In the case of the input medium TERMINAL, AID uses the CCSN which was set using the BS2000 command MODIFY-TERMINAL-OPTIONS. Here AID knows none of the settings which were made directly in the terminal emulation but were not made known via the MODIFY-TERMINAL-OPTIONS command.

If no CCSNs were defined for input and output media, AID uses the CCSN of the user ID's Join entry as the default CCSN. This setting can be modified using the %AID EBCDIC= … command.

For input and output media, AID supports only those CCSNs which are also supported by XHCS and, except for UTFE, represent a 1-byte EBCDIC code. A prerequisite for this is the subsystem XHCS-SYS version ≥V02.0.

The AID command %SHOW %CCSN enables the CCSNs currently supported by XHCS-SYS to be displayed.

9.1.1.3 Conversion functions %C() and %UTF16()

These functions can be used to modify the type of character encoding of a character literal.

%UTF16() converts the literal into a UTF16 string.
%C() converts a UTF16 literal into a 1-byte EBCDIC encoding which was defined by the %AID EBCDIC command.

If the literal is available in 1-byte EBCDIC encoding, %C() has no effect.

During conversion a character is replaced by the substitute character '.' if it is not available in UTF16 or the 1-byte EBCDIC character set. In this case AID issues a message.

9.1.2 Hexadecimal literal

{X'f...f' | 'f...f'X}

Maximum length: 80 hexadecimal digits (equals 40 bytes).
A literal with an odd number of digits is complemented with X'0' on the right.
Character set for *f*: any character in the range 0-9 and A-F.

The type modification %UTF16 is permissible for a hexadecimal literal. As a result of this type modification, the literal is treated like a character literal (see section "Character literal" on page 101).

%DISPLAY
The literal is output. It may be converted using a type modification.

%FIND
f may also be the wildcard symbol '%', which stands for an arbitrary character and is always reported as a hit by %FIND.

%MOVE
The literal is transferred to the receive field left-justified and in the length of the literal. If the literal is longer than the receive field, the transfer is rejected with an appropriate message.

%SET
The literal is transferred left-justified. If the receive field is longer than the literal, padding with X'00' occurs on the right. If the literal is longer than the receive field, it is truncated on the right.
This literal can be used for transfer to a receive field with any data type definition.

9.1.3 Binary literal

{B'b...b' | 'b...b'B}

Maximum length: 80 binary digits (equals 10 bytes).

Padding with binary zeros (B'0') occurs on the right up to byte length (8 binary digits).
Character set for *b*: characters 0 and 1.

%DISPLAY
The literal is output. It may be converted using a type modification.

%FIND
B'b...b' cannot be specified.

%MOVE
The literal is transferred to the receive field left-justified and in the length of the literal. If the literal is longer than the receive field, the transfer is rejected with an appropriate message.

%SET
The literal is transferred left-justified. If the receive field is longer than the literal, padding with binary zeros occurs on the right. If the literal is longer than the receive field, it is truncated on the right.
This literal can be transferred to a receive field with any data type definition.

9.2 Numeric literals

9.2.1 Integer

[{±}]n

Maximum length:20 digits
Value range: $-10^{21} \leq n \leq +10^{21}$

For the user the internal representation of an integer is undefinded, i.e. if the user references to the internal representation within a AID command, the result is also undefined.
Example: `%D 12345 %X / %M 123456789 INTO V'xxxx'`

For compatibility reasons, the internal representation in the range $2^{31} \leq n \leq +2^{31}-1$ is like %FL4.

%DISPLAY
The literal is output. It may be converted using a type modification.

%FIND
An integer cannot be specified.

%MOVE
The integer is edited as a one-word hexadecimal value (4 bytes) and stored left-justified in the receive field. If the receive field is too short, the transfer is rejected with an appropriate message.

%SET
The integer can be transferred to any numeric receive field; it is adapted to the type of the receive field if necessary and transferred so that its value is retained.

9.2.2 Hexadecimal number

#'x...x'

Maximum length: 16 hexadecimal digits (corresponds to storage type %FL8, signed integer).

Value range:
With a max. length of 8 hexadecimal digits: $-2^{31} \leq$ #'x...x' $\leq +2^{31}-1$ (%FL4)
With a max. length of at least 9 hexadecimal digits: $-2^{63} \leq$ #'x...x' $\leq +2^{63}-1$ (%FL8)

Character set for *x*:

Negative hexadecimal number:
32-bit numbers:
have precisely 8 hexadecimal digits, the first bit in the leftmost digit defining the sign. In order to define a negative value, this digit must come from the range X'8' , X'9', ..., X'F'.

64-bit numbers:
have precisely 16 hexadecimal digits; as with a 32-bit hexadecimal number, the leftmost digit must come from the range X'8' , X'9', ..., X'F'.

Example:
#'FFFFFFFF' defines a 32-bit hexadecimal number with the value -1;
#'0FFFFFFFF' defines a 64-bit hexadecimal number with the value +2**32-1;

Consequently the behavior of older AID versions in the case of 32-bit hexadecimal numbers is retained.

%DISPLAY
The literal is output. It may be converted using a type modification.

%FIND
A hexadecimal number cannot be specified.

%MOVE
The hexadecimal number is edited in word length (4 bytes) and stored in the receive field left-justified. If the receive field is too short, the transfer is rejected with an appropriate message.

%SET
The hexadecimal number can be transferred to any numeric receive field; it is adapted to the type of the receive field if necessary and transferred so that its value is retained.

9.2.3 Decimal number

[{±}]n.m

Maximum length: 18 digits, decimal point and sign

The leftmost digit may be preceded by a sign. A decimal point may be located at any position within the digit sequence. If it is to occupy the leftmost position, it must be preceded by a zero.

%DISPLAY
The literal is output.

%FIND/%MOVE
A decimal number cannot be specified.

%SET
The decimal number can be transferred to any numeric receive field; it is adapted to the type of the receive field if necessary and transferred so that its value is retained.

9.2.4 Floating-point number

[{±}]mantissaE[{±}]exponent

The floating-point number is set up internally with double precision (8 bytes). If *mantissa* and/or *exponent* are unsigned, they are assumed to be positive. No blanks are allowed within the floating-point number.

mantissa

Maximum length: 16 significant digits, decimal point and sign.

mantissa must contain a decimal point, which may be located at any position within. If it is to occupy the leftmost position, it must be preceded by a zero.

exponent

Maximum length: 2 digits and sign.

Value range: $-75 \leq exponent \leq 76$.

%DISPLAY
The literal is output.

%FIND/%MOVE
A floating-point number cannot be specified.

%SET
The floating-point number can be transferred to any numeric receive field; it is adapted to the type of the receive field if necessary and transferred so that its value is retained.

10 Keywords

Keywords are predefined declarations for AID and start with the % character. They stand for storage types, registers, program counter, memory areas, system information, execution counters, logical values, feed control, address switchover, output of the current call hierarchy, instruction types, and events.
In a complex memory reference, keywords for memory areas, program registers, program counter, AID registers and execution counters can be used. The implicit storage types and lengths are stated below in the respective sections.

10.1 General storage types

The keywords for storage types can be used to change the interpretation of a memory location or of a literal (see section “Type modification” on page 86). This may be necessary or expedient, for instance, for the %SET command (when the storage types of *sender* and *receiver* are incompatible) or for %DISPLAY (if a memory location or a literal is to be output after conversion; each storage type is implicitly assigned an output type which defines how the memory contents are to be output) or when a memory location is to be included in the address calculation.

The optional length specification *L-mod* also permits a length modification (see section “Length modification” on page 89). This is not permitted in the case of literals. No blank is allowed between the type and length entries. The length entry may assume any form of length modification.

```
%X[L-mod]   Hexadecimal, length 1 ≤ n ≤ 65535
            The default storage type for a virtual address is %XL4
            Output type: dump (hexadecimal and character)

%C[L-mod]   Character, length 1 ≤ n ≤ 65535
            Output type: character

%UTF16[L-mod]
            Unicode character, length 2 ≤ n ≤ 65 534,
            The length must be a multiple of 2.
            Output type: dump (hexadecimal and character)

%P[L-mod]   Packed, length 1 ≤ n ≤ 9, can only contain the sign
            (last half-byte) and digits.
            Output type: numeric (signed integer)
            Not permitted for literals.
```

```
%D[L-mod]   Floating-point, length n = 4, 8 or 16 bytes
            Output type: numeric (floating-point)
            Not permitted for literals.

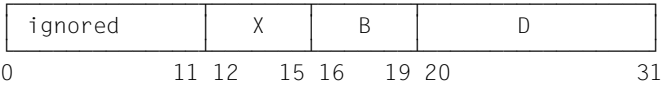
%F[L-mod]   Binary, signed integer, length n = 1..8 bytes
%H          Corresponds to %FL2
            Output type: numeric (signed integer)
            %H is not permitted for literals.

%A[L-mod]   Address, length n = 1..8 bytes
%Y          Corresponds to %AL2
            Output type: numeric (unsigned integer)
            Not permitted for literals.
```

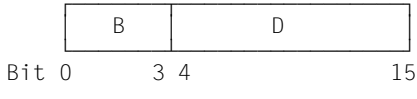
10.2 Storage types for interpreting machine instructions

These storage types are used to calculate an address expressed in the memory operands of machine instructions in the form of base register and displacement or index register, base register and displacement. Only a subsequent pointer operator (->) actually initiates address calculation. Without such a pointer operator, %SX equals %XL4 and %S equals %XL2. Examples are given in section “Indirect addressing "->" / "*"” on page 83.

```
%SX     SX address, length 4 bytes, index-base-displacement (X-B-D),
        corresponds to a machine instruction in SX format
        The index register number X is only evaluated if it is  0.
        Output type: HEX
```




```
%S      S address, length 2 bytes, base-displacement (B-D)
        The base register number B is only evaluated if it is | 0.
        Output type: HEX
```



10.3 Program registers and program counter

The program registers (general and floating-point registers) and the program counter are addressed by AID via keywords. Their contents can be output (%DISPLAY), modified (%MOVE, %SET) or used for addressing. The program registers are located in the privileged area and cannot be referenced by their virtual addresses. The floating-point registers occupy common memory space:
%0Q overwrites %0D and %2D; %4Q overwrites %4D and %6D.

AID handles the contents of general registers in subcommand conditions and in arithmetic expressions as signed numeric values in accordance with the type %FL4.
Before a pointer operator, however, AID assumes the type %AL4 and the register contents can be used as an address without type modification. A register is output with output type 'dump'. If numeric values are to be signed, a type modification with %F must be effected prior to output.

In ASSEMBH the program registers have the symbolic name _Rn, which can only be used for the symbolic debugging of Assembler programs however.

```
%PC   Program counter, type %AL4
%n    General register 0 ≤ n ≤ 15, storage type %FL4, output as hex. number
%nE   Floating-point register with single precision n = {0,2,4,6}, type %DL4
%nD   Floating-point register with double precision n = {0,2,4,6}, type %DL8
%nQ   Floating-point register with quadruple precision n = {0,4}, type %DL16

%MR   All 16 general registers edited in tabular form
%FR   All 4 floating-point registers with double precision
      edited in tabular form

%nAR  Access register 0 ≤ n ≤ 15, storage type %FL4, output as
      hexadecimal number (ESA support)
%AR   All 16 access registers edited in tabular form (ESA support)
```

The keyword %PC and the registers may be indexed if required. This is only necessary in the case of a program which has defined contingency processes or which was interrupted by AID during STXIT processing: if information other than on the interrupted contingency or STXIT process is desired (e.g. on the base process) the index of the appropriate process level must be specified. This index may be queried with %DISPLAY %PCBLST (see Executive Macros [11]).

The index is specified in the format: keyword(index).

10.4 AID registers

The AID registers are located in the memory area reserved for AID and can thus be referenced from any AID work area without a new work area declaration. The type and length of AID registers are the same as for program registers.
There are no keywords for addressing all AID registers collectively.

```
%nG     AID general register 0 ≤ n ≤ 15, storage type %FL4, output as hex. no.
%nGD    AID floating-point register with double precision n = {0,2,4,6},
        type %DL8
```

10.5 Memory classes

The keywords listed below are used to reference memory classes. They can be specified in %CONTROLn, %DISASSEMBLE, %DISPLAY, %FIND and %TRACE. In a complex memory reference, keywords for memory classes can be used with all attributes, i.e. name, address, content, length and type.

In class 5 memory the program occupies privileged and non-privileged areas. Both areas can be accessed with the keyword %CLASS5, but the privileged area can only be accessed with a higher test privilege.

The keywords %CLASS5 and %CLASS5BELOW designate the same address space on XS computers or in XS dumps. The same is true of %CLASS6 and %CLASS6BELOW.

```
%CLASS5             Class 5 memory, type %X
%CLASS6             Class 6 memory, type %X
```

In addition, there are the following keywords (all of type %X) for debugging a program under a BS2000 version ≥V9 or for processing a dump generated under such a BS2000 version:

```
%CLASS5BELOW        Class 5 memory below the 16-Mb boundary (≙%CLASS5)
%CLASS5ABOVE        Class 5 memory above the 16-Mb boundary
%CLASS6BELOW        Class 6 memory below the 16-Mb boundary (≙%CLASS6)
%CLASS6ABOVE        Class 6 memory above the 16-Mb boundary
```

10.6 System information

The keywords for system information support the output of information on a particular task via %DISPLAY. If a given keyword returns more than one value, AID edits the values and outputs an appropriate table.

```
%CC                    Condition code
%PCB                   Process control block
%PCBLST                List of all process control blocks
%LINK                  Name of the segment last loaded, which was determined
                       with %ON %LPOV
%PM                    Program mask
%AUD1                  Hardware audit table, starting with the oldest
                       entry (only if created at system generation)
%AMODE                 System information field for addressing mode
                       (can be modified with %MODE24 or %MODE31 only)
%ASC                   ASC mode on ESA systems
                       (with regard to AR mode: X'00' = off; X'01' = on)
%DS[(ALET/SPID-qua)]   Information about SPIDs and/or ALETs of the active
                       data spaces on ESA systems

%LOC(memref)           Machine-oriented localization information for
                       an address in the executable part
%HLLOC(memref)         Symbolic localization information for an address
                       in the executable part

%SORTEDMAP             List of all CSECTs and COMMONs of the user program
                       (sorted by names and addresses)
```

```
        ⎧ CTX=context [•L=loadunit]    ⎫
        ⎪ L=loadunit                   ⎪
%MAP [(⎨                               ⎬)]
        ⎪ SCOPE = ⎧USER⎫               ⎪
        ⎩         ⎩ALL ⎭               ⎭
```

`{CTX=context | L=loadunit}#1`
When a path is specified, all CSECTs/COMMONs of the specified context or load unit are listed.

`SCOPE=USER` CSECTs/COMMONs of the default contexts CTXPHASE or LOCAL#DEFAULT and of the contexts formed by the BIND macro with operand LNKCTX[@] are listed.

`SCOPE=ALL` In addition to the CSECTs/COMMONs of the user-defined contexts, the map of all contexts is output to which the program has connected, e.g. DSSM subsystems or user pool contexts.

All BLS names (context, load unit, CSECT and COMMON) are output unabbreviated. Within the contexts and load units, the output list is sorted according to CSECT name.

Examples

1. `/%D %HLLOC(PROG=UPRONUM.S'22DIS'->)`

   AID output

```
 V'000083EC' = CONTEXT  : LOCAL#DEFAULT
               SMOD     :  UPRONUM
               PROC     :  UPRONUM
               PARAGRAPH:  UNPCK
               SRC-REF  :     21
               LABEL    :  UNPCK
```

2. `/%D %LOC(PROG=UPRONUM.S'22DIS'->)`

   AID output

```
 V'000083EC' = CONTEXT  :  LOCAL#DEFAULT
               LMOD     :  %UNIT
               SMOD     :  UPRONUM
               OMOD     :  UPRONUM
               CSECT    :  UPRONUM (00008018) + 000003D4
```

3. `/%D %MAP`

   AID output

```
*** TID: 00230056 *** TSN: OFZB ***********************************************
CURRENT PC: 00002000    CSECT: LLMTEST2  ***************************************
**CSECT-LISTING(MAP) OF CONTEXT : LOCAL#DEFAULT
**MAP OF LOAD UNIT             : %UNIT
  CSECT-NAME                      START     SIZE    VER/DATE_OF_MOD
  ASSTEST                         00000150  000830   ............
  ASS2                            00000980  000008   ............
  COMM1                           00001000  00012C   %COMMON.....
  LLMTEST1                        00000000  000150   ............
**CSECT-LISTING(MAP) OF CONTEXT : CTX2
**MAP OF LOAD UNIT             : LLMTEST2
  CSECT-NAME                      START     SIZE    VER/DATE_OF_MOD
  ASSTEST                         00002078  000650   ............
  ASS2                            000026C8  000008   ............
  COMM1                           FFFFFFFF  000000   %COMMON.....
  LLMTEST2                        00002000  000078   ............
```

10.7 Execution counter

An execution counter is set up for each subcommand. It counts the number of times the subcommand is executed. The subcommand's own execution counter may be addressed with %• within the subcommand. If the subcommand is assigned a name, the execution counter receives the same name and can then be referenced with %•subcmdname outside the subcommand. Execution of a subcommand may be made dependent on the status of the execution counter by querying %•subcmdname in a condition (see chapter "Subcommand" on page 49).

A numeric value may be assigned to the counter via %SET. The content of an execution counter can be read via %DISPLAY. The counter is incremented every time the associated subcommand is encountered in the program sequence. Execution counters can be used wherever a numeric value is permissible.

```
%•[subcmdname]   Variable of type %FL4
                 subcmdname is the name of the associated subcommand.
                 The abbreviation %• designates the execution counter of
                 the currently active subcommand.
```

10.8 Logical values

The two keywords listed below can be used to assign values to logical variables from Fortran programs via %SET.

```
%TRUE
%FALSE
```

10.9 Feed control

The two keywords for feed control are only effective for the output medium SYSLST and can only be specified in %DISPLAY.

```
%NP              Beginning of a new page
%NL[(n)]         Output of n blank lines, 1 ≤ n ≤ 255
                 The default value for n is 1.
```

10.10 Address switchover

XS programming uses 31-bit addresses instead of the customary 24-bit addresses. The two keywords listed below serve to change the addressing mode for the test object or the address interpretation in indirect addressing.

`%MOVE %MODE31 INTO %AMODE` changes the addressing mode

`%AINT %MODE24` changes the AID address interpretation in indirect addressing

```
%MODE24  24-bit addressing
%MODE31  31-bit addressing
```

10.11 Current call hierarchy

In %SDUMP the keyword %NEST causes the current call hierarchy to be output.

```
%NEST    Output of the current call hierarchy
```

10.12 Criterion for %CONTROLn and %TRACE

The keywords listed below are used to group programming language commands or statements according to their type. These keywords can be specified as a monitoring criterion (*criterion* operand) in the %CONTROLn and %TRACE commands.

In the case of %CONTROLn the associated subcommand is processed when a command or statement of the group to be monitored is about to be executed.
In the case of %TRACE a log line is output when a command or statement of the group to be monitored is executed. In symbolic debugging, output occurs before statement execution; in debugging on machine code level, logging occurs after command execution.

The default value is the symbolic *criterion* %STMT. The consequence is that, in the case of a %CONTROLn or %TRACE with an area specification on machine code level, specification of a keyword for *criterion* is mandatory unless a monitoring criterion from a previous %CONTROLn or %TRACE is still valid.

criterion	Command group or statement group
Debugging on machine code level:	
%INSTR	All machine instructions being executed
%B	Branch instructions (i.e. the machine instructions BAL, BALR, BAS, BASSM, BASR, BC, BCR, BCT, BCTR, BSM, BXH and BXLE)
%BAL	Subprogram calls (using the machine instructions BAL, BALR, BAS, BASSM and BASR)
Debugging on the symbolic level:	
%STMT	All statements being executed
%ASSGN	Assignment statements
%CALL	Subroutine calls (CALL statements)
%COND	IF(...) THEN, ELSE IF(...) THEN, ELSE and IF(...) statements
%DB	Statement for calling a database
%EXCEPTION	Conditional statement branches
%GOTO	GOTO statements
%IO	Input/output statements
%LAB	Statement after label
%PROC	STOP, END, RETURN, SUBROUTINE and FUNCTION statements
%SORT	MERGE and SORT statements

10.13 Event for %ON

The keywords listed below stand for write monitoring, program errors, program termination, supervisor calls and other events during program execution. They can be specified in the %ON command (*event* operand). The *event* operand defines the occurrence upon which the program is to be interrupted so that the associated subcommand can be executed.
If several %ON commands with differing *event* declarations are active and satisfied at the same time, AID executes the related subcommands in the sequence in which the respective keywords are listed in the table below. If various %TERM events occur, the associated subcommands are processed according to the FIFO principle.

Write monitoring cannot be active for a number of different areas at the same time. A new %ON %WRITE(...) command overwrites one entered earlier (see section “%ON %WRITE with %INSERT, %CONTROLn and %TRACE” on page 121).

Further information on selecting the suitable %TERM can be found in the Executive Macros [11].

event	Subcommand is processed:
%WRITE(memref)	after overwriting the memory area identified by *memref* (as of BS2000/OSD V1.0)
%ERRFLG(z)	after occurrence of an error with the specified event code and before program abortion
%INSTCHK	after occurrence of an addressing error, an invalid supervisor call (SVC), a non-decodable operation code, a paging error or a privileged operation and before program abortion
%ARTHCHK	after occurrence of a data error, a divide error, an exponent overflow or a mantissa equalling zero and before program abortion
%ABNORM	after occurrence of one of the errors covered by the above events, of a DMS error (as of BS2000 V10) or of a %ILLSTX
%ERRFLG	after occurrence of an error with any weight
%SVC(z) %SVC	before execution of the supervisor call with the specified number before execution of any supervisor call
%LPOV(name) %LPOV	after loading of the segment with the specified name after loading of any segment
%TERM(N[ORMAL]) %TERM(A[BNORMAL]) %TERM(D[UMP]) %TERM(ND|NODUMP) %TERM(P[RGR]) %TERM(S[TEP]) %TERM	before program termination with TERM MODE = NORMAL, TERM or TERMD before program termination with TERM MODE = ABNORMAL, TERMJ or TRMJD before program termination with TERM DUMP = Y, TERMD or TRMJD before program termination with TERM DUMP = NO, TERM or TERMJ before program termination with TERM UNIT = PRGR, TERM or TERMD before program termination with TERM UNIT = STEP, TERMJ or TRMJD before program termination with TERM, TERMD, TERMJ or TRMJD
%ANY	before program termination due to a program error or a TERM with any operand values or TERMJ, TERMD or TRMJD, or due to a DMS error (as of BS2000 V10) or a %ILLSTX
%ILLSTX	before occurrence of a STXIT call during processing of a preceding STXIT call (STXIT in STXIT)

z is an integer where: $1 \leq z \leq 255$. z may be specified as an unsigned decimal number of up to three digits or as a two-digit hexadecimal number (#ff').'

No check is made as to whether the specified event code or the SVC number is meaningful or permissible.

11 Special applications

11.1 %ON and STXIT

There are different possible ways of responding to events which occur during execution of a program:

– STXIT routines can be assigned in the program to individual events; these routines are executed in order to process the events when they occur (see Executive Macros [11]).

– Events can be assigned via the %ON command during debugging with AID. When one of these events occurs, the subcommand specified in the %ON command is processed.

Events for which STXIT routines have been assigned in the program cannot be processed by AID: AID has no knowledge of the occurrence of such events. Any subcommands specified in the %ON command for these events will therefore not be executed.

STXIT routines are assigned among other things by the compiler runtime systems, by ILCS, openUTM and the database systems, for example for the "program error" or "unrecoverable program error" events. These STXIT events correspond to the %ERRFLG(zzz), %ERRFLG, %INSTCK, %ARTHCHK and %ABNORM events in the %ON command. These events can only be processed with AID is assignment of the STXIT routines has been suppressed.

For FOR1 programs without standard linkage, the assignment of STXIT routines is suppressed by specifying the option RUNOPT STXIT=NO. This is not possible for Fortran90 programs, but the handling of EXPONENT-UNDERFLOW and INTEGER-OVERFLOW can be controlled separated by means of the EXCEPTION runtime option. The assignment of STXIT events cannot be prevented in the case of COBOL programs and programs that use standard linkage

(C as of V2.0A, C++ as of V2.1A, COBOL85 as of V1.1A, COBOL2000 V1.0A, FOR1 as of V2.2A, Fortran90 as of V1.0A and PLI1 as of V4.1A).

However, in the event of errors which do not affect memory such as address errors or illegal operation code, ILCS offers the following approach: after the STXIT routines have been processed, ILCS restores the former program counter contents, reproduces the error and passes control to the system, thus allowing subsequent processing of the error with the

%ON command. With all other errors ILCS aborts the program.
%ON %ANY or %ON %TERM can be used, however, to stop the program run before it is unloaded to investigate the cause of the error using AID command.

In the case of openUTM applications the openUTM STXIT routines can be deactivated as of V3.2A (interactively for UTM-T and UTM-P) by specification of the option STXIT=OFF in the START parameter. If openUTM is running under ILCS, which is possible as of V3.2A (operand PROGRAM COMP=ILCS in the KDCDEF statement), the ILCS STXIT routines still remain effective even after deactivation of UTM-STXIT.

In Assembler and C++/C programs it is possible to write separate routines (in C++/C: signal handling via signal() library function, in Assembler: STXIT macro) for the purpose of error handling. In this case, too, AID has no possibility of responding via the %ON command to an error intercepted by individually programmed routines. It is possible, though, to use %INSERT to insert a test point in the error handling routines; the associated subcommand will then be executed when the error occurs.

11.2 Programs with an overlay structure

AID normally assumes that a program is linked without an overlay structure. It uses the LSD records once they are loaded without checking every time whether the CSECT that is referenced is contained in a segment that has since been dynamically loaded. However, if the program being debugged is one that was linked statically as an overlay or one that dynamically loads or unloads segments with the BIND/UNBIND macro calls, the operand OV=YES must be specified in the %AID command in order to ensure that AID checks every time the LSD is accessed whether dynamic loading has occurred in the meantime.

In programs with an overlay structure, a test point can only be set in a segment that was loaded at the time the command is entered. Similarly, a test point can only be deleted if the associated segment is loaded. If the segment is unloaded or overwritten, the test point is retained unless it has first been explicitly deleted with %REMOVE. If the segment is loaded again, the test point is also set again.

12 Restrictions and interaction

12.1 %ON %WRITE with %INSERT, %CONTROLn and %TRACE

Attention must be paid to the following interactions between %ON *write-event* and the AID commands %CONTROLn, %INSERT and %TRACE:

– If a %CONTROLn or a %TRACE with a machine-oriented *criterion* is assigned, an entry of %ON *write-event* is rejected with an error message, and vice versa.

– If a machine instruction has been overwritten by a %CONTROLn or %TRACE with a symbolic *criterion* by the internal AID label (X'0A81'), AID does not detect the write access to that instruction.

– If a machine instruction has been overwritten by the test point defined with %INSERT with the internal AID label, again AID does not detect the write access to that instruction.

To achieve continuous write monitoring it is advisable to delete all %CONTROLn and %INSERT commands using %REMOVE and to delete any %TRACE command that may still be entered by continuing with %RESUME after the %ON command.

12.2 Interaction between execution monitoring and the output or modification of memory contents

If %INSERT is used to set a test point, AID overwrites the instruction code at the address of the test point with an SVC X'81'. Similarly, AID labels the first instruction of the executable statements in *control-area* and *trace-area* (with a symbolic *criterion*) with X'0A81'. These labels can be viewed if the relevant instruction code is output using %DISPLAY. If, on the other hand, the code is disassembled using %DISASSEMBLE, AID replaces the entered SVC with the original instruction, revealing the instruction code as it was generated by the compiler. AID also falls back on the original code if an address is to be calculated from the corresponding instruction via the type modification %S or %SX, and that address is to be referenced via a pointer (->).

The labels set in the instruction code can be searched with the %FIND command by specifying X'0A81' as the search key.

In the case of the %MOVE and %SET commands, any labels that are entered are not replaced. This may have the effect when instruction code is transferred that labels disappear or new ones are set which AID is no longer able to bring into connection with its internal command management. The user must therefore personally ensure that before transfer or overwriting takes place no SVCs entered by AID are contained in the relevant instruction code. Test points and labels associated with the %CONTROLn command can be deleted with the %REMOVE command. If a %TRACE is still entered, this can be deleted by starting the program with %RESUME or by entering
%TRACE 1 %INSTR.

12.3 Test points in the common memory pools

If a test point is set in a common memory pool, it is known only to the task that set the test point. All other tasks that are also connected to the common memory pool stop at this test point.

The only way that other users of the common memory pool can avoid this is to reinsert the original code by hand. Otherwise the hung task must wait until the test point is explicitly reset using %REMOVE by the task that set it, or until the task terminates and thereby implicitly removes the test points that it set.

A further problem arises however when the task that set the test point disconnects from the common memory pool without first having removed the test point with %REMOVE. If the task is then ended, implicit resetting of the test point is not performed as there is no longer a connection between task and common memory pool. The original code can then only be recovered manually.

In order to avoid the above problems, it is advisable to run the code locally and not in a common memory pool when debugging. If, however, a test point is required in a common memory pool, it should be removed using %REMOVE once it is reached, so that other users are not prevented from using the common memory pool. During debugging, at least disconnection of the task from common memory should be suppressed so that the test points can be removed implicitly when the task terminates.

12.4 Low level trace and control in conjunction with contingencies

12.4.1 %TRACE

In tasks with event-driven processing, individual machine instructions may be logged incorrectly when running %TRACE at machine code level. If, as a result of an asynchronous event, a contingency routine is called and a low level trace is running at the same time, errors may occur when entering the contingency routine and when returning to the task interrupted by the event. Depending on the particular program and debugging constellation,

You can expect the following errors at the interface between the base process and contingency:

– an instruction is incorrectly logged, e.g. with incorrect register contents
– an instruction is lost
– an instruction is displayed twice.

12.4.2 %CONTROL

Similarly, a subcommand may be omitted or executed too many times during instruction monitoring with %CONTROL.

STXIT routines started when program errors are detected are not affected by the error conditions described above. All instructions used in conjunction with %TRACE and %CONTROL are executed correctly and completely.

13 Messages

AID0250 Internal AID error in module (&00) . Please contact maintenance (CMD: (&01))

Meaning
Error in AID.
(&00) number of the module that detected the error.

Response
Contact the system administrator.

AID0251 No memory available

Meaning
AID has exhausted its memory space.

Response
Terminate the test and contact the system administrator.

AID0252 AID error in module (&00) : RTC (&01) (CMD: (&02))

Meaning
Error in AID.
(&00) number of the module that detected the error
(&01) internal AID error flag

Response
Contact the system administrator.

AID0253 Function not yet implemented (CMD: (&00))

Meaning
The current version of AID or BS2000 does not provide the desired function.

AID0254 System error (&00) (CMD: (&01))

Meaning
System error.
(&00) number of the module that detected the error

Response
Contact the system administrator.

AID0255 System error in module (&00) : RTC (&01) (CMD: (&02))

Meaning
System error.
(&00) number of the module that detected the error
(&01) internal error flag of the system

Response
Contact the system administrator.

AID0256 AID message file not available or inconsistent.

Meaning
The AID text file containing the AID messages is either not available or inconsistent.

Response
Contact the system administrator.

AID0257 Text for message I (&00) cannot be issued.

Meaning
The AID text file containing the AID messages is either not available or inconsistent. The text for message number (&00) cannot be displayed.

Response
Contact the system administrator.

AID0258 Test point already set by another task (CMD: (&00))

Meaning
The test point has already been set by another task and is therefore rejected for the present task.

AID0259 Exponent overflow in floating point literal

Meaning
Actual value of exponent exceeds supported range of -76 <= exp <= +75.

AID0260 File error on (&00) ; DMS error code (&01) (CMD: (&02))

Meaning
Error in file access
(&00)link name or file name
(&01)DMS error code or blank

Response
Remove the error affecting the file with the aid of the DMS error code (cf. BS2000 messages) and reenter the command.

AID0261 File contains no recognizable dump (CMD: (&00))

Meaning
The content of the file could not be recognized as a dump.

AID0262 Insufficient memory to process command (CMD: (&00))

Meaning
Insufficient memory; the amount of data to be processed is too large.

Response
Split the data into subsets for processing, if necessary.

AID0263 Symbolic linkage information (ESD/ESV) missing (CMD: (&00))

Meaning
The command could not be executed because no symbolic linkage information is available.

Response
Repeat linkage of the program to be tested, specifying SYMTEST=ALL or SYMTEST=MAP or TEST-SUPPORT=YES.

AID0264 Name of (&00) qualification not found (CMD: (&01))

Meaning
The name of the specified qualification could not be found in the valid environment
(&00) type of qualification that could not performed: CTX, T, L, O, C, E, D

Response
Modify the name or the valid environment and reenter the command.

AID0265 Invalid qualification:(&00)

Meaning
Invalid qualification.

Response
Enter the correct qualification.

AID0266 O qualification not supported by old linkage format (CMD: (&00))

Meaning
Object module qualification is not supported for programs linked with a TSOSLNK version earlier than V15.

Response
Link the program with a more recent TSOSLNK version and reenter the command.

AID0267 Index not allowed for keyword (CMD: (&00))

Meaning
No index is permitted for the specified keyword.

Response
Enter the keyword without the index.

AID0268 No additional parameter allowed for event (CMD: (&00))

Meaning
No additional parameter is permitted for the specified event.

Response
Enter the event without the additional parameter.

AID0269 AID allows only LSDCHECK=YES

Meaning
LSDCHECK=NO not allowed (opt. REP must be used)

AID0270 Invalid keyword / event (CMD: (&00))

Meaning
Invalid keyword or event.

AID0271 Syntax error (CMD: (&00))

Meaning
? Syntax error / infringement against conventions.
- Syntax is not conforming to the description.
- Syntax is ambiguous.
! Correct the syntax/ to avoid ambiguity you can use brackets

AID0272 Command too long (CMD: (&00))

Meaning
The command entered is too long.

Response
Divide the command into several subcommands and reenter it in this form.

AID0273 Address overflow (CMD: (&00))

Meaning
The address lies outside the valid address range.

Response
Enter a valid address.

AID0274 Change desired? Reply (Y=Yes; N=No)

Meaning
Is the old content to be replaced by the new content ?

AID0275 CSECT qualification required

Meaning
For this command a CSECT qualification is necessary.

Response
Add the correct CSECT qualification.

AID0276 %INSERT or %ON too deeply nested (CMD: (&00))

Meaning
The nesting depth of the entered %INSERT or %ON command is too large.

Response
Change the subcommand string of an internal %INSERT or %ON command to " %STOP ". AID is interrupted when this test point or event is reached, and the user can then enter more nested %INSERT or %ON commands.

AID0277 No qualification defined (CMD: (&00))

Meaning
Reference was made to a prequalification without a %QUALIFY command having been issued.

Response
Enter a %QUALIFY command and reenter the original command.

AID0278 Value outside supported range (CMD: (&00))

Meaning
Value outside supported range.

AID0279 Type not supported (CMD: (&00))

Meaning
The specified type is not supported by AID.

AID0280 Task / file qualification not allowed (CMD: (&00))

Meaning
The T or E=Dn qualification is not permitted in the command entered.

AID0281 Explicit qualification ignored

Meaning
Warning message: the explicit qualification of the operand is superfluous and therefore ignored.

AID0282 TSN not in use

Meaning
The specified TSN does not exist.

AID0283 No information given for dump file (CMD: (&00))

Meaning
When dump file qualification is active, there will be no information given for this command.

AID0284 Address inaccessible

Meaning
The specified address cannot be accessed.

AID0285 Warning: (&00) parameter too long; shortened to "(&01)"

Meaning
The parameter specified in the command to connect a user interface to AID is too long and will be shortened.
(&00) parameter type
(&01) shortened parameter

AID0286 Variable type is not a pointer type (CMD: (&00))

Meaning
Indirect addressing is only possible for variable of the specified type %A.

Response
Specify the variable type to be used for indirect addressing as a type %A (with type modification).

AID0287 Address not within code or linkage info (CSECT list) missing (CMD: (&00))

Meaning
Either the address lies outside the code or there is no CSECT list for the loaded program.

Response
Generate a CSECT list by linking the program again with the specification SYMTEST=ALL or SYMTEST=MAP TEST-SUPPORT=YES .

AID0288 No message with this number

Meaning
There is no message corresponding to this message number

AID0289 No message with this number

Meaning
There is no message corresponding to this message number

AID0290 Test point does not exist (CMD: (&00))

Meaning
The test point to be deleted is not set or no longer set.

AID0291 Event does not exist (CMD: (&00))

Meaning
The event to be deleted is not set or no longer set.

AID0292 %INSERT / %ON rejected (CMD: (&00))

Meaning
The test point or event could not be set.

AID0293 Event not allowed in this OS version

Meaning
Event does not yet exist in this OS version.

AID0294 Page(s) from address (&00) to (&01) not dumped

Meaning
The page(s) from address (&00) to address (&01) is or are not contained in the specified dump file.

AID0295 Page(s) from address (&00) to (&01) not allocated

Meaning
The page(s) from address (&00) to address (&01) are not assigned to the current task.

AID0296 Invalid memory class (CMD: (&00))

Meaning
The specified class keyword is not permitted.

AID0297 TSN qualification required for dump file (&00) (CMD: (&01))

Meaning
The specified dump file cannot be accessed unless a T qualification is specified
(&00) link name

Response
Specify a T qualification.

AID0298 Testpoint removed; code not restored because of user modification

Meaning
User replaced code at testpoint (%SET/%MOVE-command);old code is not restored; testpoint is removed

AID0299 RECFORM parameter error

Meaning
Recform parameter not of type 'V'.

Response
Change recform type.

AID0300 Error on medium (CMD: (&00))

Meaning
The output medium reports an error.

Response
Specify another output medium.

AID0301 Segment of task (&00) inaccessible (CMD: (&01))

Meaning
It is not possible to access the specified address of a foreign task.
(&00)task sequence number

AID0302 System table does not exist or inaccessible (CMD: (&00))

Meaning
The system table doesn't exist or is inaccessible. (Possible reasons: program not loaded, list not within dump file, etc.).

AID0303 Memory allocation error (CMD: (&00))

Meaning
Error during memory allocation.

AID0304 Keyword / event parameter out of range (CMD: (&00))

Meaning
The parameter of the specified keyword / event lies outside the permissible value range.

AID0305 Illegal keyword / event parameter (CMD: (&00))

Meaning
The parameter of the specified keyword or event is not permissible.

Response
Correct and reenter the command.

AID0306 Invalid range specification

Meaning
Invalid range specification

Response
Please choose a valid range specification.

AID0307 Output medium (&00) not available

Meaning
Output medium (&00) is not available.

AID0308 Exception handling: testpoint could not be set

Meaning
Coding for exception handling not yet present;
stop program in a C/C++ function and try again

AID0309 PCB index out of range (CMD: (&00))

Meaning
There is no process control block with the specified index.

AID0310 Terminal request not honored (CMD: (&00))

Meaning
Input/output can only be switched to the operator console or reset to SYSOUT under the system administrator ID TSOS.

AID0311 Error during execution of a system command (CMD: (&00))

Meaning
An error occurred during execution of a system command in a sequence of AID commands.

Response
Correct and reenter the command; then enter the rest of the command sequence.

AID0312 Variable / literal not convertible

Meaning
The variable or literal cannot be converted.

AID0313 No program loaded (CMD: (&00))

Meaning
The specified command cannot be executed unless a program has been loaded.

AID0314 TID not in use

Meaning
This TID is not in use.

AID0315 Dump file not open (CMD: (&00))

Meaning
The dump file has not been opened.

Response
Open the dump file with the %DUMPFILE command and reenter the original command.

AID0316 %TITLE string too long (CMD: (&00))

Meaning
The string specified in the %TITLE command is too long; the maximum length is 80.

Response
Shorten the %TITLE string and reenter the command.

AID0317 Program enters STXIT / CONTINGENCY routine (PC: (&00))

Meaning
User enters STXIT or CONTINGENCY routine.
(&00)Content of the program counter

AID0318 Further (&00) byte(s) not displayed

Meaning
A number of further (&00) bytes of the bytes to be moved will not be displayed.

AID0319 PARTNER/ROUTE/IP address unknown or inactive

Meaning
A connection to the user interface could not be established because partner/route/IP-address were unknown or inactive.

AID0320 User interface or version of user interface not supported

Meaning
A user interface or this version of a user interface is not supported.

AID0321 TID / TSN not in use

Meaning
This TID/TSN is not in use.

AID0322 ONLY parameter invalid (CMD: (&00))

Meaning
The ONLY parameter of the %INSERT or %ON command contains an error.

Response
Correct the ONLY parameter and reenter the command.

AID0323 No match found for virtual address (CMD: (&00))

Meaning
The program does not contain the virtual address, i.e. it is not possible to specify the name of a program, load module, object module or CSECT which contains this address (e.g. because the linkage editor did not generate a CSECT list).

Response
If no CSECT list exists, link the program again with the specification SYMTEST=ALL or SYMTEST=MAP or TEST-SUPPORT=YES and reenter the command.

AID0324 Privilege too low but can be raised

Meaning
The test privilege for the desired function is too low, but could be raised by means of the SDF command /MODIFY-TEST-OPTIONS.

Response
Raise the test privilege by means of the /MODIFY-TEST-OPTIONS command.

AID0325 Privilege too low and cannot be raised

Meaning
The test privilege for the desired function is too low and cannot be raised to a suitable level even with the aid of the SDF command /MODIFY-TEST-OPTIONS.

Response
The user cannot use the desired function under his user ID. Request the system administrator to raise the test privilege.

AID0326 IEEE floatingpoint value not a number

Meaning
The IEEE floatingpoint value does not represent a number

AID0327 %LOC parameter invalid (CMD: (&00))

Meaning
The %LOC function contains an incorrect parameter.

Response
Correct the parameter and reenter the command.

AID0328 Invalid test point location (CMD: (&00))

Meaning
The address of the test point is invalid.

Response
Correct the test point address and reenter the command.

AID0329 Execution of %HELP command impossible (CMD: (&00))

Meaning
The %HELP command cannot be executed. Reason: the AID text file is missing or inconsistent.

Response
Contact the system administrator.

AID0330 System table empty (CMD: (&00))

Meaning
The accessed system table is empty.

AID0331 Invalid link name (CMD: (&00))

Meaning
The link name is invalid.

Response
Correct the link name and reenter the command.

AID0332 Access to dump file on tape not supported (CMD: (&00))

Meaning
The dump file on tape cannot be accessed.

Response
Transfer the dump file to disk and reenter the command.

AID0333 Inconsistency detected by AID in module (&00) (CMD: (&01))

Meaning
AID has detected an inconsistency.
(&00)number of the module that detected the error

Response
Inform the system administrator.

AID0334 Invalid length (CMD: (&00))

Meaning
The length is invalid.

Response
Correct the length and reenter the command.

AID0335 No message with this number

Meaning
There is no message corresponding to this message number

AID0336 File not open (CMD: (&00))

Meaning
The file is not open.

AID0337 Invalid command name or invalid msg nr. in %HELP command (CMD: (&00))

Meaning
Invalid command name in the %HELP command, or the specified message number does not exist.

AID0338 %INSERT / %ON not allowed for foreign task or dump file (CMD: (&00))

Meaning
A foreign task or dump file cannot be accessed by means of the %INSERT or %ON command.

AID0339 Interrupt in connection to user interface. Connection closed

Meaning
There was an interrupt in the connection to the client. Connection is closed

AID0340 Terminal output terminated by user interrupt

Meaning
Output to SYSOUT was terminated due to an interrupt generated by a user.

AID0341 %MOVE/%SET rejected because of intermediate modification of old content

Meaning
In the case of a %MOVE/%SET command, the old contents have been modified in the time between their output and the entering of "Y" in response to the query "Change ... (Y=Yes; N=No)". For this reason the modification requested in the %MOVE / %SET command is rejected.

Response
Reenter the command. If necessary remove the CHECK parameter from %AID command.

AID0342 Nothing changed

Meaning
No change was performed. The user responded to the query "Change ... (Y=Yes; N=No)" by entering "N".

AID0343 Change of dump file not supported (CMD: (&00))

Meaning
A modification to a dump file is not supported by AID.

AID0344 LSD version for SOURCE module (&00) not supported

Meaning
The compiler has generated LSD for the source module in a version that cannot be processed by AID.

Response
Either the source module (and maybe others) must be translated by a compiler which generates LSD in a version supported by AID or a version of AID is needed which can process the version of LSD generated by the compiler.

AID0345 %MOVE/%SET exceeds segment boundaries; only (&00) bytes moved

Meaning
The modification extends beyond the segment boundaries; only (&00) bytes were modified.

Response
Use a seperate %MOVE command to transfer the remaining bytes to be modified.

AID0346 Location to be changed not allocated (CMD: (&00))

Meaning
The memory location to be modified has not been located.

AID0347 Array (&00) must be subscripted (CMD: (&01))

Meaning
It is necessary to subscript the array (&00) in this case.

AID0348 Program stopped due to (&00) event (&01)

Meaning
An event (&00), that was watched due to an AID switch (FORK or EXEC) has happend or a %STOP command has been entered in another task of the family. In fork tasks the PID is given in (&01).

AID0349 No message with this number

Meaning
There is no message corresponding to this message number

AID0350 %FIND without parameters not allowed in this context (CMD: (&00))

Meaning
A %FIND command without parameters is not permitted in this context.

Response
Reenter the %FIND command with parameters (with at least a "search-criteria").

AID0351 No match in range

Meaning
No match was found in the %FIND area.

AID0352 No additional match in range

Meaning
No additional match was found in the %FIND area.

AID0353 Length exceeds boundaries; only (&00) bytes moved.

Meaning
The specified length exceeds the boundaries; only (&00) bytes were transferred.

AID0354 Given length exceeds boundaries; nothing moved (CMD: (&00))

Meaning
The specified length exceeds the boundaries of a keyword or a variable ; no bytes were transferred.

AID0355 %MOVE / %SET rejected

Meaning
The %MOVE / %SET command was rejected.

AID0356 Keyword not allowed in command (CMD: (&00))

Meaning
This keyword is not permitted in the command entered.

AID0357 Modification not allowed for keyword (CMD: (&00))

Meaning
The modification entered is not permitted for this keyword.

AID0358 No information given for foreign task (CMD: (&00))

Meaning
When foreign task qualification is active, there will be no information given for this command.

AID0359 (&00) is neither a class nor a namespace specification

Meaning
The name in front of the '::'-operator is neither the name of a class nor the name of a namespace .
(&00) name of the class or the namespace

AID0360 "::" must not succeed a BLK/PROC qualification

Meaning
The global data scope operator '::' may not succeed a block or procedure qualification .

AID0361 The command is not allowed in dump file environment

Meaning
The command is allowed only with base qualification on virtual storage.

AID0362 Function is locked (CMD: (&00))

Meaning
The function in question cannot currently be called.

AID0363 %MOVE exceeds CSECT boundaries; REP(s) suppressed (CMD: (&00))

Meaning
No REP information could be generated because the receiver in the %MOVE command exceeds the CSECT boundaries.

Response
Split the %MOVE command into several %MOVE commands which do not exceed the CSECT boundaries.

AID0364 No match found for target address; REP(s) suppressed (CMD: (&00))

Meaning
Receiver could not be found due to lack of CSECT information; no REP information was generated.

Response
If necessary, link the program again with SYMTEST=ALL or SYMTEST=MAP or TEST-SUPPORT=YES .

AID0365 REP generation error; no REP issued (CMD: (&00))

Meaning
An error occuring during output of REP information to a file, e.g. syntax error, no medium available, no file available.

AID0366 File error : (&00) ; no REP issued (CMD: (&01))

Meaning
An error was detected during output of REP information to file (&00); the REP Information was not generated.

AID0367 File (&00) cannot be opened; DMS error code (&01); no REP issued (CMD: (&02))

Meaning
DMS reported error (&01) during output of REP information to file (&00); the REP Information was not generated.

AID0368 FCB type not supported for file (&00) ; no REP issued (CMD: (&01))

Meaning
An attempt was made to output REP information to a file generated by the user; assignment takes place via link name (&00). The FCB type of the file is not supported by AIDSYS. No REP information was generated.

Response
Modify the FCB type and reenter the command.

AID0369 No linkage information available; REP(s) suppressed (CMD: (&00))

Meaning
The program to be tested was linked with a linkage editor version earlier than V16, or the creation of CSECT lists was suppressed. Localization on object modules is not possible. An LMS correction statement cannot be output and is therefore suppressed.

Response
Link the program again with TSOSLNK >= V16, specifying the parameter SYMTEST=ALL or SYMTEST=MAP or TEST-SUPPORT=YES then reenter the command.

AID0370 File error (&00) or SLED without virtual address space (CMD: (&01))

Meaning
Either an error occurred during access to the file with link name (&00), or the SLED file does not have a virtual address.

AID0371 Task/file qualification not allowed for event (CMD: (&00))

Meaning
When specifying events it is not permitted to qualify a foreign task or a dump file.

AID0372 Task/file qualification not allowed for test point (CMD: (&00))

Meaning
When specifying test points it is not permitted to qualify a foreign task or a dump file.

AID0373 Stop not possible for given (&00). (Reason (&01))

Meaning
The command is not possible if task (&00) is
Reason -10 : own task.
-14 : no fork task or does not exist.
-18 : no member of the task family.

AID0374 Requested information not in dump file

Meaning
Not all the pages required to perform the requested function are contained in the dump file.

AID0375 (&00) (&01) not found

Meaning
Either source module, procedure/block, symbol or source reference could not be found in the currently valid or explicity specified environment, or source module could not be found in the loaded program.
(&00)SOURCE_MODUL, PROC/BLK, SYMBOL, SRC_REF_#
(&01)name

AID0376 Ambiguous or incomplete qualification for (&00) (&01)

Meaning
The qualification for procedure or symbol is not unique or is incomplete when full qualification is required.
(&00) PROC or SYMBOL
(&01) name of the procedure (PROC) or symbol (SYMBOL)

Response
Select a unique qualification or a complete qualification if it is required.

AID0377 Symbolic information inconsistent for (&00)

Meaning
The symbolic information was not generated in the same compilation operation as the loaded object.
(&00)source module

Response
Specify the correct library, or repeat compilation

AID0378 Symbolic information missing

Meaning
A command was entered that requires symbolic information. This information is incorrect or missing.

Response
Compile using symbolic information option, link and load, or specify the library for the symbolic information or use commands for debugging on machine code level (e.g. %TRACE %INSTR).

AID0379 S and PROC qualification required or LSD information missing

Meaning
The interrupt point does not lie within a unique source module and procedure environment or there is no LSD information (symbolic information) available.

Response
Specify a S and PROC qualification or symbolic information.

AID0380 Invalid explicit basing (&00) (&01) (CMD: (&02))

Meaning
Either the base is not of the type "pointer", or the variable type was not declared as "based".
(&00)name
(&01)->

AID0381 (&00) (&01) not of type INTEGER (CMD: (&02))

Meaning
Either multiplier is not an integer, or index is not of the type "integer".
(&00) INDEX or MULTIPLIER
(&01) name of index (INDEX) or multiplier (MULTIPLIER)

AID0382 Invalid dimension of array (&00) (CMD: (&01))

Meaning
Array (&00) has the wrong dimension.

AID0383 No subscript allowed for symbol (&00)

Meaning
Symbol (&00) must not have a subscript.

AID0384 Component list conflicts with structure type/DSECT (&00) (CMD: (&01))

Meaning
The layout for the mode constant does not tally with base model (&00).

AID0385 (&00) (&01) not within nest (CMD: (&02))

Meaning
Source module or procedure is not contained in the current call hierarchy.
(&00)SOURCE-MODUL oder PROC
(&01)name of source modul or procedure

AID0386 No message with this number

Meaning
There is no message corresponding to this message number

AID0387 Too many PROC/BLK qualifications (CMD: (&00))

Meaning
The nesting depth of the PROC or BLK is too deep (&00)PROC or BLK

Response
Select a shorter qualification that is still unique.

AID0388 Types are not convertible; nothing changed/compared (CMD: (&00))

Meaning
The types are not convertible. The value of the receiver was not changed (%SET) or the comparison in a subcommand condition is not practicable and supplied by FALSE (CONDITION).

AID0389 %SET array INTO array not yet implemented (CMD: (&00))

Meaning
The function is not supported by the AID version.

AID0390 Warning: source truncated

Meaning
Warning message: the converted value does not conform to the original value.

Response
None unless explicity desired, in which case reenter the command with the correct input value.

AID0391 Task with (&00) is unknown.

Meaning
Task with given TSN/PID (&00) was not found.

AID0392 Value(s) of >(&00)< does(do) not match to type declaration

Meaning
The actual content of the current or following data or data element (&00) does not correspond to the declared type or contents of one or several array elements don't match to type.

AID0393 Symbol too complex or a too deeply nested struct component (CMD: (&00))

Meaning
There are too many components or AID restrictions do not allow debugging of symbol expression

Response
Continue processing with a series of subcomponents

AID0394 Structure type/DSECT without comp. list/explicit basing not supported

Meaning
Model names can only be used as layout identifiers by mode constants.

AID0395 Division by zero

Meaning
Division by zero is prohibited.

AID0396 Invalid address for (&00)

Meaning
The memory object (&00) cannot currently be accessed.

AID0397 (Next component of) (&00) has length <= 0 or is an empty string

Meaning
The length of the component (&00) has a length <= 0 or is an empty string (C/C++)

AID0398 Symbol (&00) represents no instruction address

Meaning
The symbol (&00) does not represent an instruction address.

AID0399 Odd or unallocated address

Meaning
The address is either uneven or not occupied.

AID0400 Dimension (&00) of array (&01) out of range or array has no element

Meaning
The dimension (&00) of the array (&01) lies outside the permissible value.

AID0401 %MOVE / %SET into constant not allowed

Meaning
Constants cannot be overwritten by the %MOVE or %SET command.

AID0402 Warning: absolute value moved!

Meaning
Warning message: Receiver is an unsigned numeric value; the absolute value of sender was transferred.

AID0403 (&00) must not be indexed by (&01)

Meaning
(&00) is a special index data field or a special index with a base field other than (&01).

AID0404 AID cannot reference statement (&00) due to compiler optimization

Meaning
AID cannot use the statement with source reference (&00) due to compiler optimization.

AID0405 Too many libraries

Meaning
More than 15 libraries were signed on by means of the %SYMLIB command.

Response
Sign off superfluous libraries by means of the empty %SYMLIB command and then sign less than 15 libraries.

AID0406 AID cannot access library (&00); DMS error code (&01)

Meaning
AID cannot currently access the library (&00); (&01) DMS error code. Possible reasons are : protected/locked/password missing.

Response
Possibly solve the problem and then reenter the %SYMLIB command.

AID0407 File (&00) is not a PLAM library

Meaning
File (&00) is not a PLAM library; the remaining libraries of the %SYMLIB command are signed on.

AID0408 Member (&00) of library (&01) is LOCKED

Meaning
AID cannot currently access the member (&00) of the library (&01).

Response
Make the member accessible and reenter the AID command.

AID0409 Only positive INTEGER values allowed for special index

Meaning
The receiver is of the type "index"; its contents must always be positive integer values.

AID0410 %TRACE / %CONTROL not supported in foreign task or dump file

Meaning
The %TRACE or %CONTROL command is only permitted in the user's own task.

Response
Correct and reenter the command, taking a valid %BASE command into account.

AID0411 File with specified link (&00) is already open

Meaning
The file with the link name (&00) has already been opened.

Response
a) Close the dump file with "%DUMPFILE Dn" and reopen it with "%DUMPFILE Dn=filename"
b) Use a different link name.

AID0412 Symbol (&00) not within nest and not of storage class STATIC/CONSTANT

Meaning
The symbol (&00) cannot currently be referenced.

Response
Reenter the command at a time when the symbol is within the call nesting.

AID0413 Range exceeds segment / CSECT boundaries

Meaning
The value range exceeds the segment or CSECT boundaries.

Response
Specify the range again.

AID0414 Range specification incorrect

Meaning
Invalid range specification.

AID0415 Combination of HIGH LEVEL range and LOW LEVEL mode illegal

Meaning
Operands for symbolic level and operands for machine code level should not be mixed in this command.

AID0416 (&00) was not set

Meaning
The %CONTROL command to be deleted (%C1 to %C7) was not set.

AID0417 Program counter not within code; use expl. qualification (PROG= / S=)

Meaning
The program counter lies outside the user program; use the S or PROG qualification.

AID0418 Invalid parameter combination in this context (CMD: (&00))

Meaning
There were several -or at least one -invalid parameters given with this command.

AID0419 Compiler register optimization prohibits modification of variable (&00)

Meaning
The value of the variable (&00) cannot currently be changed as this could result in inconsistencies in the further processing of the program.

Response
Retry the command at another point in the program.

AID0420 AID cannot reference variable (&00) due to compiler optimization

Meaning
The variable (&00) was declared but is not referenced in the program. AID cannot reference it because of compiler optimization.

AID0421 Nested %INSERT on label or entry variable not allowed (CMD: (&00))

Meaning
When a test point is encountered or an event occurs, the subcommand must not be an %INSERT command for a label or entry variable.

AID0422 Program stack corrupted (invalid back link or stack address)

Meaning
AID cannot interpret a program stack due to corruption of the stack chaining (e.g. because of a program error).

Response
Localize the program error with the aid of machine-oriented AID functions.

AID0423 No (&00) set

Meaning
The entered %REMOVE/%SHOW command had no effect as no %CONTROLn, %INSERT or %ON was signed on.

AID0424 File (&00) could not be opened; DMS error code (&01)

Meaning
DMS detected error (&01) (cf. the BS2000 system messages) during output of information to file (&00).

Response
Correct the error and reenter the command.

AID0425 FCB type not supported for file (&00)

Meaning
An attempt was made to output REP information to a file generated by the user; assignment takes place via link name (&00). The FCB type of the file is not supported by AIDSYS.

Response
Modify the FCB type and reenter the command.

AID0426 Warning: ambiguity of (&00) (&01) not completely checked (OVERLAY loading)

Meaning
Warning message: The specified procedure or symbol was identified as unique in the loaded portion of the program linked by means of the overlay method, but it was not possible to check whether it is unique throughout the source module.
(&00)PROCEDURE or SYMBOL
(&01)name of procedure or symbol

AID0427 (&00) (&01) not found in loaded part of SOURCE module (OVERLAY loading)

Meaning
Name of procedure or symbol could not be found in the loaded portion of the program linked by means of the overlay method. The non-loaded portion could not be checked.
(&00)PROCEDURE or SYMBOL
(&01)name of procedure or symbol

AID0428 Warning: LSD information corrupted; failure possible

Meaning
The LSD information (symbolic information) has at least one error, AID possibly works erroneous, symbolic testing is possible

AID0429 No message with this number

Meaning
There is no message corresponding to this message number

AID0430 Name of dump file too long

Meaning
The specified dump file could not be opened as AID cannot process file names that are longer than 54 characters.

Response
Rename the dump file by means of the /MODIFY-FILE-ATTRIBUTES command and modify the AID command accordingly.

AID0431 Requested information not within dump file

Meaning
The dump file does not contain the requested information.

AID0432 LSD-extra-information for source-modul (&00) is invalid

Meaning
Extra LSD-Information for testing with a graphical user interface is invalid or corrupted.

AID0433 LSD information corrupted; symbolic test not possible

Meaning
The LSD information (symbolic information) was errored or overwritten. Symbolic testing not possible.

Response
Recompile the program, or send an error message to system diagnosis.

AID0434 Offset operation only admitted for an operand yielding an address

Meaning
An offset operation is only possible for Operands that supply an address.

Response
Insert a memory reference with address attribute before performing the offset operation.

AID0435 Warning: no output given for specified operands (CMD: (&00))

Meaning
Warning message: the specified command does not generate any output; e.g. name is not found or ambiguous.

AID0436 AID cannot reference symbol (&00) due to incomplete LSD information

Meaning
The symbol (&00) was declared but cannot be referenced by AID as the LSD information (symbolic information) by the compiler is incomplete.

AID0437 CSECT/ENTRY has length 0

Meaning
CSECT/ENTRY cannot be output as its length is 0.

AID0438 String too long

Meaning
The string entered exceeds the permissible length.

Response
Shorten the string.

AID0439 Name too long

Meaning
The name entered exceeds the permissible length.

Response
Enter a proper name.

AID0440 Source info file for %TRACE doesn't match with loaded object (CMD: (&00))

Meaning
The edited text file for %TRACE information has not been generated in the same compilation operation as the loaded object.

Response
If none, all %TRACE output is given according to LSD-defined format; else compile source again, link and generate new text file for %TRACE information.

AID0441 Wrong continuation in %TRACE command

Meaning
The continuation parameter in the %TRACE command must be 'R' or 'S'

AID0442 Keyword does not yet exist in this OS version

Meaning
The specified keyword has been introduced in a more recent version of the operating system; it is undefined in the OS version of the test object.

AID0443 Keyword no longer exists in this OS version

Meaning
The specified keyword exists in an earlier version of the operating system; it is undefined in the OS version of the actual test object.

AID0444 Outfile could not be opened

Meaning
The specified file could not be opened as an outfile.

Response
Check whether the file has already been opened.

AID0445 Source modul (&00) has no LSD for source based debugging.

Meaning
Test with graphical user interface based on source code is impossible because of missing or incorrect extra LSD.

AID0446 Variable (&00) has neither implicit nor explicit base

Meaning
The variable (&00) has been declared without any based pointer. A reference to this variable with AID is only possible when an explicit based pointer is specified.

Response
Please specify an explicit based pointer
(e.g. expl_ptr -> variable).

AID0447 No libraries existing to be released

Meaning
At present no library requested by user is allocated to specified link names (%SYMLIB (E/D)=...) or to all link names (%SYMLIB).

AID0448 Pointer value exceeds BIT pointer boundary

Meaning
Type of sender is pointer, type of receiver is bit pointer. Furthermore the high-value byte of sender is not equal to X'00'.

Response
Set receiver explicitly with %SET X'00' into ... %XL1.

AID0449 OFFSET ptr value exceeds BIT OFFSET ptr boundary; nothing changed

Meaning
Type of sender is offset pointer, type of receiver is bit offset pointer. Furthermore the high-value byte of sender is not equal to X'00'.

Response
Set receiver explicitly with %SET X'00' into ... %XL1.

AID0450 Warning: BIT offset ignored

Meaning
High-value byte of sender is not equal to X'00'.
sender:bit pointer
receiver:pointer
or
sender:bit offset pointer
receiver:offset pointer
and high-value byte (equal to bit offset) of sender is not equal to X'00'.

AID0451 Length of (&00) could not be determined by AID

Meaning
AID cannot determine the length of the variable string, because the compiler is not able to give the appropriate information (for example because of being dependent on an expression).

AID0452 Boundary of dimension (&00) of array (&01) couldn't be determined by AID

Meaning
AID cannot determine the boundaries of the variable array, because the compiler is not able to give the appropriate information (for example because of being dependent on an expression) or the boundaries are dependent on non-initialized variables.
(&00)dimension
(&01)name of the array

Response
Value assignment to non-initialized variables.

AID0453 Type of symbol (&00) not described in LSD information

Meaning
AID cannot display the symbol (&00) because there is no LSD information (symbolic information) about the type of the symbol

AID0454 Range parameter error: upper bound lower than lower bound

Meaning
The area boundaries of the %CONTROL and %TRACE command has to be specified in ascending order.

Response
Please specify the area in the correct order.

AID0455 Unknown CPU-Type or operating system version

Meaning
AIDSYS yields unknown or wrong information about hardware type or operating system version.
Therefore the machine code level %TRACE may record wrong instructions or SVCs.

Response
Contact the system administrator.

AID0456 No message with this number

Meaning
There is no message corresponding to this message number

AID0457 Surplus bits of source have been ignored

Meaning
Sender greater than receiver; surplus bits have been ignored.

AID0458 Test point not set; entry couldn't be confirmed within LSD Information

Meaning
Before AID is able to set a test point into an entry name the address of the entry point has to be verified by means of the LSD information.
Possible reasons for errors:
1) Entry variable contains an invalid address (non-initialized overwritten)
2) No LSD information exists for the entry point
3) Entry point cannot be verified as an entry in the LSD information

Response
1) Initialize variable or use variable after allocation
2) Reload LSD information

AID0459 %SET may only be applied to components of complex number variables

Meaning
Variables of the type complex can only be changed one component at a time (real and imaginary part).

Response
Please execute the change one component at a time.

AID0460 Actual value of variable (&00) doesn't match with predefined TRUE / FALSE

Meaning
Warning message: The logical variable (&00) has a current value that is not consistent with the standard values defined for TRUE/FALSE by the compiler.
According to the compiler TRUE/FALSE value is always determined.

AID0461 Too many statement types

Meaning
There are more statement types assigned to the program address of this %TRACE output line than can be output. As many statement types as possible will be output.

AID0462 %JUMP to given target (&00) not allowed

Meaning
An invalid address (&00) has been specified as "continuation" operand in the %JUMP command.

Response
Please choose a valid address.

AID0463 Given target (&00) is not within the actual valid program/procedure

Meaning
The %JUMP command can only be used to branch within the currently active procedure or program.
(&00)address specified as "continuation" operand

Response
Please choose a valid address.

AID0464 Given target (&00) of %JUMP command doesn't represent a program label

Meaning
The operand "continuation" of the %JUMP command has to describe a code address; data addresses are not permitted.
(&00)address specified as "continuation" operand

Response
Please choose a valid address.

AID0465 Offset / length exceeds (&00) boundaries

Meaning
The defined offset value or the explicit/implicit length or the combination of the two exceeds the area limits of the memory reference (&00).

Response
Either change offset value or length as required or switchover to machine code level with the aid of %@(...)->.

AID0466 Back tracking information not found or inconsistent

Meaning
The module AIDIT0, which contains the back tracking information,
- is not available in linked program,
- is only partially available,
- contains an old inconsistent version or
- is not yet availible just after loading the programm.

_ has no entry for a language indicator speciefied in a CSECT of active nesting.

AID0467 Start/end address exceeds CSECT/keyword/symbol boundaries

Meaning
Either the start address or end address does not lie within the area defined by CSECT/ keyword/symbol.

Response
Reenter the command with correct input values.

AID0468 Label not allowed for range description

Meaning
A statement name cannot describe a %TRACE or %CONTROL area.

AID0469 Illegal jumping off place

Meaning
Branching by means of the %JUMP command is not possible here, because either the program has not been initialized, or the test point lies within a run timeroutine, or no LSD information is available for this procedure.

Response
Let the program run to a valid exit point.

AID0470 Arithmetic overflow when (&00) is calculated

Meaning
User error: An arithmetic overflow occurred during calculation of address, length or subscript.

Response
Reenter the command with correct input values.

AID0471 HIGH LEVEL trace / control not allowed in program (&00)

Meaning
The %TRACE or %CONTROL command with symbolic level criterion is not supported in this program unit.

Response
Use command with machine code level criterion.

AID0472 LSD information is incomplete or wrong for CSECT "(&00)"

Meaning
The LSD information (symbolic information) is incomplete/wrong for CSECT (&00).

Response
Link and/or compile the program again.

AID0473 Condition not supported (CMD: (&00))

Meaning
The condition (&00) is not supported by AID.
The logical operands must be a comparison.

AID0474 No message with this number

Meaning
There is no message corresponding to this message number

AID0475 Subcommand label already exists

Meaning
The subcommand name entered already exists.

Response
Use a different name for the subcommand.

AID0476 Too many subcommand labels defined

Meaning
Too many subcommand names defined.

Response
Remove subcommands that are no longer required.

AID0477 Subcommand label does not exist: (&00)

Meaning
The subcommand label (&00) has not been defined or has already been removed.

Response
Enter the correct subcommand label.

AID0478 No message with this number

Meaning
There is no message corresponding to this message number

AID0479 Labels, source-references, template_instances must not be subscripted

Meaning
Syntax error

Response
Correct syntax

AID0480 Address selection, offset, type modification not allowed for constants

Meaning
Address selector, type modification, offset operator must not be applied to constants.

AID0481 No LSD information provided for address to be located

Meaning
No LSD (symbolic information) description is provided for the address which is to be located. which is to be located.

Response
Load LSD from library or recompile module containing address to be located.

AID0482 Address cannot be located in LSD information

Meaning
The address cannot be associated with a source reference (e.g. it is part of the procedure prolog).

AID0483 Reference to undefined subcommand label

Meaning
Reference to the name of the current subcommand label by %. is only permissible within the active subcommand.

Response
Refer to the name of the subcommand by %.subcmdname.

AID0484 Subcommand label too long

Meaning
The subcommand label must not be longer than 30.

Response
Shorten the subcommand label.

AID0485 "*" operator is only allowed for type pointers

Meaning
The variable following the '*' operator must be a type pointer.

AID0486 Type modification of symbol (&00) not allowed

Meaning
The address of symbol (&00) is not byte-aligned or the symbol has bit length.

AID0487 Test point is not a CLASS6 address

Meaning
The address of the test point is not contained in class 6 memory.

AID0488 OVERLAY not loaded

Meaning
The segment to which a test point is to be set or from which it is to be removed is not loaded.

Response
Set or remove the test point when the segment is loaded.

AID0489 Address in %ON %WRITE command invalid

Meaning
The address specified in the %ON %WRITE command is invalid.

Response
Enter a valid address.

AID0490 %ON %WRITE is not allowed when LOW LEVEL %CONTROL/%TRACE is active

Meaning
It is not permissible to enter %ON %WRITE while a %TRACE or %CONTROL command with a "criterion" for debugging on machine code level (%INSTR, %B, %BAL) is active.

Response
Remove %TRACE by entering %TRACE 1 %INSTR;
remove %CONTROLn by entering %REMOVE %C.

AID0491 LOW LEVEL %CONTROL/%TRACE is not allowed when %ON %WRITE is active

Meaning
%TRACE or %CONTROLn with a "criterion" for debugging on machine code level (%INSTR, %B, %BAL) is not permissible when %ON %WRITE is active.

Response
Enter %REMOVE %WRITE.

AID0492 %STOP was sent to fork task ((&00)).

Meaning
A contingency routine was installed for the given task. It will set the task into debug mode at next task activating.

AID0493 %ON %WRITE command is only allowed in status TU

Meaning
%ON %WRITE must not be entered in privileged status.

Response
Transition to non-privileged status.

AID0494 Test point must not be set into CLASS6 memory pools

Meaning
It is not permissible to set test points in class 6 memory pools.

AID0495 No message with this number

Meaning
There is no message corresponding to this message number

AID0496 Warning: previously defined event %WRITE is replaced

Meaning
Warning message: As it is not permissible to define more than one %WRITE event, any previously defined %WRITE event is replaced.

AID0497 Length of area in %ON %WRITE command exceeds 65535 bytes

Meaning
It is not permissible to supervise an area longer than 65535 bytes with an %ON %WRITE command.

Response
Shorten the length.

AID0498 Hardware does not support ALET/SPID qualification

Meaning
ALET/SPID qualification is only permissible when ESA hardware is available.

Response
Try without ALET/SPID qualification.

AID0499 Value of ALET is unknown

Meaning
The value of ALET is unknown.

Response
Enter a correct value.

AID0500 Number of DATA SPACES changed during execution of command

Meaning
The number of data spaces changed during execution of the command.

Response
Retry later.

AID0501 CTX qualification not allowed in this OS version

Meaning
AID does not support CTX-Qualification in this BS2000 Version

AID0502 No DATA SPACES dumped

Meaning
There are no data space dumps to be found in the dump file.

AID0503 DATA SPACE with specified SPID not found in dump file

Meaning
There is no data space with the specified SPID in the dump file.

AID0504 DATA SPACE with specified ALET not found in dump file

Meaning
There is no data space with the specified ALET in the dump file.

AID0505 Hardware does not support ESA

Meaning
This hardware does not support ESA.

AID0506 No DATA SPACES defined

Meaning
There are no data spaces defined at the moment.

AID0507 DATA SPACE not allowed for event

Meaning
Data spaces are not allowed for events.

AID0508 No symbolic library opened

Meaning
There is no symbolic library opened

AID0509 No qualification defined

Meaning
There is no qualifikation defined

AID0510 No %FIND command entered

Meaning
There was no %FIND command entered

AID0511 No outfile assigned or opened

Meaning
There is no outfile assigned by the command %OUTFILE or no outfile opened.

AID0512 No AID command entered in this task

Meaning
There was no actual AID command entered yet.

AID0513 No subcommand label defined

Meaning
There was no subcommand label defined yet.

AID0514 Connection name (OWN) already busy.

Meaning
The given name of the own connection access point is already in use.

AID0515 Connection name (APPL) or route unknown.

Meaning
The given name of the remote connection access point or the route is unknown.

AID0516 Connection name (APPL) busy or inactive.

Meaning
The given name of the remote connection access point is busy or inactive.

AID0517 User interface internal error (&00) in module (&01)

Meaning
There was an error in the user interface or in the communication of the user interface with AID.
(&00) internal AID error flag.
(&01) number of the module that detected the error.

Response
Contact the system administrator.

AID0518 Connection ID wrong.

Meaning
The connection-ID is wrong.

Response
Type in correct connection-ID.

AID0519 Warning: no output given

Meaning
There is no Information for output.

AID0520 User interface not connected.

Meaning
The user interface was not connected to AID because an error happened.

AID0521 User interface already connected; input ignored.

Meaning
There is only one connection to a user interface allowed at a time. Another attempt to connect will be ignored.

AID0522 Check dialog is not allowed, when user interface is connected.

Meaning
A check dialog is not allowed, when a connection to a user interface is active.

AID0523 Warning: Check dialog aborted.

Meaning
Connecting a user interface, while a check dialog is active, will terminate the check dialog automatically.

AID0524 Partial array operand not allowed for function/selector

Meaning
A function or selector (as %@ or %L) may not be applied to a partial array as operand.

AID0525 Register specification is not allowed on /390 objects.

Meaning
It is not allowed to use this register specification on /390 objects.

AID0526 Hardware does not support %(&00).

Meaning
Using the keyword (&00) in the case of this hardware is not allowed.

AID0527 Program mode RM does not support keyword.

Meaning
Keyword is not supported during program mode RM.

AID0528 Warning: /390-trapcode for absolute-address

Meaning
/390-trap on an absolut-address; if code is Risc-code testpoint must be removed!

AID0529 Symbol (&00) ambiguous because of using directive .

Meaning
the symbol (&00) is not unique because of a using directive .

AID0530 Alias name (&00) is undeclared.

Meaning
Alias name has not been declared or was deleted.

AID0531 Alias name (&00) is ambiguous.

Meaning
The specified alias-name is ambiguous because of preceding declaration.

AID0532 No message with this number

Meaning
There is no message corresponding to this message number

AID0533 Too many active alias declarations.

Meaning
Too many active alias declarations. Delete some unnecessary ones.

AID0534 Pool overflow for alias declarations.

Meaning
Storage pool overflow for alias declarations. Delete unnecassary ones.

AID0535 Alias declaration contains an alias name (&00) specified previously.

Meaning
Alias declarations may not contain alias names declared in preceding declarations. Replace the the formerly declared alias name by its substitute or take new name

AID0536 Warning! Alias name (&00) is an non-percent aid-keyword.

Meaning
Alias name is an non percent AID-keyword. Replacement of the alias name in some AID commands may produce a syntax error.
syntax error

AID0537 Alias name is too long.

Meaning
Maximum size of alias names is 32 bytes.

AID0538 There exist no alias declarations.

Meaning
There is no active alias declaration.

AID0539 SPL4-Stack-globals not supported.

Meaning
SPL4-Stack-globals are not supported because of a address path depending on the runtime-system.

AID0540 Constant expressions in template arguments not supported.

Meaning
Currently evaluation of constant expression in template arguments is not supported. Replace expression by its value of result.

AID0541 Simple type specification too long

Meaning
Specification of a simple type is too long. Eliminate redundant tpye specifiers.

AID0542 Alias names not allowed in %QUALIFY command

Meaning
Alias names are not allowed in a %QUALIFY command. If you want to know which Alias names are declared, enter %SHOW %ALIAS.

AID0543 A proc-qualification may not contain a structured name.

Meaning
if lsd-version generated for test object is less or equal 6 a proc qualification may not contain a structured name

AID0544 No message with this number

Meaning
There is no message corresponding to this message number

AID0545 Pointer to member value invalid

Meaning
The internal value of the pointer to member may not be transformed to a membername

AID0546 Right operand is no 'pointer to member' type

Meaning
Right operand is no 'pointer to member' type

AID0547 Left operand does not refer to a class object

Meaning
Left operand does not refer to a class object.

AID0548 Operand refers to class types which are not compatible

Meaning
Left and right operand of the ->* or .* operator refer to class types which are not compatible.

AID0549 Class of pointer to member has ambiguous subobjects

Meaning
Class of pointer to member has ambiguous subobjects in class of left operand.

AID0550 sizeof-/&-selector may not be applied to HLL/LL-operands.

Meaning
sizeof-/-& selectors have to have pure symbolic expressions as arguments and not any HLL/LL-transitions.

AID0551 sizeof-Operator may not be applied to functions.

Meaning
According to C/C++ semantics sizof may not be applied to functions

AID0552 syntax-error by alias-name/prequalifiction (CMD: (&00))

Meaning
Substitution of alias-name or pre-qualifiction by values causes syntactically incorrect command.

AID0553 Partial page(s) from address (&00) to (&01) not accessible

Meaning
Accessing a not allocated 4K half of an 8K SPARC page does not necessarily cause a program interrupt on SPARC HSI

AID0554 Change on register %g0 not allowed.

Meaning
Change of %g0 not allowed.

AID0555 Specified CCS-name not supported

Meaning
XHCS does not support CCS-name or OSD-Version does not support UNICODE

AID0556 Specified CCS-name not supported by AID

Meaning
AID supports only single byte EBCDIC-codes

AID0557 No even number for UTF16 typ length specification

Meaning
Type UTF16 requires even number as length specification.

AID0558 UTFE type HEX value doesn't end on character boundary

Meaning
Operand with type UTFE has hexadecimal value not terminating on a character boundary.

AID0559 value is not an UTFE-string

Meaning
XHCS has found illegal UTFE-encodings
Interpretation as HEX-string

AID0560 CCS-name of output media unknown to actual XHCS subsystem

Meaning
No conversion to CCS of output media performed.

AID0561 CCS-name of output media not supported by AID

Meaning
Supported CCSN by AID: one byte EBCDIC or UTFE

AID0562 Subsystem XHCS not available, no unicode support

Meaning
No conversion UNICODE types possible.
Subsystem XHCS not available.

AID0563 Version of subsystem XHCS does not support UNICODE

Meaning
Version of subsystem XHCS in system is too low for UNICODE support by AID.

AID0564 Version of Subsystem SYSFILE cannot provide CCS-name

Meaning
Actual SYSFILE subsystem version cannot provide SYSOUT/SYSLST EBCDIC. Default value is accepted.

AID0565 LSD information for UNICODE types requires at least LSD Version 10

Meaning
The LSD information (symbolic information) has string data types with unicode coding. that requires at least LSD version 10.

Response
Compiler error, invalid LSD Version

AID0566 Variable boundary of (component of) (&00) is out of valid range.

Meaning
Variable boundary of an array is outside range specified in program (for example OCCURS DEPENDING field in COBOL)

Response
AID assumes array of 0 elements.

AID0567 No redefinition of a char literal by a different char type.

Meaning
Output type of a character literal may not be redefined by a different character type. For example it makes no sense to redefine the bits of character literal C'A' by character type UTF16.

AID0568 Invalid type of argument of string conversion.

Meaning
Type of argument of a string conversion has to be a character type. otherwise an explicit type modification with %C or %UTF16 has to be performed

AID0569 Size of argement of string conversion exceeds 80 characters.

Meaning
Size of argument of string conversion is restricted to 80 characters. Otherwise string has to be abbreviated via explicit length modification.

AID0570 String conversion with substitution by default characters performed.

Meaning
conversion replaces characters of source by a default character because of no equivalent representation in target CCS.

AID0571 Search critera in %FIND command too long.

Meaning
Search criteria of %FIND command is restricted to size of 80 Bytes. X-Literals may not exceed size of 40 Bytes.

14 Appendix

14.1 SDF/ISP commands illegal in command sequences and subcommands

List of BS2000 commands which must not be used in command sequences and subcommands

Command	Function	Manual
ABORT	Abort procedure	[17]
ADD-SHARED-PROGRAMM	declare object modules as shareable	[10]
BEGIN-PROCEDURE	define procedure file attributes	[8]
BREAK	request command mode	[17]
CATEGORY	control configuration workload	[9]
CANCEL-PROCEDURE	terminate (execution of) procedure	[8]
CHANGE-ACCOUNTING-FILE	close current accounting file and create new one	[9]
CHANGE-CONSLOG	close current logging file and create new one	[18]
CHANGE-SERSLOG	close current ISERSLOG file and create new one	[10],[9]
DELON	delete ON command	[17]
END	close spoolin file	[17]
ENDON	terminate ON statement sequence	[17]
ENDP	terminate procedure file	[17]
END-PROCEDURE	terminate procedure file	[8]
EOF	mark end of file for SYSDTA	[17]
ESCAPE	Interrupt procedure run	[17]
EXIT-PROCEDURE	terminate procedure run and return control to procedure file last exited	[8]
HOLD-JOB	halt job	[9],[10]
HOLD-PROCEDURE	halt procedure run to allow command input from display terminal	[8]

Command	Function	Manual
HOLD-RSO	place RSO subsystem in wait state	[10]
HOLD-SPOOL	place SPOOL subsystem in wait state	[10]
HOLD-SPOOLOUT	halt spoolout job	[10]
INTR	send INTR event to program	[17]
LOAD-EXECUTABLE-PROGRAM	load program	[8]
LOAD-PROGRAM	load program	[8]
LOADAID	load AID	[9]
LOGON	initiate job	[17]
MARGIN	modify line length of display terminal	[17]
MODIFY-ACCOUNTING-PARAMETERS	specify accounting records and extensions for accounting file	[9],[10]
MODIFY-CHANNEL-OPTIONS		
MODIFY-SYMBOLIC-PARAMETER		
MODIFY-TASK-CATEGORIES	limit number of active tasks	[10]
ON	conditionally execute command sequence	[17]
PROCEDURE	specify procedure file attributes	[17]
READ-CHANNEL		
RESTART	restart program from checkpoint	[17]
RESTORE		
RESUME-JOB	cancel wait state for user job	[10]
RESUME-PROCEDURE	continue interrupted procedure run	[10]
RESUME-RSO	cancel wait state for RSO subsystem	[10]
RESUME-SPOOL	cancel wait state for SPOOL subsystem	[10]
RFD	assign floppy disk unit for waiting spoolin jobs	[9]
RTI	return to interrupted procedure	[17]
SAVEFILE		
SET-JOB-STEP	terminate spin-off	[8]
SET-SPACE-SATURATION-LEVEL	define storage saturation levels a pubset	[10]
SHARE	declare object modules as shareable	[9]
SHOW-ACCOUNTING-STATUS	display information on accounting system	[9],[10]
SHOW-FILE	display file on screen	[17]
SHOW-RSO-STATUS	display information on RSO status	[10]
SHOW-SERSLOG	display information on SERSLOG	[10]

Command	Function	Manual
SHOW-SPOOL-STATUS	display information on SPOOL status	[10]
SKIP	conditional branch (task switch)	[17]
SKIPJV	conditional branch (job variable)	[17]
SKIPUS	conditional branch (user switch)	[17]
SPMGT	manage storage space	[9]
START-ACCOUNTING	activate accounting system	[9],[10]
START-DISKETTE-INPUT	assign floppy disk drive for spoolin	[10]
START-EXECUTABLE-PROGRAM	load and start program	[8]
START-PROGRAM	load and start program	[8]
START-RSO	load and start RSO subsystem	[9]
START-SERSLOG	activate SERSLOG	[9],[10]
START-SPOOL	load and initialize SPOOL subsystem	[18]
STOP-ACCOUNTING	deactivate accounting system	[9],[10]
STOP-RSO	terminate RSO subsystem	[10]
STOP-SERSLOG	terminate SERSLOG	[10]
STOP-SPOOL	terminate SPOOL subsystem	[10]
WAIT	specify conditional waiting time (batch job)	[17]
WHEN	set conditional halt for batch job (user switch)	[17]
WRITE-CHANNEL		

List of ISP commands terminating a loaded program in AID command sequences and subcommands

Command	Function	Manual
CALL	Invoke a CALL procedure	[17]
DO	Invoke a DO procedure	[17]
EXECUTE	Load and start a module	[17]
LOAD	Load a module	[17]
LOGOFF	Terminate a job	[17]

14.2 Operands described for the last time

This AID version is the last in which the operands *AS output-type* of the %DISPLAY command (debugging on machine code level only) and *control* of the %ON command (both machine-oriented and symbolic debugging) are described.

14.2.1 Operand "AS output-type"

```
%DISPLAY    {data [AS output-type]}{,...}    [medium-a-quantity][,...]
```

The %DISPLAY command offers the *AS output-type* option. This operand follows the associated *data* operand whose output type is to be modified. Without this operand it is possible to change the output type with a type modification (see section “Type modification” on page 86).

```
output-type
```

AID assigns each address operand a type which determines how a particular (type-dependent) set of bytes in memory is to be interpreted (storage type) and how its value is to be output (output type). Each storage type has its corresponding output type (see section “General storage types” on page 109 and section “Storage types for interpreting machine instructions” on page 110). Through an explicit *output-type* specification this assignment can be modified.

output-type can assume the following values:

HEX hexadecimal
DEC numeric (signed decimal)
CHAR character
BIN binary
DUMP dump (hexadecimal and character)

14.2.2 Operand "control" with %ON

```
%ON         event            [<subcmd>]          [control]
```

control is specified as the last operand in %ON, i.e. after *event* and *subcmd*.

control

Defines whether *event* is to be deleted after the n-th occurrence and whether AID then expects input of new commands. If the *control* operand is omitted, AID uses the default values 65535 (for *n*) and K (*KEEP*).

```
control-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

ONLY  n [{ K }]
        { S }
        { C }

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

n Number with a value 1 ≤ *n* ≤ 65535.

Specifies at which occurrence of *event* the other declarations of this *control* operand are to be executed.

K *event* is not deleted (KEEP).

Program execution is interrupted and AID expects command input.

S *event* is deleted (STOP).

Program execution is interrupted and AID expects command input.

C *event* is deleted (CONTINUE).

Program execution is not interrupted.

14.2.3 Linkage using TSOSLNK

During static linkage with TSOSLNK it must always be ensured that the ESD records are included (LINK-SYMBOLS statement).

The SYMTEST operand in the PROGRAM statement controls the handling of LSD records from object modules (see TSOSLNK [13]). The overview below covers only those two operand values which support the use of symbolic names and of statement lines during debugging.

```
PROGRAM...,SYMTEST =        { ALL }
                            { MAP }
```

ALL The LSD records are transferred from the object module to the load unit, but the linkage editor does not check whether the object module being processed actually contains LSD records.

There are thus two options for program loading:

1. The program is loaded with LSD records.
2. The program is loaded without LSD records. If the object module is stored in a PLAM library, this library can be announced to AID via %SYMLIB and AID can dynamically load the LSD records as required.

MAP From the ESD, the linkage editor creates an object structure list which is included in the load unit. This information can be used to trace call hierarchies. LSD records are not linked in, even if they are available in the object module. AID can, however, dynamically load LSD records for symbolic debugging if required.

When linking prelinked modules (TSOSLNK statement MODULE), there is no possibility of incorporating LSD records in the load unit. Moreover, important information in the ESD is lost. If the object modules with the LSD records are in a PLAM library, which is opened with %SYMLIB, AID is able to dynamically load the LSD records and symbolic debugging is possible, subject to the following constraint: if the interrupt point is located in a module for which there is no LSD, not even in the PLAM library, the data and statements from modules with LSD cannot be referenced symbolically either.

Example

```
/START-PROGRAM FROM-FILE=$TSOSLNK
 PROG EXAMPLE, FILENAM=EXAMPLE.FOR1, SYMTEST=ALL
 INCLUDE *
 RESOLVE, FOR1MODLIB
 END
```

Program EXAMPLE is linked from the temporary object module file and stored, with the LSD records, in the file EXAMPLE.FOR1. All unsatisfied external references are to be satisfied via autolink from the library FOR1MODLIB.

14.3 Event codes

The assignment of interrupt events and event codes to the STXIT event classes is shown in the table below:

STXIT event class	interrupt event	Event code
Program error	illegal SVC illegal operation code Data error Exponent overflow Divide error Significance error Exponent underflow Decimal overflow Fixed-point overflow	X' 04' X' 58' X' 60' X' 64' X' 68' X' 6C' X' 70' X' 74' X' 78'
Interval timer for CPU time	"SETIC interval" expired for CPU time	X' 20'
Interval timer for CPU time	"SETIC interval" expired for real time	X' A0'
End program runtime	End of program runtime	X' 80'
unrecoverable program error	Privileged SVC Access to a non-existent memory page Privileged operation Address error XA error (incorrect addressing mode) Realtimer (Condition Error) Alignment error Validation error unrecoverable vector processor error	X' 08' X' 48' X' 54' X' 5C' X' 9C' X' A4' X' AC' X' B0' X' B4'
communication to the program	Command	X' 44'
ESCPBRK	BREAK/ESCAPE (via keys)	X' 84'
ABEND	System error, performance loss START-EXECUTABLE-PROGRAM, LOAD-EXECUTABLE-PROGRAM, ABEND, LOGOFF, CANCEL-JOB Address translation error due to hardware fault Hardware fault (CPU) forced unloading of a subsystem (system management)	X' 88' X' 8C' X' 94' X' A8' X' B8'
Program termination	TERM CMD	X' 90' X' 98'
SVC interrupt	SVC call of a specified SVCs	X' 50'
Hardware fault	Input/output error in data-in-virtual technology	X' 28'

Glossary

addressing mode

AID assumes the addressing mode of the test object (either 24-bit or 31-bit addresses). AID can also be used for testing programs that were linked from modules with differing addressing modes. The system information field %AMODE always shows the current addressing mode. The addressing mode can be changed via `%MOVE %MODE{24|31} INTO %AMODE` and queried via `%DISPLAY %AMODE`.

address operand

This is an operand used to address a memory location or a memory area. Virtual addresses, data names, statement names, source references, keywords, complex memory references, C qualifications (debugging on machine code level) or PROG qualifications (symbolic debugging) may be specified. The memory location/area is situated either in the loaded program or in a memory dump in a dump file.
If a name has been assigned more than once in a user program and thus no unique address reference is possible, area qualifications or an *identifier* (COBOL) can be used to assign the name unambiguously to the desired address.

AID default address interpretation

Indirect addresses, i.e. addresses preceding a pointer operator, are interpreted according to the currently valid addressing mode by default. %AINT can be used to deviate from the default address interpretation and to define whether AID is to use 24-bit or 31-bit addresses in indirect addressing.

AID input files

These are files required by AID for the execution of AID functions, as opposed to input files used by the program. AID processes disk files only.
AID input files include:

1. dump files containing memory dumps (%DUMPFILE)
2. PLAM libraries containing object modules; if the library has been assigned using the %SYMLIB command, AID can dynamically load the LSD records.

AID literals

AID provides the user with both alphanumeric and numeric literals (see chapter “AID literals” on page 101):

```
{C'x...x' | 'x...x'C | 'x...x'| U'x...x'}      Character literal
{X'f...f' | 'f...f'X}                         Hexadecimal literal
{B'b...b' | 'b...b'B}                         Binary literal
[{±}]n                                        Integer
#'f...f'                                      Hexadecimal number
[{±}]n.m                                      Decimal number
[{±}]mantisseE[{±}]exponent                   Floating-point number
```

AID output files

These are files to which the output of the %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP and %TRACE commands may be written. The files are referenced in the output commands via their link names F0 through F7 (see %OUT and %OUTFILE).
The REP records are written to the file assigned to link name F6 (see %AID REP=YES and %MOVE).

There are three ways of creating an output file:

1. /%OUTFILE command with link name and file name
2. /FILE command with link name and file name
3. AID issues a FILE macro with the file name AID.OUTFILE.Fn for a link name to which no file name has been assigned.

An AID output file always has the format FCBTYPE=SAM, RECFORM=V and OPEN=EXTEND.

AID standard work area

This is the non-privileged area of the virtual memory in a user task, which is occupied by the program and all its connected subsystems.
In conjunction with symbolic debugging, this is the current program segment of the program which has been loaded.
If no declaration has been made via %BASE and no base qualification has been specified, the AID standard work area applies by default.

AID work area

The AID work area is the address space in which memory references can be accessed without specification of a base qualification. It comprises the non-privileged part of virtual memory in the user task, which is occupied by the program and all its connected subsystems, or the corresponding area in a memory dump.
Using the %BASE command, you can shift the AID work area from the loaded program to a memory dump, or vice versa. You may deviate from the AID work area in a command by specifying a base qualification in the address operand.

area check

For byte offset and length modification operations and for *receiver* in the %MOVE command, AID checks whether the area limits of the referenced memory objects are exceeded, in which case an error message is issued.

area limits

Each memory object is assigned a specific area, which is defined by the address and length attributes in the case of data names and keywords. For virtual addresses, the area limits are between V'0' and the last address of the virtual memory (V'7FFFFFFF'). In area qualifications, the area limits are derived from the start and end addresses of the program segment thus identified (see chapter "Addressing in AID" on page 65).

area qualification

These qualifications are used to identify part of the work area. If an address operand ends with one of these qualifications, the command is effective only in the part that is identified by the last qualification. An area qualification delimits the active area of a command, or makes a data name or statement name unique within the work area, or allows a name to be reached that would otherwise not be addressable at the current interrupt point.

attributes

Each memory object has up to six attributes:
address, name (opt), content, length, storage type, output type.
The address, length and storage type can be accessed using selectors. AID uses the name to locate all the associated attributes in the LSD records so as to be able to correctly interpret the associated memory object.
Address constants and constants from the source program have only up to five attributes:
name (opt), value, length, storage type, output type.
They have no address. When a constant is referenced, AID does not access a memory object but merely inserts the value of the constant.

base qualification

This is the qualification designating either the loaded program or a memory dump in a dump file. It is specified via E={VM | Dn}.
The base qualification may be globally declared by means of %BASE or specified in the address operand for a single memory reference.

character conversion functions

AID provides two functions for character conversion, %C() and %UTF16(). The %UTF16() function converts strings from a 1-byte EBCDIC encoding to UTF16 encoding; the %C function performs conversion in the other direction.

command mode

The term "command mode" in the AID manuals designates the EXPERT mode of the SDF command language. Users who are working in a different mode (GUIDANCE={MAXIMUM | MEDIUM | MINIMUM | NO}) should select the EXPERT mode by issuing the command MODIFY-SDF-OPTIONS GUIDANCE=EXPERT when they wish to enter AID commands.
AID commands are not supported by SDF syntax, i.e.

– operands cannot be entered via menus and

– AID issues error messages but does not offer a correction dialog.

The system prompt for command input in EXPERT mode is "/".

command sequence

Several commands separated by semicolons (;) form a command sequence, which is processed from left to right. Like a subcommand, a command sequence may contain both AID and BS2000 commands. Certain commands are not permitted in command sequences: this refers to the AID commands %AID, %BASE, %DUMPFILE, %HELP, %OUT, %QUALIFY and the BS2000 commands listed in the appendix.
If a command sequence contains one of the commands for runtime control, the command sequence is aborted at that point and the program is started (%CONTINUE, %RESUME, %TRACE) or halted (%STOP). Any subsequent commands in the command sequence are not executed.

compilation unit

A compilation unit is part of a program that has been compiled as a unit. The term program unit is used for this in Fortran. A compilation unit can be referenced with the S qualification.

constant

A constant represents a value which is not accessible via an address in program memory.
The term "constants" includes the constants defined in the source program, the results of length selection, length function and address selection, as well as the statement names and source references.
An address constant represents an address. This subset includes statement names, source references, and address selection results. An address constant in a complex memory reference must be followed by a pointer operator (->).

CSECT information

Information contained in the object structure list.

current call hierarchy

The current call hierarchy represents the status of subprogram nesting at the interrupt point. It ranges from the subprogram level at which the program was interrupted, to the hierarchically intermediate subprograms exited by means of CALL statements, to the main program.
The hierarchy is output using the %SDUMP %NEST command.

current CSECT

This is the CSECT in which the program was interrupted. Its name is output in the STOP message.

current program

The current program is the one which is loaded in the task in which the user enters AID commands.

current program segment

This is the program segment in which the program was interrupted. Its name is output in the STOP message.

dataname

This operand stands for all names assigned for data in the source program. *dataname* can be used to address variables, constants, structures, tables and structure/table items in symbolic debugging. Items in structures/tables can be referenced just like in the relevant programming language by means of an *identifier* or an *index*.

data type

In accordance with the data type declared in the source program, AID assigns one of the following AID storage types to each data item:

– binary string (≙ %X)

– character (≙ %C or %UTF16)

– numeric (not all data types treated numerically in the relevant programming languages correspond to a numeric storage type in AID; see the individual language-specific AID manuals [2]-[6]).

The allocated storage type determines how a data item is output by %DISPLAY, transferred/overwritten by %MOVE or %SET, and compared in the condition of a subcommand.

ESD/ESV

The External Symbol Dictionary (OMs) / External Symbols Vector (LLMs) lists the external references of a module. It is generated by the compiler and contains, among other things, information on CSECTs, DSECTs and COMMONs. The linkage editor accesses the ESD when creating the object structure list.

global settings

AID offers commands which serve to adapt the behavior of AID to particular user requirements, save input efforts and facilitate addressing. The global presettings made via these commands are valid throughout the debugging session. See %AID, %AINT, %BASE and %QUALIFY.

main program

In this manual main program is used as a collective term for the program (COBOL), the function (main in C++/C) or the external procedure (PL/I) which is started by the system when the program starts.

index

An index is part of the address operand. An index defines the position of a vector element. It may be specified in the same way as in the programming language or by means of an arithmetic expression from which AID calculates the value of the index.

input buffer

AID has an internal input buffer. If this buffer cannot accommodate a command input, the command is rejected with an error message indicating that the command is too long. The required operation must be divided between two commands to enable AID to execute it.

interrupt point

The address at which a program is interrupted is known as the interrupt point. The STOP message reports the address and the program segment where the interrupt point is located. The program is then continued there. For COBOL85 and FOR1 programs a different continuation address can be specified via %JUMP.

LIFO

Last In First Out principle. If statements from different inputs concur at a test point (%INSERT) or upon occurrence of an event (%ON) the statements entered last are processed first (see section “Chaining” on page 60).

localization information

%DISPLAY %HLLOC(memref) for the symbolic level and %DISPLAY %LOC(memref) for the machine code level can be used to output the static program nesting for a specified memory location.
Conversely, %SDUMP %NEST outputs the dynamic program nesting, i.e. the call hierarchy for the current interrupt point.

LSD

The List for Symbolic Debugging (LSD) stores the data/statement names defined in the module as well as the compiler-generated source references. The LSD records are created by the compiler and used by AID to retrieve the information required for symbolic addressing.

memory object

A memory object is constituted by a certain number of bytes in memory. At the program level, this comprises the program data (provided it has been assigned a memory area) and the program code. All registers, the program counter and all other areas which can only be referenced via keywords are likewise memory objects.
Any constants defined in the program, the statement labels, source references, address selection results, length selection/function and AID literals do not constitute memory objects, however, because they represent a value which cannot be changed.

memory reference

A memory reference addresses a memory object. There are two types of memory reference: simple and complex.
Simple memory references are virtual addresses, names whose address AID can fetch from the LSD information, and keywords. Statement names and source references are allowed as memory references in the AID commands %CONTROLn, %DISASSEMBLE, %INSERT, %JUMP, %REMOVE and %TRACE although they are merely address constants.

Complex memory references constitute instructions for AID indicating how to calculate the desired address and which type and length are to apply. The following operations may occur in a complex memory reference: byte offset, indirect addressing, type/length modification and address selection.

monitoring

%CONTROLn, %INSERT and %ON are monitoring commands. When a command or statement of the selected group (%CONTROLn) or the defined program address (%INSERT) is encountered in the program sequence or if the selected event occurs (%ON), program execution is interrupted and the specified subcommand is processed by AID.

name range

Comprises all the data names stored for a program segment in the LSD records.

object structure list

The linkage editor creates the object structure list on the basis of the External Symbol Dictionary (ESD) if the default setting SYMTEST=MAP applies or SYMTEST=ALL has been specified.

output type

Attribute of a memory object; defines how the memory contents are to be output by AID. Each storage type is assigned an output type. In chapter “Keywords” on page 109 the AID-specific storage types are listed with their respective output types. A similar assignment applies for the data types in the various programming languages. A type modification in %DISPLAY and %SDUMP causes the output type to be changed.

program segment

This is a general term for any program part which can be addressed by means of an area qualification. In the various programming languages a program segment is known under different designations, which are described in the language-specific AID manuals.

program state

AID makes a distinction between three program states which the program being tested may assume:

program state

AID makes a distinction between three program states which the program being tested may assume:

1. The program has stopped.
%STOP or actuation of the K2 key interrupts a program which is executing. The program is also interrupted if a %TRACE has been fully processed. The task is in command mode, i.e. the user may enter commands.

2. The program is running without tracing.
%RESUME starts or continues a program. %CONTINUE has the same effect; but if a %TRACE has not yet finished, issuing a %CONTINUE command will continue not only the program but also tracing.

3. The program is running with tracing.
%TRACE starts or continues a program. The program sequence is logged in accordance with the declarations in the %TRACE command. %CONTINUE has the same effect if a %TRACE is still active.

program unit

This is a term used in Fortran for that which is referred to as a compilation unit in other programming languages. A program unit can be addressed with the S qualification.

qualification

A qualification addresses a memory reference which is not in the AID work area or is outside the current main program or subprogram or is not unique therein. The base qualification specifies whether the memory reference is located in the loaded program or in a dump.
An area qualification specifies the program segment containing the memory reference.
If an operand qualification is found to be superfluous or contradictory it is ignored. This is the case, for example, if an area qualification is specified for a virtual address.

source reference

A source reference designates an executable statement. It is specified as S'number/name'.
Knumber/name k is generated by the compiler and stored in the LSD records.

statement name

A statement name is a name, assigned in a source program, via which an executable statement can be referenced in AID. Such names are labels or names of main programs or subprograms. An address constant containing the address of the first statement after the label or in the main program or

subprogram is stored in the LSD records for this purpose. To be more precise it is the address of the first instruction that was generated for the first executable statement after a label or in the main program or subprogram.

storage type

This is the data type that was either defined in the source program or selected via type modification. AID knows the storage types %X, %C, %P, %D, %F, %A. See %SET and chapter “Addressing in AID” on page 65 and chapter “Keywords” on page 109.

subcommand

A subcommand is an operand of the monitoring commands %CONTROLn, %INSERT and %ON. A subcommand consists of a command section which may optionally be preceded by a name and a condition. The command section may consist of a single command or a command sequence and may contain both AID and BS2000 commands. Each subcommand has an execution counter. See chapter “Subcommand” on page 49 on how an execution condition is formulated, how a name and an execution counter are assigned and referenced, and which commands are not permitted within subcommands.
The command section of a subcommand is executed if the monitoring condition (*criterion*, *test-point*, *event*) of the corresponding command is satisfied and any execution condition defined in the subcommand has been met.

subprogram

In this manual subprogram is used as a collective term for functions (C D++ d/C, Fortran, COBOL), procedures (PL/I) or programs (COBOL) which are subordinate to the main program in the call hierarchy.

tracing

%TRACE is a tracing command. It defines which and how many commands or statements are to be logged. In the default case, program execution can be viewed on the screen.

update dialog

The %AID CHECK=ALL command initiates the update dialog, which takes effect when a %MOVE or %SET is executed. AID queries during the dialog whether updating of the memory contents really is to take place. If N is entered as a response, no modification is carried out; if Y is entered, AID performs the transfer.

user area

Area in virtual memory which is occupied by the loaded program with all its connected subsystems. Corresponds to the area represented by the keywords %CLASS6, %CLASS6ABOVE and %CLASS6BELOW.

Related publications

The manuals are available as online manuals, see *http://manuals.fujitsu-siemens.com*, or in printed form which must be paid and ordered separately at *http://FSC-manualshop.com*.

[1] **AID** (BS2000/OSD)
Debugging on Machine Code Level
User Guide

Target group
Programmers and debuggers
Contents
– Description of the AID commands for debugging on machine code level
– Sample application
The %SHOW, %SDUMP and %NEST commands are described, plus context COMMON qualification and (on ESA systems) the ALET/SPID qualifications for data spaces. Additional keywords have been included.

[2] **AID** (BS2000)
Advanced Interactive Debugger
Debugging of COBOL Programs
User Guide

Target group
COBOL programmers
Contents
– Description of the AID commands for symbolic debugging of COBOL programs
– Sample application
Applications
Testing of COBOL programs in interactive or batch mode

[3] **AID** (BS2000)
Advanced Interactive Debugger
Debugging of FORTRAN Programs
User Guide

Target group
FORTRAN programmers
Contents
– Description of the AID commands for symbolic debugging of FORTRAN programs
– Sample application
Applications
Testing of FORTRAN programs in interactive or batch mode

[4] **AID** (BS2000)
Advanced Interactive Debugger
Debugging of PL/I Programs
User Guide

Target group
PL/I programmers
Contents
– Description of all the AID commands available for the symbolic debugging of PL/I programs
– Sample application

[5] **AID** (BS2000)
Advanced Interactive Debugger
Debugging of ASSEMBH Programs
User Guide

Target group
Assembly language programmers
Contents
– Description of the AID commands for symbolic debugging of ASSEMBH-XT programs
– Sample application
Applications
Testing of ASSEMBH-XT programs in interactive or batch mode

[6] **AID** (BS2000/OSD)
Debugging of C/C++ Programs
User Guide

Target group
This manual is intended for C/C++ programmers.
Contents
The manual contains a description of the AID commands and the C/C++-specific address operands for symbolic debugging of C/C++ programs. It contains information on debbugging under POSIX and on RISC systems, and comprehensive applications examples.
Application
Debugging of C/C++ programs in interactive and batch mode

[7] **AID** (BS2000)
Advanced Interactive Debugger
Ready Reference
Target group
Programmers in BS2000
Contents
– Debugging of programs written in ASSEMBH, C/C++, COBOL, FORTRAN, PL/I and ar machine code level
– Summary of the AID commands and operands
– %SET tables
Applications
Testing of programs in interactive or batch mode

[8] **BS2000/OSD-BC**
Commands, Volumes 1 - 5
User Guide

Target group
This manual is addressed to nonprivileged users and systems support staff.
Contents
Volumes 1 through 5 contain the BS2000/OSD commands ADD-... to WRITE-... (basic configuration and selected products) with the functionality for all privileges. The command and operand functions are described in detail, supported by examples to aid understanding. An introductory overview provides information on all the commands described in Volumes 1 through 5.
The Appendix of Volume 1 includes information on command input, conditional job variable expressions, system files, job switches, and device and volume types.
The Appendix of Volumes 4 and 5 contains an overview of the output columns of the SHOW commands of the component NDM. The Appendix of Volume 5 contains additionally an overview of all START commands.
There is a comprehensive index covering all entries for Volumes 1 through 5.

[9] **BS2000/OSD-BC**
Introductory Guide to Systems Support
User Guide

Target group
This manual is addressed to BS2000/OSD systems support staff and operators.
Contents
The manual covers the following topics relating to the management and monitoring of the BS2000/OSD basic configuration: system initialization, parameter service, job and task control, memory/device/system time/user/file/pubset management, assignment of privileges, accounting and operator functions.

[10] BS2000
System Administrator Commands (SDF Format)
Reference Manual

Target group
BS2000 system administrators
Contents
SDF commands for the system administrator
Applications
System administration

[11] **BS2000/OSD-BC**
Executive Macros
User Guide

Target group
The manual addresses all BS2000/OSD assembly language programmers.
Contents
The manual contains a summary of all Executive macros, detailed descriptions of each macro with notes and examples, including job variable macros, and a comprehensive general training section.

[12] **LMS** (BS2000)
SDF Format
User Guide

Target group
BS2000 users.
Contents
Description of the statements for creating and managing PLAM libraries and the members these contain.
Frequent applications are illustrated with examples.

[13] BS2000
TSOSLNK
User Guide

Target group
Software developers
Contents
– Statements and macros of the linkage editor TSOSLNK for linking load modules and prelinked modules
– Commands of the static loader ELDE

[14] **BINDER**
Binder in BS2000/OSD
User Guide

Target group
Software developers
Contents
The manual describes the BINDER functions, including examples. The reference section contains a description of the BINDER statements and BINDER macro.

[15] **BLSSERV**
Dynamic Binder Loader / Starter in BS2000/OSD
User Guide

Target group
This manual is intended for software developers and experienced BS2000/OSD users
Contents
It describes the functions, subroutine interface and XS support of the dynamic binder loader DBL as a component of the BLSSERV subsystem, plus the method used for calling it.

[16] **SDF** (BS2000/OSD)
Introductory Guide to the SDF Dialog Interface
User Guide

Target group
BS2000/OSD users
Contents
This manual describes the interactive input of commands and statements in SDF format. A Getting Started chapter with easy-to-understand examples and further comprehensive examples facilitates use of SDF. SDF syntax files are discussed.

[17] BS2000
User Commands (ISP Format)
User Guide

Target group
BS2000 users (nonprivileged)
Contents
– All BS2000 system commands in alphabetical order with detailed explanations and examples
– The following products are dealt with: BS2000-GA, MSCF, JV, FT, TIAM
Applications
BS2000 interactive/batch mode, procedures

[18] **BS2000/OSD-BC V1.0**
System Operator's Guide
User Guide

Target group
This manual addresses operators at installations of the BS2000/OSD operating system.
Contents
It describes the operator's functions and responsibilities as well as the commands available for this purpose at the operator terminal. The following items are dealt with:
– System initialization and termination (startup types, shutdown)
– Commands in alphabetical order
– Device management (reconfiguration, resource allocation, volume monitoring, NDM handling, duplex reconfiguration)
– Tools and methods for facilitating system operation
– Memory dumps (SLED)
– Messages and responses in the event of saturation states

[19] **Fortran90 V1.0**
BS2000/OSD
Fortran90 Compiler
User Guide

Target group
Fortran90 users in BS2000
Contents
This manual describes all the activities involved in generating an executable Fortran90 program: compiling, linking, loading, debugging. It includes programming notes and further information on file processing and language linkage.

[20] BS2000
Programmiersystem *
Technische Beschreibung
(Programming System, Technical Description)

Target group
- BS2000 users with an interest in the technical background of their systems (software engineers, systems analysts, computer center managers, system administrators)
- Computer scientists interested in studying a concrete example of a general-purpose operating system

Contents
Functions and principles of implementation of
- the linkage editor
- the static loader
- the Dynamic Linking Loader
- the debugging aids
- the program library system

Order number
U3216-J-Z53-1

Index

Fujitsu Siemens Computers GmbH
User Documentation
81730 München
Germany

Fax: 0 700 / 372 00001

e-mail: manuals@fujitsu-siemens.com
http://manuals.fujitsu-siemens.com

Comments
Suggestions
Corrections

Submitted by

Comments on AID V3.2A
Core Manual

Information on this document

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.

This document from the document archive refers to a product version which was released a considerable time ago or which is no longer marketed.

Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.

Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format *…@ts.fujitsu.com*.

The Internet pages of Fujitsu Technology Solutions are available at *http://ts.fujitsu.com/*...
and the user documentation at *http://manuals.ts.fujitsu.com*.

Hinweise zum vorliegenden Dokument

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.

Das vorliegende Dokument aus dem Dokumentenarchiv bezieht sich auf eine bereits vor längerer Zeit freigegebene oder nicht mehr im Vertrieb befindliche Produktversion.

Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.

Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form *…@ts.fujitsu.com*.

Die Internetseiten von Fujitsu Technology Solutions finden Sie unter *http://de.ts.fujitsu.com/*..., und unter *http://manuals.ts.fujitsu.com* finden Sie die Benutzerdokumentation.